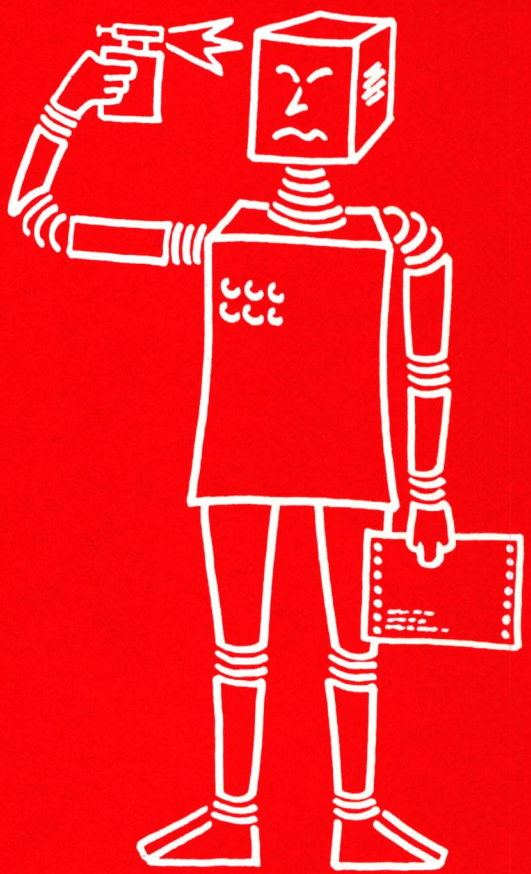
SEKI-PROJEKT

SEKI MEMO

Institut für Informatik III
Universität Bonn
Bertha-von-Suttner-Platz 6
D 5300 Bonn 1, W. Germany

Institut für Informatik I
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe 1, W. Germany



MEMO SEKI-81-08

APE - Benutzerbeschreibung

Ulrich Bartels, Walter Olthoff

Institut für Informatik III
Universität Bonn

A P E

Ein Expertensystem

zur automatischen Programmierung

von abstrakten Spezifikationen

von Datentypen und Algorithmen

Ulrich Bartels

und

Walter Olthoff

Bonn, September 1981

Hier soll ich finden, was mir fehlt?
Soll ich vielleicht in tausend Büchern lesen,
Daß überall die Menschen sich gequält,
Daß hie und da ein Glücklicher gewesen? –
Was grinsest du mir, hohler Schädel, her?
Als daß dein Hirn, wie meines, einst verwirret
Den lichten Tag gesucht und in der Dämmrung schwer,
Mit Lust nach Wahrheit, jämmerlich geirret.

.....

Geheimnisvoll am lichten Tag
Läßt sich Natur des Schleiers nicht berauben,
Und was sie deinem Geist nicht offenbaren mag,
Das zwingst du ihr nicht ab mit Hebeln und mit Schrauben.

.....

Weit besser hätt' ich doch mein Weniges verpraßt,
Als mit dem Wenigen belastet hier zu schwitzen!
Was du ererbt von deinen Vätern hast,
Erwirb es, um es zu besitzen.
Was man nicht nützt, ist eine schwere Last,
Nur was der Augenblick erschafft, das kann er nützen.

(Goethe, Faust I)

ÜBERBLICK

1. EINLEITUNG / DAS APE-SYSTEM

2. KONZEPTE ABSTRAKTER SPEZIFIKATIONEN

   2.1. ABSTRAKTE DATENTYPEN
   2.2. PARAMETRISIERUNGEN
   2.3. ABSTRAKTE ALGORITHMEN
   2.4. SPEZIFIKATIONSSPRACHEN

3. PARAMETRISIERTE ABSTRAKTE DATENTYPEN

   3.1. DAS ADTCOMP-PARAMETRISIERUNGSKONZEPT FÜR ADT'EN
   3.2. BEISPIEL: IMPLEMENTIERUNG UND BENUTZUNG
                  DES PADT OFFENELISTE-INTEGER

4. ABSTRAKTE ALGORITHMEN

   4.1. ÜBERBLICK ÜBER DAS ALGCOMP-SYSTEM
   4.2. NUMERISCHE ALGORITHMEN
   4.3. SUCHALGORITHMEN
   4.4. SORTIERALGORITHMEN
   4.5. ANDERE ALGORITHMEN
   4.6. ANWENDUNGSBEISPIEL: PV-SYSTEM

5. SYSTEMBESCHREIBUNG APE

   5.1. SYSTEMBESCHREIBUNG ADTCOMP
   5.2. SYSTEMBESCHREIBUNG ALGCOMP
   5.3. ANWENDUNGSFUNKTIONEN
   5.4. ALLGEMEIN VERWENDBARE FUNKTIONEN
   5.5. FEHLERBEHANDLUNG

6. LITERATUR

ANHANG 1: FUNKTIONSVERZEICHNIS

ANHANG 2: STATISTISCHE DATEN ZUR REGELANWENDUNG
          UND IMPLEMENTIERUNG

ANHANG 3: DATEIVERZEICHNIS

1. Einleitung

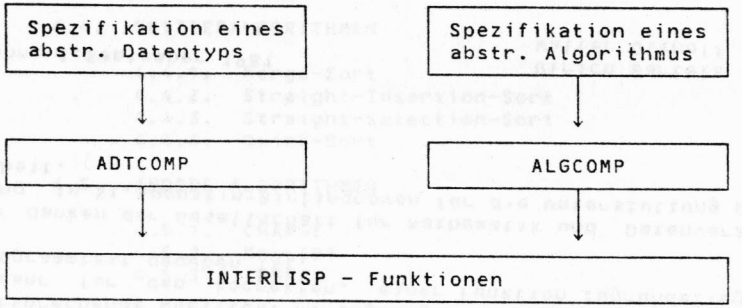
Die vorliegende Arbeit gibt eine Beschreibung des APE-Systems, eines automatischen Programmiersystems zur Erzeugung von Programmen aus abstrakten Spezifikationen von Datentypen und Algorithmen. Die hinter APE (Automatic Programming Expert) stehende Idee ist, Softwareentwicklung ausgehen zu lassen von einer möglichst abstrakten Ebene, auf der insbesondere auf Repräsentationsfragen von Objekten in einer Rechenanlage weitgehendst verzichtet wird. Diese sogenannten abstrakten Spezifikationen sind dann schrittweise zu verfeinern zu konkreten Implementierungen; das APE-System automatisiert diese Arbeit und bildet abstrakte Spezifikationen in Objekte der Zielsprache INTERLISP ab.

Der auch von einem menschlichen Programmierer eingeschlagene Weg in der Erzeugung eines konkreten Programms besteht gewöhnlich aus einer Anzahl von nur bedingt untereinander abhängigen Design-Entscheidungen, die meist auf Erfahrung, Training und individueller An/Einsicht basieren. Viele Enscheidungen dabei sind isoliert derart, daß sie auf verschiedene und sich nicht überschneidenden Wissensgebieten aufbauen, woraus folgt, daß man das bei einer Implementierung zur Anwendung kommende Programmierwissen in teilweise disjunkte Mengen gliedern kann. Dieser Gedanke hat es nahegelegt, die von dem zu konstruierenden System zu verrichtende Arbeit in Form eines Expertensystems durchzuführen. Expertensysteme enthalten kleine 'Portionen' des einem menschlichen Experten (Programmierer) innewohnenden Wisssens (Programmierwissen) in Form sogenannter Produktionsregeln. Mengen dieser Regeln werden zu Regelbasen zusammengefaßt, und zusammen mit einer geeigneten Kontrollstruktur für die Arbeit mit Regeln erhält man ein Produktionssystem [DK 77], das in dieser Struktur auch dem APE-System zugrunde liegt.

Produktionssysteme sind (unter anderem) immer dann von Vorteil, wenn eine Entwicklungsaufgabe experimentell durchgeführt wird. Man beginnt mit einer Anfangsmenge von Regeln (im vorliegenden Fall: Portionen Programmierwissens) und versucht, mit ihnen ein Problem (Implementierung) bezüglich eines typischen Beispiels (Spezifikation eines Standarddatentyps/algorithmus) zu lösen. Wenn der Versuch mißlingt, sind die Gründe zu analysieren und meistens Veränderungen an Regeln und Regelbasen oder der Art der Kontrolle vorzunehmen mit dem Ziel, die Mangel zu beheben. Danach kann zu anderen Beispielen (andere Spezifikationen) übergegangen werden, so daß im weiteren Verlauf in den Regeln codiertes Wissen angehäuft wird und tatsächlich ein 'Experte' entsteht.

Das APE-System besteht gegenwärtig aus zwei Subsystemen:

- ADTCOMP erzeugt aus abstrakten algebraischen Spezifikationen von Datentypen INTERLISP-ausführbare Funktionsbündel.
- ALGCOMP erzeugt aus abstrakten Algorithmen, die in Form von Term-Ersetzungsregeln formuliert und aus Operationssymbolen derjenigen Datentypen aufgebaut sind, die ADTCOMP erzeugt, eine INTERLISP-ausführbare Funktion.



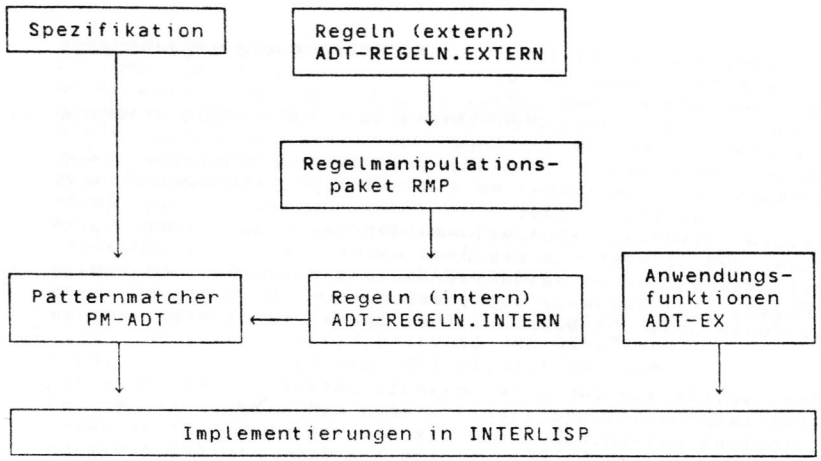
Beide Subsysteme stellen im wesentlichen Funktionen dar, die abstrakte Spezifikationen in INTERLISP abbilden.

Die interne Struktur beider Systeme differiert etwas, da für das ADTCOMP-System z.Zt. bereits ein sogenanntes Regelmanipulationspaket existiert, das z.B. jede einzelne Regel aus einer externen (lesbareren) Darstellung in eine interne überführt, einzelne Regeln verändern und Systemvariablen setzen kann. Ein analoges Paket für das ALGCOMP-System hat sich auf Grund des augenblicklich geringeren Umfangs der Regelmengen nicht als notwendig erwiesen; mit steigender Regelanzahl wird es jedoch unumganglich sein.

Die angedeutete Ebene der INTERLISP-Funktionen differiert ebenfalls: während ALGCOMP für eine Spezifikation eines Datentyps eine Art Bündel von LISP-Funktionen erzeugt, implementiert ALGCOMP die Spezifikation eines Algorithmus als eine Funktion. Diese Unterscheidung ist aber durch die verschiedenen Implementierungsziele einsichtig.

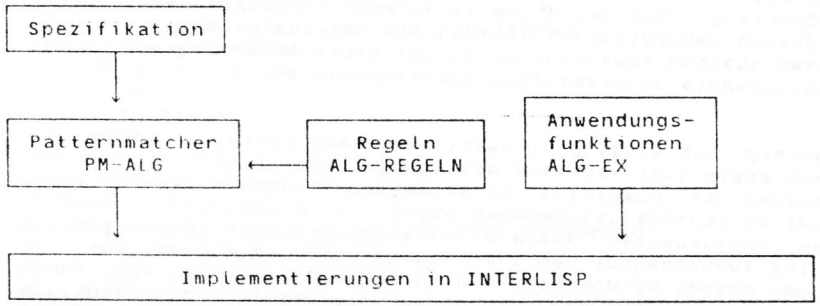
Die beiden Hauptmodule sollen noch kurz etwas detaillierter vorgestellt werden.

Die Komponenten von ADTCOMP sind:



Die Anwendungsfunktionen, die in der Komponente ADT-EX zusammengefaßt sind, dienen zur Laufzeit einer vom ALGCOMP-System erzeugten Implementierung zur korrekten Bearbeitung von Hilfs-, Test- und Fehlerfunktionen, die im erzeugten Code referiert, aber nicht definiert werden.

Das ALGCOMP-System besteht aus folgenden Komponenten:



In beiden Systemen sind die Komponeten PM-ADT und PM-ALG diejenigen Teile des Systems, in denen die Regelauswahl, -anwendung und Kontrolle durchgeführt wird.

Zur Begriffsklärung werden im Kapitel 2 zunächst einige Konzepte abstrakter Spezifikationen und der jeweilige Einfluß auf das APE-System besprochen.
Im 3. Kapitel wird beschrieben, wie parametrisierte abstrakte Spezifikationene durch ADTCOMP implementiert werden. Dieses Kapitel gibt im wesentlichen die Neuerungen wieder, die gegenüber der ADTCOMP-Beschreibung in [EKT 79] durchgeführt wurden, um das Parametrisierungskonzept zu realisieren.
Im 4. Kapitel befindet sich eine ausführliche Erläuterung aller Komponenten des ALGCOMP-Systems. Ferner werden die Implementierungsläufe von 15 abstrakt spezifizierten Algorithmen dokumentiert.
Im 5. Kapitel sind Funktionen und Regeln beider APE-Subsysteme dokumentiert und ausführlich erklärt; in 5.5. werden weitere Informationen über die Fehlerbehandlung im APE-System bereitgestellt.

An dieser Stelle sei noch folgendes erwähnt: ein umfassender Teil dieser Arbeit ist dadurch erzeugt worden, daß Programme, Protokolle und sonstige Texte aus entsprechenden Dateien in die Druckdateien kopiert wurden. Nach dem Kopieren mußten in sehr vielen Fällen per Hand noch Manipulationen an den logischen Sätzen der Druckdateien vorgenommen werden, damit das Druckbild die vorliegende Gestalt bekam. Bei dieser Arbeit ist es vorgekommen, daß der Quelltext fehlerhaft verändert worden ist. Wir haben durch genaues Arbeiten und ein intensives Gegenlesen von Dritten versucht, alle Fehler dieser Art aufzuspüren und zu korrigieren. Dennoch kann es insbesondere in Programmtexten noch derartiges geben. In Fällen, wo offensichtliche Ungereimtheiten auftreten, sollte zur Klärung der entsprechende Quelltext herangezogen werden, da nur dort die Gewähr für den korrekten, einer Funktion zugrundeliegenden Programmtext gegeben ist.

Wir danken der Gesellschaft für Mathematik und Datenverarbeitung in St.Augustin-Birlinghoven für die Unterstützung dieser Arbeit.

Bonn, 1.September 1981

Ulrich Bartels
Walter Olthoff

INHALTSVERZEICHNIS

1. EINLEITUNG / DAS APE-SYSTEM 3

2. KONZEPTE ABSTRAKTER SPEZIFIKATIONEN 11

2.1. ABSTRAKTE DATENTYPEN 11

2.1.1. Algebraische Spezifikationen 12
2.1.2. ADTen im ADTCOMP-System 15

2.2. PARAMETRISIERUNGEN 17

2.2.1. Parametrisierte Spezifikationen 17
2.2.2. Parametrisierungen im ADTCOMP-System 21

2.3. ABSTRAKTE ALGORITHMEN 22

2.4. SPEZIFIKATIONSSPRACHEN 24

2.4.1. ADT-Eingabe 24
2.4.2. Algorithmen-Eingabe 27

3. PARAMETRISIERTE ABSTRAKTE DATENTYPEN 36

3.1. DAS ADTCOMP-PARAMETRISIERUNGSKONZEPT FÜR ADT'EN 36

3.1.1. Erweiterungen von ADTCOMP: Regeln 41
3.1.2. Erweiterungen von ADTCOMP: Funktionen 41

3.2. BEISPIEL: IMPLEMENTIERUNG UND BENUTZUNG DES PADT OFFENELISTE-INTEGER 46

4. ABSTRAKTE ALGORITHMEN 75

4.1. ÜBERBLICK ÜBER DAS ALGCOMP-SYSTEM 75

4.1.1. Ziel des Verfahrens 76
4.1.2. Die Phasen des ALGCOMP-Systems 81

4.1.2.1. Die Einlese-Phase 81
4.1.2.2. Die Algorithmus-Konstruktions-Phase 81
4.1.2.3. Die LISP-Phase 94

4.1.3. ALGCOMP-Pattern-Matcher und Regelbasen 97
4.1.4. Beispiel: Implementierung und Benutzung des Algorithmus Bubble-Sort 104

4.2. NUMERISCHE ALGORITHMEN 122

4.2.1. Fakultätsfunktion 122
4.2.2. Größter gemeinsamer Teiler 128
4.2.3. Kleinstes gemeinsames Vielfaches 134

4.3. SUCHALGORITHMEN 143

4.3.1. Sequential-Search 143
4.3.2. Sequential-Search-on-ordered-Tables 150
4.3.3. Binary-Search 157
4.3.4. FIBONACCIAN-Search 167

4.4. SORTIERALGORITHMEN 182

4.4.1. Merge-Sort 183
4.4.2. Straight-Insertion-Sort 194
4.4.3. Straight-Selection-Sort 207
4.4.4. Quick-Sort 213

4.5. ANDERE ALGORITHMEN 242

4.5.1. CHANGE 242
4.5.2. MAXLIST 255
4.5.3. INDMAX 262

4.6. ANWENDUNGSBEISPIEL: PERSONALDATEN-VERWALTUNGS-SYSTEM 271

5. SYSTEMBESCHREIBUNG APE 300

5.1. SYSTEMBESCHREIBUNG ADTCOMP 300

5.1.1. Funktionen des ADTCOMP-Systems 301

5.1.1.1. Übergeordnete Funktionen 301
5.1.1.2. Untergeordnete Funktionen 313

5.1.1.2.1. Hilfsfunktionen von RADT 313
5.1.1.2.2. Hilfsfunktionen der ->- und =-Phase 316
5.1.1.2.3. Hilfsfunktionen der &-, I- und B-Phase 321
5.1.1.2.4. Hilfsfunktionen der L-Phase 329
5.1.1.2.5. Allgemeine Hilfsfunktionen 331

5.1.1.3. Testfunktionen 373

5.1.2. Regeln des ADTCOMP-Systems 376

5.2. SYSTEMBESCHREIBUNG ALGCOMP 393

5.2.1. Funktionen des ALGCOMP-Systems 393

5.2.1.1. Hauptfunktion zur Algorithmus-Implementierung

und globale Variable 394
5.2.1.2. Funktionen der Einlese-Phase 397

5.2.1.2.1. Einlesefunktion RALGSPEC 397
5.2.1.2.2. Unterfunktionen von RALGSPEC 406

5.2.1.3. Funktionen der Algorithmus-Konstruktions-Phase 430
5.2.1.4. Funtionen der LISP-Phase 463

5.2.2. Regeln des ALGCOMP-System 472

5.3. ANWENDUNGSFUNKTIONEN 501

5.3.1. Anwendungsfunktionen für Datentyp-Implementierungen 501
5.3.2. Anwendungsfunktionen für Algorithmus-Implementierungen 505

5.4. ALLGEMEIN VERWENDBARE FUNKTIONEN 517

5.5. FEHLERBEHANDLUNG 526

6. LITERATUR 529

ANHANG 1: FUNKTIONSVERZEICHNIS 533

ANHANG 2: STATISTISCHE DATEN ZUR REGELANWENDUNG UND IMPLEMENTIERUNG 541

ANHANG 3: DATEIVERZEICHNIS 557

2. Konzepte abstrakter Spezifikationen

2.1. Abstrakte Datentypen

Einer der Ausgangspunkte der Entwicklung der Theorie abstrakter Datentypen war, losgelöst von Implementierungen einer Struktur und den damit verbundenen individuellen Programmiertricks und Hardwarekonstellationen eine Beschreibungsform zu finden, die hauptsächlich Eigenschaften dieser Typen wiederspiegelte, die als implementierungsunabhängig anzusehen sind und den eigentlichen "Datentyp" ausmachen. Darauf aufbauend haben sich weitere Vorteile einer abstrakten Beschreibung von Strukturen ergeben, wie etwa

- die Möglichkeit einer mathematischen Behandlung von Parametrisierungen und Übergabemechanismen,
- systematische Spezifikation von Datenhierarchien bzw. Schnittstellenspezifikation zwischen einzelnen Moduln,
- sowie die im Zusammenhang mit der Verifikation eines Programms sehr wichtige Möglichkeit von Korrektheitsbetrachtungen auf derartigen Spezifikationen.

Für diese Bereiche liegen Lösungsansätze vor, die alle auf abstrakten Beschreibungen nach folgendem Muster basieren: der Datentyp wird definiert durch die auf seinen Objekten ausführbaren Operationen und durch die Beziehungen zwischen diesen Operationen. Dabei wird weder Information über eine mögliche Implementierungsform bereitgestellt noch verwendet, und dem Benutzer einer Inkarnation dieses Datentyps wird es nur erlaubt, Manipulationen an der Inkarnation aussschließlich mit den vereinbarten Operationen durchzuführen.
Die Definition (Spezifikation) eines abstrakten Datentyps (ADT) läßt sich durchführen durch

- ein Netzmodell
- eine operationale Spezifikation
- eine axiomatische Spezifikation

Dabei stellen Netzmodelle (etwa in der Art von Petrinetzen) zum gegenwärtigen Zeitpunkt ausschließlich ein grafisches Hilfsmittel zur Beschreibung der Abhängigkeiten zwischen Operatoren und Trägermengen eines Datentyps dar. Befriedigende Lösungen für die Beziehungen zwischen Spezifikation und Implementierung, Korrektheit einer Implementierung oder Parametrisierungen liegen nicht vor und sind wohl auf Grund prinzipieller Schwierigkeiten bei der Definition geeigneter Netzmorphismen nicht angebbar.
Eine operationale Spezifikation ist eng angelehnt an eine algorithmische Notation und beschreibt die Wirkung der Operatoren mit wohlbekannten Konstrukten von Programmiersprachen. Der offensichtliche Nachteil dieser Methode ist der Zwang zur Über-Spezifikation: man muß implementierungsabhängige Details benutzen, die einerseits "nach außen" unsichtbar sein sollten, zum anderen auch nichts mit den definierenden Eigenschaften

eines Datentyps zu tun haben. Weitere Konsequenzen sind hohe Komplexität, erschwerte Korrektheitsprüfung, und Manifestierung einer bestimmten Implementierung dieses Datentyps - wodurch das Attribut "abstrakt" nicht mehr gerechtfertigt ist. Eine operationale Spezifikation durch ein "Rule-Rewrite-Sytem" vermeidet diese Nachteile ebenso wie eine axiomatische oder algebraische Beschreibung. Da diese die theoretische Grundlage des APE-Systems liefert, werden in dem folgenden Abschnitt die wichtigsten Begriffsbildungen informell eingeführt.

2.1.1. Algebraische Spezifikationen

Mit der Definition einer Signatur wird zunächst die Möglichkeit bereitgestellt, Namen für die Operationen des Datentyps sowie für die Argument- und Wertebereiche dieser Operationen zu vereinbaren.

Def. 1 [Signatur]
Sei S Menge ("der Sorten") und sei Σ eine Familie von Mengen mit
$$\Sigma = \{ \Sigma_{\omega,s} \mid \omega \in S^*, s \in S \}$$
Dann heißt das Tupel (S,Σ) Signatur .
Für ein $f \in \Sigma_{\omega,s}$ heißt (ω,s) Stelligkeit von f sowie ω Rang und s Sorte von f.

Mit der Signatur wird durch Festlegung der Stelligkeiten der Operationsnamen die syntaktische Struktur der aus Operationsnamen bildbaren Terme festgeschrieben.
Über einer Signatur (S,Σ) läßt sich die Menge ihrer Σ-Terme definieren durch:

Def. 2 [Σ-Terme]
1) $\forall s \in S.\ \forall \Sigma_{\varepsilon,s} \in \Sigma.\ \Sigma_{\varepsilon,s} \subseteq (T_\Sigma)_s$
2) $\forall s_1,\ldots,s_n \in S.\ \forall s \in S.$
$\forall f \in \Sigma_{s_1 \ldots s_n,s}.\ \forall t_i \in (T_\Sigma)_{s_i}\ (1 \le i \le n).$

$$f(t_1,\ldots,t_n) \in (T_\Sigma)_s$$

Jede Menge $(T_\Sigma)_s$ enthält also alle nach der Signatur (S,Σ) syntaktisch zulassigen Terme der Sorte s.
Eine heterogene (mehrsortige) Algebra stellt die geeignete mathematische Struktur für einen abstrakten Datentyp dar. Den Zusammenhang zu einer Signatur konstruiert man durch Zuordnung der Sorten- und Operationsnamen der Signatur zu den Trägern (Sorten) und Operationen der Algebra.

Def. 3 [Σ-Algebra]
Sei (S,Σ) eine Signatur.
Dann heißt A Σ-Algebra , wenn gilt:

1) $\forall s \in S.\ \exists A_s$
2) $\forall s \in S.\ \forall f \in \Sigma_{\varepsilon,s}.\ \exists f_A \in A_s$
3) $\forall w \in S^*\ (w = s_1 s_2 \ldots s_n).\ \forall s \in S.\ \forall f \in \Sigma_{\omega,s}.$
$\exists f_A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$

Die Wahl der Mengen A_s und der Operationen f_π der Algebra A wird durch die Signatur nicht vorgeschrieben, lediglich die Existenz .
Wählt man daher für jedes A_s die Menge $(T_\Sigma)_s$ der Σ-Terme der Sorte s und für jedes $f \in \Sigma_{\omega,s}$ die Funktion

$$f_{T_\Sigma}(t_1,\ldots,t_n) := f(t_1,..t_n) \text{ (der Term selbst)},$$

so erhält man die Termalgebra T_Σ.
Diese Σ-Algebra nimmt unter allen Σ-Algebren eine besondere Stellung ein: sie ist initial in der Klasse der Σ-Algebren, d.h. zu jeder anderen Σ-Algebra A gibt es einen Homomorphismus $h: T_\Sigma \to A$. Durch die homomorphe Zuordnung lassen sich so die formalen Operationsaufrufe (in T_Σ) und aktuelle Werte (in einer Algebra/Implementierung A) in Beziehung setzen.
Initialität ist eindeutig bis auf Isomorphie, und die Termalgebra T_Σ als initiale Σ-Algebra eignet sich aus zwei Gründen dazu, als der aus der Signatur (S,Σ) hervorgehende abstrakte Datentyp bezeichnet zu werden:

- sie ist repräsentationsunabhängig in dem Sinne, daß äquivalente (isomorphe) Algebren lediglich durch Umbenennung aus ihr gewonnen werden können,
- sie ist operationserzeugt, d.h. für jeden nach der Signatur syntaktisch korrekten Term gibt es in den Trägern von T_Σ einen Wert, nämlich den Term selbst.

Problematisch wird es allerdings, wenn ein Datentyp um neue Operationen angereichert werden soll, oder von komplexerer Struktur ist, die sich nicht einfach in den Stelligkeiten der Operationen ausdrücken läßt. Im ersten Fall erhält man eine Termalgebra, die nicht mit der intendierten (unangereicherten) identisch ist, z.B. dadurch, daß syntaktisch unterschiedliche Terme mit "eigentlich gleichem" Wert nicht identifiziert werden können. Diese Schwierigkeiten überbrückt man durch Einführung von Σ-Gleichungen über Σ-Termen , in denen neben den Bezeichnern aus der Signatur (S,Σ) auch Variablenbezeichner auftreten dürfen. Eine dem zugrundeliegende Signatur $(S,\Sigma(X))$ entsteht aus (S,Σ) durch

$$\Sigma(X)_{\varepsilon,s} := \Sigma_{\varepsilon,s} \cup X_s$$
$$\Sigma(X)_{\omega,s} := \Sigma_{\omega,s}$$

wobei die X_s (unendliche) Mengen von Variablenbezeichnern mit $X_{s_1} \cap \ldots \cap X_{s_n} = \emptyset$ für $s_1 \neq \ldots \neq s_n$ sind.
Da ein $x \in X_s$ aber "variabel" sein und geeignet "belegt" werden soll, kann es nicht als (Konstanten-)Operationssymbol $x_{T_{\Sigma\omega}}$ in der Termalgebra auftauchen, sondern lediglich als Term im Träger $(T_{\Sigma(\omega)})_s$. Die Elimination all dieser Operationen $x_{T_{\Sigma\omega}}$ aus $T_{\Sigma\omega}$ führt zu der "freien" Termalgebra $T_\Sigma(X)$.
Signaturen werden jetzt um Gleichungen erweitert:

Def. 4 [Spezifikation]
Sei $(S,\Sigma(X))$ Signatur.
Sei $E := \{ E_s \mid s \in S \}$ eine Menge von Mengen E_s von Gleichungen (l,r), wobei l,r $\Sigma(X)$-Terme der Sorte s

sind.
Dann heißt $(S,\Sigma(X),E)$ Spezifikation.

Mit der Einführung der Gleichungen wird zweierlei erreicht:

- Die Wirkung einer neu hinzugefügten Operation auf bereits vorhandene läßt sich beschreiben, und syntaktisch verschiedene Terme lassen sich per Axiom identifizieren.
- Die Menge der Gleichungen induziert eine Kongruenzrelation $\equiv$ auf der Menge der $\Sigma(X)$-Terme , so daß genau die durch Instanzen der Gleichungen identifizierten Terme in einer Kongruenzklasse zusammengefaßt werden.

Insbesondere der zweite Punkt behebt die oben angegebene Schwäche der Termalgebra T_Σ : mit Hilfe der Kongruenzklassen als Träger läßt sich eine "Quotiententermalgebra" $T_{\Sigma/\equiv}$ definieren, die wiederum operationserzeugt und repräsentationsunabhängig (initial) ist, letzteres allerdings in der Klasse der (Σ,E)-Algebren.

Def. 5 [Quotiententermalgebra]
Sei $\equiv$ die durch E auf $T_{\Sigma\omega}$ erzeugte Konguenz.
Dann heißt $T_\Sigma/\equiv$ Quotiententermalgebra mit
1) $\forall s \in S.\ (T_\Sigma/\equiv)_s := \{ [t] \mid t \in T_{\Sigma\omega,s} \}$
2) $\forall w \in S^*\ (w = s_1 s_2 \ldots s_n).\ \forall s \in S.\ \forall f \in \Sigma_{\omega,s}.$
$\forall [t_i] \in (T_\Sigma/\equiv)_{s_i}\ (1 \leq i \leq n).$

$$f_{T_{\Sigma/\equiv}}([t_1],\ldots,[t_n]) := [f(t_1,\ldots,t_n)]$$

Für $E=\emptyset$ stimmen T_Σ und $T_\Sigma/\equiv$ überein.

An dieser Stelle sei noch darauf hingewiesen, daß die so definierte Semantik einer Spezifikation solange ausreichend ist, wie man Datenstrukturen spezifiziert. In dem Augenblick, wo Kontrollstrukturen modelliert werden sollen, müssen zusätzliche Forderungen gestellt werden. Ein Vorschlag in dieser Richtung sind die sogenannten "stetigen Algebren" [ADJ 77], wo Trägermengen als Halbordnungen und Operationen als stetig vorausgesetzt werden. Mit dieser Einschränkung behalten einerseits die meisten Ergebnisse initialer Algebrasemantik in der Sub-Klasse der stetigen Algebren einer Spezifikation ihre Gültigkeit, andererseits kann man zur Erklärung der Bedeutung von Schleifen oder Rekursionen fixpunkttheoretische Sätze heranziehen und damit derartigen Konstrukten innerhalb dieses Ansatzes ihre wohlbekannten Semantiken zuordnen.

Die Entscheidung für die (initiale) Termalgebra als Semantik einer Spezifikation ist nicht zwingend. Insbesondere die Tatsache, daß Terme ausschließlich über die Gleichungen identifiziert werden müssen, führt in manchen Kontexten dazu, daß "eigentlich" gleichwertige Terme unterschieden werden (also in verschiedenen Kongruenzklassen liegen), weil es keine explizite identifizierende Gleichung oder Gleichungsschemata in der Spezifikation gibt. Ein dieses Problem lösender Ansatz ist der der terminalen Algebrasemantik [HR 80]. Mit einer geeigneten

(terminalen) Kongruenzrelation, rekursiv definiert auf den durch eine Spezifikation erzeugten Termen, wird dieser Spezifikation als Semantik die terminale Algebra der entsprechenden Klasse zugeordnet. Terminale Algebrasemantik und die oben skizzierte initiale Algebrasemantik verhalten sich aber in gewissem Sinne "dual", d.h. man kann viele der Begriffsbildungen und Ergebnisse der initialen Algebrasemantik geeignet übersetzen und in "terminalem Kontext" anwenden. Für die vorliegende Arbeit und das APE-System sind jedoch die Konzepte initialer Algebrasemantik grundlegend.

Mit der Quotiententermalgebra als Semantik und der Signatur als Syntax lassen sich abstrakt spezifizierte Datentypen mathematisch beschreiben. Insbesondere erlaubt dieses Konzept Aussagen über die "Korrektheit" einer Spezifikation: nur wenn die Quotiententermalgebra der Spezifikation isomorph zur vorliegenden Struktur ist, hat man das korrekt beschrieben, was man wollte; andernfalls beschreibt die Spezifikation nicht den intendierten Datentyp.

2.1.2. ADT'en im ADTCOMP-System

(vgl. [EKT 79][EKRT 80a][EKRT 80b])
Das dem ADTCOMP-Subsystem des APE zugrundeliegende ADT-Konzept ist im wesentlichen eine Modifikation der algebraischen Spezifikationsmethode, jedoch mit Besonderheiten, die zusammen mit der Entwicklung des Systems aus einer Sammlung von Datentypspezifikationen von Lockemann [LO 77] zu sehen sind. Das Konzept in etwas abgekürzter Form sieht aus wie folgt:

Def. 6 [ALGCOMP ADT-SPEC]
Sei Σ (S,Σ)-Signatur mit $S=\{d,x\}$ (d "Datentypvariable", x "Komponentenvariable").
Sei $A = \{A_s \mid s \in S\}$ mit $A_s = \{\langle l,r\rangle \mid l,r \in (T_\Sigma(x))_s\}$ Axiomenmenge.
Dann heißt das Tripel (S,Σ,A) ADT-Spezifikation , wenn gilt:
1) Es gibt ein ausgezeichnetes Exemplar DO des Datentyps ("leeres Exemplar").
2) Sei $w=s_1 \ldots s_n \in S^*$ mit $s_1 = d$
   $\Rightarrow$ $\Sigma_{\omega,d} \neq \emptyset \wedge \Sigma_{\omega,x} \neq \emptyset$.
3) Sei $f \in \Sigma_{\omega,x}$. Dann
   $\exists\, k \in \Sigma_{\omega,d}$ mit $w=s_1 \ldots s_n \in S^*$. $s_1=d \wedge s_n=x$.
   $\exists\, a \in AA.\ a = \langle f(k(\ldots) , \ldots) , x\rangle$.
   k heißt elementarer Konstruktionsoperator.
   Die Wirkung aller anderen Operationen muß durch Anwendungen von k auf DO beschreibbar sein.
4) Sei $w=s_1 \ldots s_n \in S^*$, $s_i \neq x$, $s \in S$.
   Sei $f \in \Sigma_{\omega,d}$.
   $\Rightarrow$ $\exists\, a \in A_d.\ a = \langle f(DO \ldots) , UNDEF \rangle$.
5) Sei $w=s_1 \ldots s_n \in S^*$ mit $s_1=d$, $s_n=x$.
   Sei $l \in \Sigma_{\omega,d}$, $l \neq k$ [elem. Konstr.] .
   $\Rightarrow$ $\exists\, a \in A_d.\ a = \langle l(DO \ldots) , k(DO \ldots)\rangle$.

Diese Definition einer Spezifikation schränkt die in 2.1.1. gegebene wesentlich ein, und zwar einerseits, was die erlaubten Operatoren angeht (Bed. 2 + 3), sowie andererseits bzgl. der Axiomenmenge (Bed. 4 + 5). Der Grund dafür liegt in dem oben angedeuteten Einfluß der Arbeit Lockemanns, der sich dieses ADT-Konzepts bedient und es das Ziel des ADTCOMP-Entwurfs war, derartige Strukturen zu implementieren.
Da die Klasse der vom System implementierbaren Spezifikationen durch die Bedingungen 1 bis 5 der Def. 6 charakterisiert wird, seien die Unterschiede zum allgemeineren Konzept noch hervorgehoben:

a) Existenz eines primitiven Exemplars.
   Um über die Wirkung eines Funktionsaufrufs TOP(K) auf einen ungefüllten Keller K eine Aussage zu machen, wird die Existenz einer ausgezeichneten Inkarnation DO gefordert, mit deren Hilfe ein entsprechendes Axiom formuliert wird.
   Andere (homogenere) Ansätze sind:

   - in der Signatur die Existenz einer nullstelligen Funktion DO:{} → d zu fordern und die notwendigen Axiome in Termen dieser Funktion zu schreiben,
   - oder die Verwendung sogenannter Fehlersignaturen [ADJ 76] , in denen für die Beschreibung unerlaubter Funktionsanwendungen geeignete Fehlerfunktionen zu der Signatur hinzugefügt werden (siehe dazu auch Abschnitt 5.5.),
   - oder in Spezifikationen partielle Operationen definieren, die auf derartigen Elementen nicht definiert sind. Aus diesem Ansatz folgen allerdings noch weitere Einschränkungen an Operatoren und Sorten (z.B. Berechenbarkeit oder Gödelisierbarkeit, [MAJ 77]).

b) Die Fortsetzbarkeit der Operatoren spielt im theoretischen Konzept keine Rolle, ebenso nicht die Existenz eines "elemtaren Konstruktionsoperators", der alle möglichen Inkarnationen erzeugt.

c) Die Forderung, daß die Wirkung einer Operation ausdrückbar sein muß in Termen des elem. Konstruktionsop. reduziert die Operatorenmenge einer Spezifikation auf einen Operator (nämlich diesen); alle anderen sind "abgeleitete Operatoren" die den spezifizierten Datentyp nicht beeinflussen, da keine neuen Kongruenzklassen in der zugehorigen Quotiententermalgebra erzeugt werden. ([KRE 79],[ADJ 76] und "abgeleitete Operatoren" in 2.3.)

Zusammen mit der mehr impliziten Festlegung der Sortenmenge S auf $S=\{d,x\}$ läßt sich die Klasse der zum gegenwärtigen Zeitpunkt implementierbaren Spezifikationen angeben als Teilmenge der Klasse der (S,Σ,A)-Spezifikationen mit

$$|S| = 2 \text{ und } \forall s \in S.\ \sum_{w \in S^*} |\Sigma_{\omega,s}| \leq 1,$$

wenn abgeleitete Operatoren nicht berücksichtigt werden.

Trotz dieser strengen Restriktionen fallen eine große Zahl von Spezifikationen wohlbekannter Datentypen in diese Klasse (siehe [EKT 79]).
In einem gewissen Sinne erweitert wurde die algebraische Methode durch die Einführung sogenannter Gedächtnisse in ADTCOMP. "Gedächtnisse" sind dabei zusätzliche Zielsorten der Operationen, die (bis jetzt) die Aufgabe von Indexsorten besaßen. Ob jedoch Signaturen mit

$$\Sigma = \{ \Sigma_{u,v} \mid u,v \in S^X \}$$

durch diese Verallgemeinerung (in der das bisherige als Spezialfall enthalten ist) einen leistungsfähigeren Ansatz darstellen, sei dahingestellt, zumal die "stiefmütterliche" Behandlung in der Literatur eher eine andere Deutung zuläßt.

2.2. Parametrisierungen

Die Frage nach dem "Inhalt" eines Kellers oder einer Symboltabelle führt zu parametrisierten Datentypen. Im ADTCOMP-System wird implizit schon mit einer vergleichbaren Struktur gearbeitet: der Komponentenmenge X, die als Standardparameter in allen Spezifikationen und Implementierungen Verwendung findet. Allerdings wird nichts über diese Komponentenmenge ausgesagt noch gefordert. Ein hierarchischer Aufbau von Datenstrukturen oder Moduln legt aber nahe, die "Bausteine" denselben Anforderungen zu unterwerfen wie sie an die Gesamtstruktur gestellt werden, sie also auch als abstrakt spezifizierbare Datenstrukturen anzunehmen. Dieser Sachverhalt liegt einem parametrisierten, abstrakt spezifizierten Datentyp (PADT) zugrunde, für den in ähnlicher Weise wie oben zur Präzisierung einige Begriffsbildungen aufgelistet werden sollen.

2.2.1. Parametrisierte Spezifikation

In Erweiterung des oben angegebenen Konzepts für unparametrisierte Spezifikationen finden jetzt kategorientheoretische Hilfsmittel Anwendung zu Definition eines parametrisierten Typs bzw. einer parametrisierten Spezifikation, zur Formulierung von Korrektheitsaussagen und zur Beschreibung der Parameterübergabe an einen parametrisierten Typ.

Def. 1 [Parametrisierter Datentyp]
Sei (Ω,C) Spezifikation und sei Σ Signatur mit $\Omega \subseteq \Sigma$.
Seien $\mathbf{Alg}_{\Omega,C}$ bzw. $\mathbf{Alg}_{\Sigma}$ die Kategorien der (Ω,C) - Algebren bzw. der Σ - Algebren.
Dann heißt ein persistenter Funktor K ,

$$K : \mathbf{Alg}_{\Omega,C} \rightarrow \mathbf{Alg}_{\Sigma},$$

parametrisierter Datentyp oder Typkonstruktor.
Schreibweise: $[\Omega,C,\Sigma,K]$.

Die Definition als Funktor sichert, daß Homomorphiebeziehungen erhalten bleiben: Sind A und B isomorphe Algebren des Parametertyps (Ω,C), so sind es auch ihre Bilder K(A) und K(B). Gibt es einen Homomorphismus $h: A \rightarrow B$, so existiert auch K(h): $K(A) \rightarrow K(B)$.
Für einen nach Def.1 gegebenen PADT soll jetzt eine Spezifikation entsprechend der Def. 2 aussehen:

Def. 2 [Parametrisierte Spezifikation]
Sei (Σ,A) Parameterspezifikation.
Sei (Σ',A') Ergebnisspezifikation mit $\Sigma \subseteq \Sigma'$.
Dann heißt das Tupel $[(\Sigma,A),(\Sigma',A')]$ parametrisierte Spezifikation , wenn der zugehörige Typkonstruktor der Funktor $F_R: \mathbf{Alg}_{\Sigma,R} \rightarrow \mathbf{Alg}_{\Sigma',R'}$ ist, der jede (Σ,A)-Algebra A auf die durch A frei erzeugte (Σ',A')-Algebra $T_{\Sigma'}(A)$ abbildet.

Die Persistenz $[\forall E \in \mathbf{Alg}_{\Sigma',R'}.\ E \cong U(K(E))$ mit U: $\mathbf{Alg}_{\Sigma',R'} \rightarrow \mathbf{Alg}_{\Sigma,R}$ Vergißfunktor] von K bzw. F_R ist gleichbedeutend mit der Forderung, daß die parametrisierte Spezifikation $[(\Sigma,A,(\Sigma',A')]$ "hinreichend vollständig" und "konsistent" ist. Konsistenz heißt dabei, daß in der durch die Gleichungen in A' erzeugten Kongruenz $\equiv_{R'}$ (Ergebniskongruenz) nicht Terme aus T_{Σ} identifiziert werden, die nach der Parameterkongruenz $\equiv_R$ verschieden sind, und hinreichende Vollständigkeit besagt, daß in den Trägern A_s einer (Σ',A')-Algebra A keine (neuen) Elemente liegen, die nicht $\equiv_R$ - kongruent zu einem Term aus U(A) sind.

Die in der Def. 2 erwähnte "frei erzeugte" Termalgebra $T_{\Sigma}(A)$ über einer (Σ,A)-Algebra A ergibt sich analog der freien Termalgebra $T_{\Sigma}(X)$ über einer Menge X : Zu den Mengen $\Sigma_{\epsilon,s}$ werden die Trägermengen A_s der Algebra A hinzugenommen. Über dieser Signatur $(S,\Sigma(A))$ bildet man wie bekannt die $\Sigma(A)$-Terme. Die Termalgebra $T_{\Sigma}(A)$ entsteht wiederum aus $T_{\Sigma(A)}$ durch "Vergessen" der Trägerelemente von A als nullstellige Operatoren:

$$\forall s \in S.\ \forall w \in S^X.\ \forall f \in \Sigma_{\omega,s}\ (\text{statt } \Sigma(A)_{\omega,s})$$
$$f_{T_{\Sigma}(A)} := f_{T_{\Sigma}(A)}$$

Die freie Termalgebra über A hat für das weitere die gleiche Bedeutung wie im "unparametrisierten" Fall: sie dient zur Bildung einer Quotiententermalgebra, die wiederum die Semantik einer parametrisierten Spezifikation darstellen soll. Unterschied ist, daß die Kongruenz jetzt nicht nur von den Gleichungen aus A' erzeugt wird, sondern es müssen auch die Gleichungen des Parametertyps beachtet werden, so daß diese ebenfalls zur Kongruenzerzeugung herangezogen werden. Als Mengen, auf denen beide Kongruenzen definiert werden, dienen die Träger der freien Termalgebra $T_{\Sigma}(A)$: mit Hilfe einer Evaluierungsfunktion EVAL lassen sich (Parameter)-Terme aus T_{Σ} abbilden in Elemente der (Σ,A)-Algebra A, und diese Elemente sind in den Tragern von $T_{\Sigma'}(A)$ enthalten:

Sei EVAL: $T_\Sigma(A) \rightarrow A$. Dann ist $\equiv_{EVAL}$ die (Parameter)- Kongruenz auf $T_{\Sigma'}(A)$ mit

$$\equiv_{EVAL} = \{\langle t_1, t_2\rangle \mid t_1, t_2 \in T_\Sigma(A) \wedge EVAL(t_1) = EVAL(t_2)\}.$$

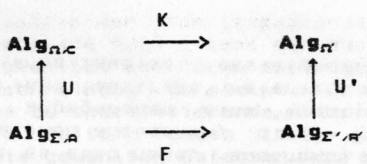
Weiter sind natürlich alle Terme aus (Σ', A') in $T_{\Sigma'}(A)$, so daß tatsächlich die Mengen $(T_{\Sigma'}(A))_s$ zur Bildung der Kongruenz ausreichen.
Mit Hilfsmitteln der Kategorientheorie läßt sich beweisen, daß unter der Annahme einer strukturierten Gleichungsmenge (d.h. die Axiomenmenge besteht entweder aus Gleichungen $E_1 = E_2$ oder aus Ausdrücken der Form $E_1 \wedge E_2 \wedge \ldots \wedge E_n \rightarrow E$, wobei die E_i Gleichungen und E Gleichung oder Ungleichung ist) immer ein Funktor ("freie Konstruktion") existiert, der eine Parameteralgebra auf ihre zugehörige Quotiententermalgbra abbildet [ADJ 79]. Für die hier verfolgten Zwecke genügt es aber, diese Algebra in Form einer Definition einzuführen.

Def. 3 [Quotiententermalgebra f. param.Spec.]
Sei (Σ, A) Parameterspezifikation und sei (Σ', A') Ergebnisspezifikation mit $S \subseteq S'$ und $\Sigma \subseteq \Sigma'$.
Sei $\equiv_{R'}$ die durch die Gleichungen in A' erzeugte Kongruenz und sei $\equiv := \equiv_{R'} \cup \equiv_{EVAL}$ die kleinste Kongruenz, die $\equiv_{R'}$ und $\equiv_{EVAL}$ enthält.
Dann heißt $T_\Sigma(A)/\equiv$ Quotiententermalgebra für $[(\Sigma, A), (\Sigma', A')]$, wenn gilt:
1) $\forall s \in S.\ (T_\Sigma(A))_s := \{[t] \mid t \in (T_\Sigma(A))_s\}$
2) $\forall w = s_1 \ldots s_n \in S^*.\ \forall s \in S.\ \forall f \in \Sigma_{w,s}.\ \forall [t_i] \in (T_\Sigma(A))_{s_i}.$

$$f_{T_\Sigma(A)/\equiv}([t_1], \ldots, [t_n]) := [f_{T_\Sigma(A)}(t_1, \ldots, t_n)]$$

Im unparametrisierten Fall hieß "Korrektheit" einer Spezifikation bzgl. einer gegebenen (heterogenen) Algebra A Isomorphie der Quotiententermalgebra mit A. Jetzt ist eine analoge Definition möglich, jedoch muß der Fall der sogenannten "versteckten" Operationen berücksichtigt werden. Operationen heißen immer dann versteckt, wenn sie zur endlichen Spezifikation eines Datentyps unerläßlich sind und deswegen künstlich zugefügt werden, obwohl sie im Datentyp nicht "gebraucht" werden und dem Benutzer nicht zugänglich sind ("Toystack" [MAJ 77]). Werden versteckte Operationen in der parametrisierten Spezifikation verwendet, so muß sich die oben angesprochene Isomorphiebeziehung auf geeignete Redukte der beteiligten Algebren beziehen.

Def. 4 [Korrektheit]
Sei PDAT = (Ω, C, Ω', K) param. Datentyp und sei SPEC = (Σ, A, Σ', A') param. Spezifikation mit $\Omega \subseteq \Sigma$, $\Omega' \subseteq \Sigma'$, $A \Longrightarrow C$.
SPEC heißt korrekt bzgl. PDAT, wenn das Diagramm



kommutiert bis auf Isomorphie, d.h.

$$\forall A \in Alg_{\Sigma,A}.\ K(U(A)) \cong U'(F(A))$$

wobei U und U' die Vergißfunktoren zwischen den angegebenen Kategorien sind.

Liegen keine versteckten Operationen vor ($\Sigma = \Omega$, $\Omega' \subseteq \Sigma'$), gilt $K(A) \cong F(A)$, d.h. die oben beschriebene (freie) Quotiententermalgebra F(A) ist isomorph zum Ergebnis der Anwendung des Typkonstruktors K auf A.

Eine Möglichkeit zur Beschreibung der Parameterübergabe sei hier nur skizziert [ADJ 80]: Ein Typkonstruktor T ist nach Def.1 nur auf Parametertypen anwendbar, deren Sortennamen und Operationennamen denen der Parametersignatur (Ω, C) literal entsprechen. In Fällen, wo Sorten lediglich umbenannt oder wo neben den Operationen der Signatur Ω noch zusätzliche Operationen in einem aktuellen Parametertyp vorliegen (und das ist die Regel), versagt dieser Beschreibungsmechanismus. Will man ihn dennoch anwenden, muß zunächst folgendes geleistet werden:

1) Umbennung der Träger(=Sorten) von P in geeignete Namen der Parameterspezifikation (Ω, C)
2) Umbennung der Operationen von P in Namen der Parameterspezifikation (Ω, C)
3) "Vergessen" der zusätzlichen Operationen in P.

Die daraus resultierende Algebra muß auf Erfüllung der Parametergleichungen untersucht werden. Sind diese erfüllt, kann man den Typkonstruktur T anwenden und erhält einen Ergebnistyp T(P). Macht man jetzt in T(P) die in 1) - 3) erfolgten Modifikationen rückgängig, ist man fertig: die Parameterübergabe ist beschreibbar durch "geeignete" Anwendung eines Typkonstruktors.
Diese kurze Beschreibung deutet darauf hin, daß Parameterübergabe wesentlich ein Problem der (Um)-Benennung und der Vollständigkeitsprüfung eines Gleichungssystems ist. Besonders der zweite Punkt ist nicht einfach zu bewältigen, insbesondere nicht mit einer einfachen Umbennung der Operatoren in C und einem anschließenden Check, ob P diese Gleichungen erfüllt.
Was man mit derartigen Überprüfungen untersucht, ist hauptsächlich die "Zulässigkeit" einer Parametrisierung. Eine bereits oben in anderem Zusammenhang erwähnte Möglichkeit für diesen Test bilden die Konzepte der Vollständigkeit und Konsistenz: Ist der Ergebnistyp eine Extension des Parametertyps, d.h. ist sein "Parameter-Redukt" isomorph zur initialen Parameteralgebra, so ist die Parametrisierung korrekt, andernfalls liegt nicht mehr der intendierte parametrisierte Typ vor.

2.2.2. Parametrisierungen im ADTCOMP-System

Das realisierte Parametrisierungskonzept erfüllt zwei Forderungen:

- Dem Benutzer soll es möglich sein, eine ADT-Spezifikation explizit als "parametrisiert" einzugeben, und
- die Parameterübergabe an eine Inkarnation eines parametrisierten Typs soll unterstützt werden.

Aus der Tatsache, daß das ADTCOMP-System nicht für parametrisierte Spezifikationen ausgelegt war, andererseits aber ein Redesign vermieden werden sollte, ergab sich die implementierte Form eines Parametrisierungskonzeptes. Auf folgendes ist dabei hinzuweisen:

Da die Spezifikationssprache von ADTCOMP nicht geändert wurde, ist es nicht möglich, Parameter- und Ergebnisspezifikation eines param. Typs getrennt einzugeben (allenfalls implizit in einer Spezifikation). Das System akzeptiert pro Implementierungslauf höchstens eine Spezifikation. Was zur Zeit getan wird, ist ausschließlich eine Qualifizierung der Komponentenmenge X der ADT-Spezifikation, und zwar mit dem vom Benutzer angegebenen Parameter. Bei Inkarnationen dieses Typs müssen dann aktuelle Parameter der zugeordneten Argumentpositionen diese Qualifikation aufweisen, andernfalls wird der Aufruf als nicht zulässig zurückgewiesen [s. auch Fehlerbehandlung in ADTCOMP, S.526].

In eine einmal implementierte formale Struktur [siehe 3.1.,S.38] lassen sich mit der Funktion INSERT beliebige formale Strukturen als Parameter einfügen. Diese Operation, die in etwa das Pendant eines Übergabemechanismus darstellt, wie er im letzten Abschnitt beschrieben wurde, geschieht wiederum auf rein syntaktischer Ebene, d.h. es wird lediglich die in der formalen Struktur festgehaltene Qualifizierung der Komponentenmenge ausgetauscht. Es findet keine Überprüfung der Zulässigkeit einer Parametrisierung statt, weil einerseits der dazu nötige Rückgriff auf eine Parameterspezifikation nicht durchfuhrbar ist, andererseits die Überprüfung selbst, je nach Methode, Dimensionen annehmen kann, die den Rahmen des Systems sprengen wurden. Erwähnt sei z.B. die Vollständigkeitsprüfung des Parametergleichungssystems (Knuth-Bendix-Algorithmus), die allein in der Größenordnung des vorliegenden Systems dimensioniert ist. Die Philosophie des APE-Systems beinhaltet, für Spezialaufgaben Spezialisten ("Experten") bereitzustellen, auf die bei Bedarf zurückgegriffen wird.

Aus diesen Randbedingungen folgt, daß vom gegenwärtigen System Korrektheitbetrachtungen der Implementierung bezüglich der vorliegenden Spezifikation nicht unterstützt werden.
Trotz dieser, in theoretischer Hinsicht gravierender Restriktionen und Vereinfachungen eines Parametrisierungskonzepts ist es doch ausreichend zum Arbeiten mit parametrisierten Inkarnationen in der bisher installierten APE-Umgebung, wie auch in

künstlich erzeugten Anwendungen [siehe PVS-System, S. 271].

2.3. Abstrakte Algorithmen

Für einen Algorithmus gibt es, in Analogie zu Datentypen, zwei Arten der Spezifikation: eine operationale (mit Rückgriff auf implementierungsabhängige Details) und eine abstrakte (nur am Zusammenspiel der Operationen interessierte). Die Nachteile der ersten Methoden sind im Abschnitt 2.1.1. kurz angeführt worden und hier ohne Abstriche gültig. Die ebenfalls angegebene algebraische Spezifikationsmethode führt zunächst zu einer Begriffsverwirrung: spezifiziert man ein Algorithmus in Termen von Gleichungen, so erhält man - nach Def. 5 aus 2.1.1. - einen Datentyp. Tatsächlich ist die Grenze zwischen diesen Begriffen schwer zu ziehen und oft werden sie auch synonym verwendet. Um dennoch zumindest für den Gültigkeitsbereich des APE-Systems eine Unterscheidung der Begriffe zu erlauben, werden als abstrakte Algorithmen von jetzt an Produktionssysteme mit Produktionen der Form

$$\{\ \text{TEST}\ \} \longrightarrow \{\ \text{AKTION}\ \}$$

bezeichnet, wobei TEST und AKTION Ausdrücke symbolisieren, die aus Termen von abstrakt spezifizierten Datentypen aufgebaut sind. Die Idee dahinter ist, die Wirkung eines Algorithmus auf bestimmten Datenstrukturen auszudrücken ausschließlich in Termen dieser Strukturen, so daß jeder der beteiligten Datentypen in gewissem Sinn "unversehrt" bleibt.
Das hinter dieser Definition stehende theoretische Konstrukt ist das eines abgeleiteten Operators:

Def. 1 [Abgeleiteter Operator]
Sei (S,Σ,E) Spezifikation.
Sei X Menge und sei $(T_\Sigma(X))_s$ frei erzeugte Termalgebra über X und $t\in(T_\Sigma(X))_s$.
Sei $var(t) = \{x_1,\ldots,x_n\}$ Menge der Variablen in t mit $sort(x_i) = s_i$.
Sei $w=s_1,\ldots s_n\in S^*$ und $s\in S$.
Dann heißt T abgeleiteter Operator von t, wenn gilt:
1) $\Sigma_{w,s} = \Sigma_{w,s} \cup \{T\}$
2) $E' = E \cup \{<T(x_1,\ldots,x_n)=t>\}$

Die aus der neuen Spezifikation (S,Σ',E') resultierende freie Termalgebra über X, $T_{\Sigma',E'}(X)$, wird durch die Hinzunahme des abgeleiteten Operators nicht verändert bezüglich $T_{\Sigma,E}$: beide sind isomorph und der spezifizierte Datentyp bleibt unversehrt.
Insofern liegt es also nahe, abstrakte Algorithmen als (abgeleitete) Operatoren anzusehen, die im Gegensatz zu den in der Spezifikation angegebenen eine "kompliziertere" Form aufweisen. Das Attribut "abstrakt" referiert auf die Tatsache, daß die Wirkung (Arbeitsweise) des Algorithmus ausschließlich in

Termen von Operationen von ebenfalls abstrakt spezifizierten Datentypen beschrieben wird.
Eine strenge Anwendung der Def.1 erfordert aber, daß zur Spezifikation eines Algorithmus genau ein Term aus $T_\Sigma(X)$ herangezogen wird. Dies ist aber aus zwei Gründen unpraktisch, wenn der Algorithmus als nicht-trivial angesehen wird:

- Man kann durch Verwendung von logischen Operationen (Datentyp BOOL wird dabei vorausgesetzt) die einem Algorithmus innewohnende Struktur nachbilden. Der resultierende Term wird aber unübersichtlich und trübt die Klarheit, die eine abstrakte Beschreibung darstellt.
- Es gibt komplexe Algorithmen, die über einer Datenstruktur nicht in einem Term beschreibbar sind.

Ein weiterer Seiteneffekt tritt auf, wenn ein Algorithmus auf zwei oder mehr Datenstrukturen definiert werden soll. In diesem Fall wird nicht ein abgeleiteter Operator definiert, sondern ein neuer Datentyp durch Kombination von Spezifikationen:

$$S=\{DAT1,\ldots,DATn\}$$
$$\Sigma=\{\Sigma_1 \cup \ldots \cup \Sigma_n \cup \{aOP\}\ \}$$
$$E=\{E_1 \cup \ldots \cup E_n \cup \{<aOP(\ldots)=\ldots>\}\ \}$$

(vgl. auch [EKP 79]).
An dieser Stelle zeigt sich erneut die Schwierigkeit einer klaren Trennung der Begriffe Algorithmus und Datentyp.
Die oben für unsere Zwecke gegebene Definition eines Algorithmus als Produktionssystems (oder Term-Ersetzungssystems) der Form

{Linke Seite} ——> {rechte Seite}

wird jetzt wie folgt realisiert: es wird nicht, wie bei Produktionssystemen oft üblich [DK 77], nach dem syntaktischen Auftreten der "linken Seite" (LS) in einer Datenbasis gesucht, sondern LS wird zu einem Wahrheitswert (T oder F) evaluiert, der bestimmt, ob die zugehörige "rechte Seite" (RS) ausgeführt wird (eval(LS)=T), oder ob in der Produktionenmenge die in der sequentiellen Aufschreibung folgende Produktion aktiviert wird (eval(LS)=F). Jede LS ist demnach als Aussageform anzusehen, deren Argumente Terme von Operationen abstrakter Datentypen oder elementarer Typen wie INTEGER sind. Jede RS dagegen besteht, abhängig von der Struktur des Algorithmus, aus rekursiven Aufrufen dieses Algorithmus (wenn Schleifen o.ä. nachgebildet werden sollen), oder aus Wertberechnungen im terminierenden Fall. Argumente sind beide Male Terme von Operationen abstrakter Datentypen, wobei es keinen Unterschied macht, ob parametrisierte Datentypen in beliebiger Komplexität vorliegen oder nicht.

In dieser Form sind die Algorithmen spezifiziert, die das ALGCOMP-Subsystem des APE's zur Zeit akzeptiert. Zur Spezifikation wird jedoch neben dem eigentlichen Regelsystem noch eine Reihe zusätzlicher Angaben über die auftretenden Datenstrukturen und eventuelle Randbedingungen angefordert. In der

praktischen Arbeit hat sich gezeigt, daß diese Form der Spezifikation durch die Möglichkeit der beliebigen Fallunterscheidung die Komplexität der Terme niedrig hält und dennoch eine kompakte Beschreibung darstellt. Trat dennoch der Fall auf, daß sich eine zu spezifizierende Aktion nicht in einen Term (eine rechte Seite) fassen ließ, lag die Begründung immer darin, daß die gewählten Datenstrukturen, auf denen der Algorithmus arbeiten sollte, nicht der vorliegenden Problemstellung angemessen waren und durch geeignetere ersetzt werden konnten.

2.4. Spezifikationssprachen

Unter Berücksichtigung der in den Abschnitten 2.1.2., 2.2.2. und 2.3. gemachten Einschränkungen werden in diesem Kapitel die Eingabesprachen für das APE-System beschrieben.
Wir unterscheiden dabei die Spezifikationssprache für abstrakte Datentypen, die in 2.4.1. definiert wird, die Spezifikationssprache für abstrakte Algorithmen, beschrieben in 2.4.2., sowie die Eingabesprache für die Regeln von ADTCOMP und ALGCOMP. Zur Definition der Regeleingabesprache verweisen wir auf [EKT 79,pp.28]; die dort angegebene Definition haben wir identisch auch für ALGCOMP-Regeln übernommen. Allerdings kann man das ebenfalls in [EKT 79] beschriebene Regel-Manipulations-Paket nicht für ALGCOMP-Regeln benutzen, da dieses speziell für Regeln zur ADT-Implementierung ausgelegt ist. Eine besondere Spezifikationssprache für parametrisierte Datentypen ist wegen der Bemerkungen in 2.2.2. nicht nötig.

2.4.1. ADT-Eingabe

In diesem Abschnitt wird die Schreibweise und die Bedeutung der ADT-Eingabe und der Regeln erläutert. Der formale Aufbau wird in einer leicht modifizierten Backus-Naur-Form beschrieben.

Anmerkung zur BNF-Schreibweise:
Wir verwenden folgende Zeichen als BNF-Zeichen:

In BNF zu definierende Variablen sind mit Kleinbuchstaben geschrieben und in "..." eingeschlossen.
::= ist das Definitionszeichen.
| ist das ODER-Zeichen.
{ } sind die Alternativen-Klammern.
[ ] sind die Options-Klammern.
.... bedeutet beliebige Wiederholbarkeit des vorherigen Ausdrucks.

Der Benutzer gibt dem ADT-Implementierungs-System eine "adt-

spezifikation" als Eingabe, die folgendermaßen aufgebaut sein muß:

```
'adt-spezifikation"          ::= "datentypangabe" ;
                                 "komponentenmengenangabe" ;
                                 ["mengenangabe" ;]
                                 ["benutzungsangabe" ;]
                                 "funktionalitaetsangabe" ;
                                 "axiomenangabe" ;;

"datentypangabe"             ::= DATENTYP: <name>
                                 {D | S = D {+ "name"} ....}

"komponentenmengenangabe"    ::= KOMPONENTENMENGE: X

"mengenangabe"               ::= [SONSTIGE] MENGEN:
                                 "mengendef" [, "mengendef"]

"mengendef"                  ::= "name" = "mengentyp"

"mengentyp"                  ::= NATZAHL | WORT | BOOLZAHL

"benutzungsangabe"           ::= BENUTZUNG: "benutzungsdef"
                                 [, "benutzungsdef"] ....

"benutzungsdef"              ::= ("name" S ["name"]) =
                                 "benutzungsausdruck"

"benutzungsausdruck"         ::= S | ("name" "benutzungsausdruck"
                                 ["name"] ....)

"funktionalitaetenangabe"    ::= {OPERATOREN | HILFSOPERATOREN |
                                 VORGEGEBEN} "funtionalitaeten

"funktionalitaeten"          ::= "funktionalitaet"
                                 [, "funktionalitaet] ....

"funktionalitaet"            ::= "name": [D [+D]] [+"name"] ....
                                 -> {D | "name"}

"axiomenangabe"              ::= "axiom" [, "axiom"] ....

"axiom"                      ::= "linksausdruck" =
                                 {UNDEF | "rechtsausdruck"}
                                 [FALLS "bedingung"]

"linksausdruck"              ::= ("name" DO [DO] "name" ....) |
                                 ("name" "d1-ausdruck"
                                 ["d1-ausdruck"] ["name"] ....)

"d1-ausdruck"                ::= D | ("name" D2 ["name"] ....) |
                                 ("name" D12 D22 ["name"] ....)

"rechtsausdruck"             ::= DO | D | D2 | X | X2 |
                                 "alphname" | "alphname"2 |
```



```
                                 "vereinbart" |
                                 ("standardop" ["name"] ....)
```

Für die in Axiomen verwendeten Variablen gilt folgende Konvention:
Die Variablen, die auf der linken Seite des Axioms eingeführt werden, werden gemäß ihrer Klammerungstiefe indiziert. Der Index 1 fällt dabei weg (also "D" statt "D1" usw.).

```
"bedingung"                  ::= "beliebiger s-ausdruck" |
                                 (EQ "nichtzahlwert"
                                    "nichtzahlwert") |
                                 ({EQ | LESSP | LESSEQP |
                                 GREATERP | GREATEREQP}
                                 "zahlwert"
                                 {"zahlwert" | "zahl" })
```

"beliebiger s-ausdruck"	kann beliebige Vergleichsfunktionen oder Relationen enthalten.
"nichtzahlwert"	ist ein S-Ausdruck, der sich nicht zu einer Zahl berechnen läßt.
"zahlwert"	ist ein S-Ausdruck, der sich zu einer Zahl berechnen läßt.
"zahl"	ist eine ganze Zahl.

Zur Feststellung von Implikationen zwischen Bedingungen, deren Wahrheitswerte auf der axiomatischen Ebene noch nicht bekannt sein können, werden folgende semantischen Beschränkungen bei Bedingungen festgesetzt:

Variablen innerhalb eines "nichtzahlwert" müssen immer in der gleichen Reihenfolge angegeben werden. (D.h. es ist unzulässig, in verschiedenen Alternativen eines Axioms anzugeben "FALLS (EQ A A2)" und "FALLS (EQ A A1)".) Dies läßt sich vermeiden, indem z.B. grundsätzlich die alphabetische Reihenfolge gewählt wird. Wenn man Zahlenvergleichsbedingungen benutzt, so gibt es zwei Möglichkeiten:

1. Man kann die "zahl" oder den zweiten "zahlwert" in allen Alternativen des Axioms gleich lassen, und die gewünschten Fallunterscheidung durch Benutzung der verschiedenen Vergleichsfunktionen ausdrücken.

2. Man kann die "zahl" ändern, darf dann aber nur die Vergleichsfunktion "EQ" benutzen.

Für die Reihenfolge der Argumente gilt das gleiche wie bei "nichtzahlwert". Es ist also nicht erlaubt, in einem Axiom folgende Fallunterscheidungen zu treffen, z.B.: "FALLS (LESSEQP A 1)", "FALLS (EQ A 2)", "FALLS (GREATEREQP A 3)". Stattdessen muß man scheiben: "FALLS (LESSP A 2)", "FALLS (EQ A 2)", "FALLS (GREATERP A 2)". Allenfalls wäre auch noch erlaubt: "FALLS (EQ A 1)", "FALLS (EQ A 2)" und "FALLS (EQ A

3)".
Diese Beschränkungen erfolgen aus rein pragmatischem Grund. Andere als die erlaubten Fälle sind bislang nie zwingend aufgetreten.

2.4.2. Algorithmen-Eingabe

Für die Schreibweise gelten die in 2.4.1. getroffenen Festlegungen. Da die Produktionsregelmenge einer Algorithmusspezifikation mit (geschweiften) Mengenklammern geklammert wird, werden diese syntaktischen Klammern durch unterstreichen von den Alternativen-Klammern unterschieden:
{ und } sind also die Alternativen-Klammern und { und } sind terminale Zeichen der Algorithmen-Spezifikationssprache.

Eine Algorithmus-Spezifikation, die der Benutzer dem System als Eingabe übergibt, muß folgendermaßen aufgebaut sein:

```
"alg-spezifikation" ::=
                ALGORITHMUS-NAME:   <alg-name> ;
                PARAMETERTYPEN:
                      BENUTZER:     "benutzer-parameter" ;
                       [SYSTEM:     "system-parameter" ;]
                [WERT:              "ausgabe-parameter" ;]
                [RELATION:          "relationssatz" ;]
                [VORAUSSETZUNGEN:   "voraussetzungen" ;]
                [HILFSALGORITHMEN:  "hilfsalgorithmen" ;]
                ALGORITHMUS:        <alg-name>
                                     "param-liste" =
                                    { "regelmenge"
                [KOMMENTAR:         <kommentartext> !] !
```

Eine Algorithmus-Spezifikation besteht also mindestens aus Angaben unter den Schlüsselworten ALGORITHMUS-NAME:, PARAMETERTYPEN: BENUTZER: und ALGORITHMUS: . Alles, was auf ein Schlüsselwort folgt, wird diesem zugeordnet, bis ein ";" auftritt. Dies beendet das Einlesen und bewirkt den Übergang zum nächsten Schlüsselwort.
Ausnahmen: Der Schlüssel ALGORITHMUS wird innerhalb von "regelmenge" mit "}" abgeschlossen, der KOMMENTAR wird mit "!" abgeschlossen (deshalb sollte innerhalb des Kommentartextes kein "!" auftreten). Die ganze Spezifikation wird ebenfalls mit einem "!" beendet.

```
"benutzer-parameter" ::= <datentyp> <parameter-name>
```

<datentyp> darf entweder ein von ADTCOMP implementierter Datentyp oder ein vom INTERLISP-System zur Verfügung gestellter sein. Dabei sind z.Z. folgende abstrakte Datentypen implementiert:

BINAERERBAUM,
EINSEITIGELISTE,
FLEXREIHE,

FLEXVERBUND,
GRENZKELLER,
KELLER,
MENGE,
OFFENERKELLER,
OFFENELISTE,
REIHE,
SCHLANGE,
SCHLUESSELLISTE,
VERBUND,
ZWEISEITIGELISTE.

(dazu siehe auch [EKT 79])
Zulässig sind auch beliebige Parametrisierungen obiger Typen.
INTERLISP stellt folgende Datentypen zur Verfügung:

ARRAY,
BOOL,
INTEGER (einfach und doppelgenau),
LISPZELLE,
LITATOM,
REAL (einfach und doppelgenau),
STRING.

Außerdem gibt es noch den Typ LISPATOM; ein LISPATOM ist entweder ein LITATOM, also ein literales Atom, oder ein INTEGER.

Werden neue Datentypen implementiert oder wird mit anderen LISP-Systemen gearbeitet, so müssen die entsprechenden Listen mit den Namen der Datentypen in den Funktionen PARAMETER-UEBERPRUEFUNG (s.a. 5.2.1.2.2., S.407), DATENTYPP (s.a. 5.2.1.2.2., S.411) und LISP-PHASE (s.a. 5.2.1.4., S.464) modifiziert werden.

```
"system-parameter" ::=  <datentyp> <parameter-name> :=
                                {<wert> | <LISP-Ausdruck>}
```

Während die Benutzer-Parameter als Argumente des implementierten Algorithmus durch Wertübergabe zum Aufrufzeitpunkt definiert werden, wird den System-Parametern vorher der spezifizierte Wert bzw. der Wert des spezifizierten LISP-Ausdrucks zugewiesen. In beiden Fällen muß natürlich der Wert vom Typ <datentyp> sein, was allerdings nur für <wert> überprüft werden kann (Funktion PARAMETER-UEBERPRUEFUNG).
<LISP-Ausdruck> muß ein geklammerter Ausdruck sein, der direkt ausführbar ist, also insbesondere nur Aufrufe von INTERLISP-Funktionen bzw. definierter Hilfsalgorithmen und Hilfsfunktionen oder Initialisierungen von Datentypen (Funktion INIT) enthält. Dies wird innerhalb von RALGSPEC durch die Funktion FKT-SYMBOL-IN (s.a. 5.2.1.2.2., S.413) überprüft.

```
"ausgabe-parameter" ::= <parameter-name>
```

Treten bei parallel-rekursiven Algorithmen auf mehr als einer rechten Seite der Produktionsregeln nur Werte (UNDEF, TRUE, FALSE, T, NIL) oder Parameter des Algorithmus auf, so muß un-

ter dem Schlüssel WERT: diejenige rechte Seite spezifiziert werden (Parameter-Name oder Wert), die bei Ausführung den Wert des Algorithmus liefern soll. Eine entsprechende Überprüfung findet in RALGSPEC statt (s.a. 5.2.1.2.1., S.397).

```
"relationssatz" ::=     <relationssatz-name>
```

Der hier spezifizierte Name eines Relationssatzes dient in der Algorithmen-Konstruktions-Phase dazu, die Relationszeichen in der Spezifikation durch Funktionsnamen zu ersetzen, die diese Relation für den entsprechenden Datentyp realisiert.

```
"voraussetzungen" ::=   "vergleich" [, "vergleich"] ....

"vergleich"             wird weiter unten innerhalb von "re-
                        gelmenge" definiert; hier sei nur er-
                        wähnt, das in RALGSPEC mittel der
                        Funktion FKT-SYMBOL-IN für Vorausset-
                        zungen eine ähnliche Überprüfung wie
                        für System-Parameter vorgenommen wird.

"hilfsalgorithmen" ::=  <hilfsalgorithmus-name>
                        [, <hilfsalgorithmus-name>] ....
```

Hier müssen alle in der Spezifikation auftretenden Hilfsalgorithmen genannt werden, insbesondere auch die, die unter den Schlüsselworten PARAMETERTYPEN: SYSTEM: und VORAUSSETZUNGEN: vorkommen.

```
"param-liste" ::=       ( <parameter-name>
                        [, <parameter-name>] ....)
```

Jeder <parameter-name> der an dieser Stelle vorkommt, muß vorher unter dem Schlüssel PARAMETERTYPEN: mit Typ (BENUTZER:) bzw. Typ und Wert (SYSTEM:) spezifiziert worden sein. Dies wird in RALGSPEC durch die Funktion PARAMETER-UEBERPRUEFUNG mit ihren Unterfunktionen DATENTYPP und BOOL-INTERN überprüft.

Syntaktisch muß die Menge der Produktionsregeln folgender Grammatik genügen:

```
"regelmenge"          ::= "regel" {; "regelmenge" | ⟩ }
"regel"               ::= "linkeseite" "rechteseite"
"linkeseite"          ::= "vergleich" {{UND | ODER}
                                       "linkeseite" | ——>}
"rechteseite"         ::= "funktions-aufruf" | "dt-lisp-op" |
                          "par-rek-aufruf" | "wert"
"par-rek-aufruf"      ::= PRA ( "parameterliste"
"wert"                ::= "algparam" | "relwert"
"vergleich"           ::= "vglseite" "rel" "vglseite"
"vglseite"                "funktions-aufruf-nr" |
                          "dt-lisp-op" |
                          "relwert" | <algparam> |
                          "kwote-atom" | <integer>
"funktions-aufruf"    ::= {<algname> | <hilfsalgorithmus>}
                          ( "parameterliste"
```



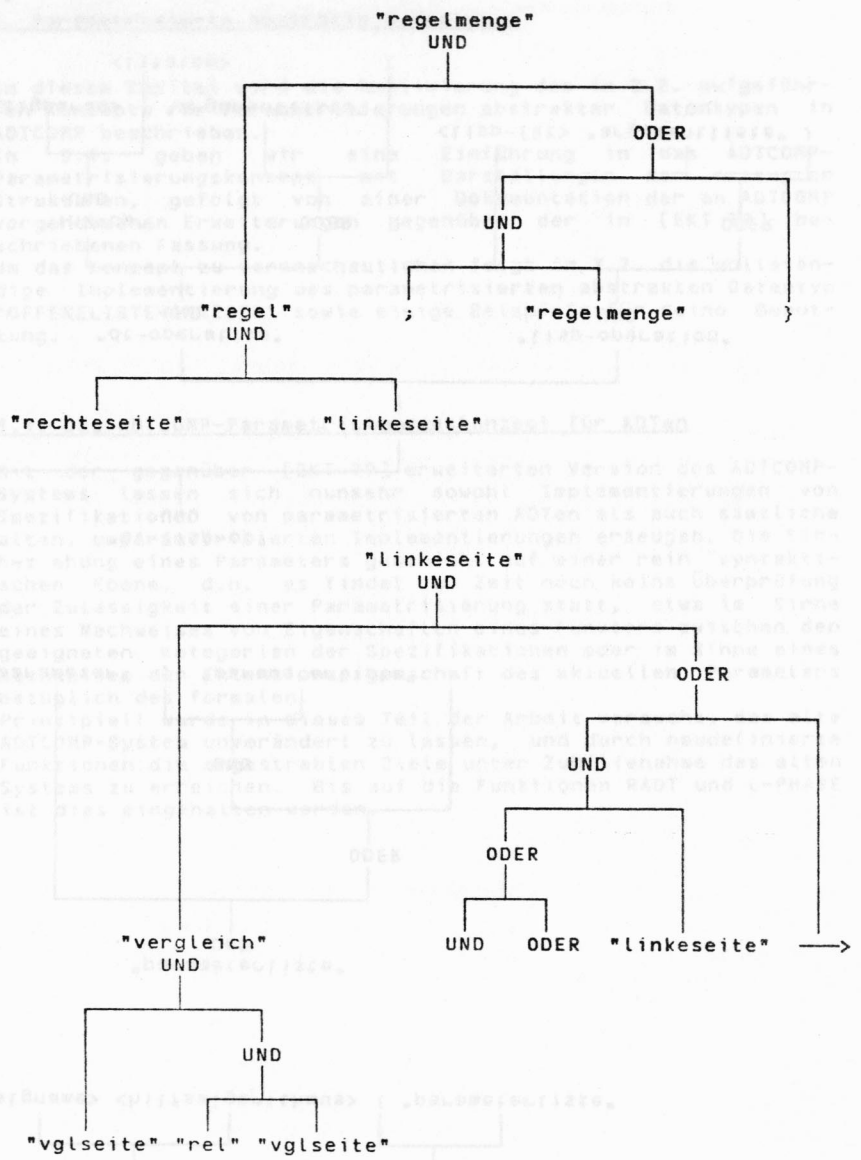
```
"funktions-aufruf-nr" ::= <hilfsalgorithmus>
                          ( "parameterliste"
"parameterliste"      ::= "parameter" {, "parameterliste" | )}
"parameter"           ::= "dt-lisp-op" | "funktions-aufruf" |
                          <atom>
"dt-lisp-op"          ::= ( {"dt-operation" |
                          "lisp-operation"}
"dt-operation"        ::= "ink+op" {"argumentliste" | )}
"lisp-operation"      ::= <lisp-fkt> {"argumentliste" | )}
"ink+op"              ::= <algparam> <litatom>
"argumentliste"       ::= "parameter" {"argumentliste" | )}
"rel"                 ::= = | /= | < | > | <= | >=
"relwert"             ::= UNDEF | TRUE | FALSE
"kwote-atom"          ::= '<litatom> (ein Atom!)
```

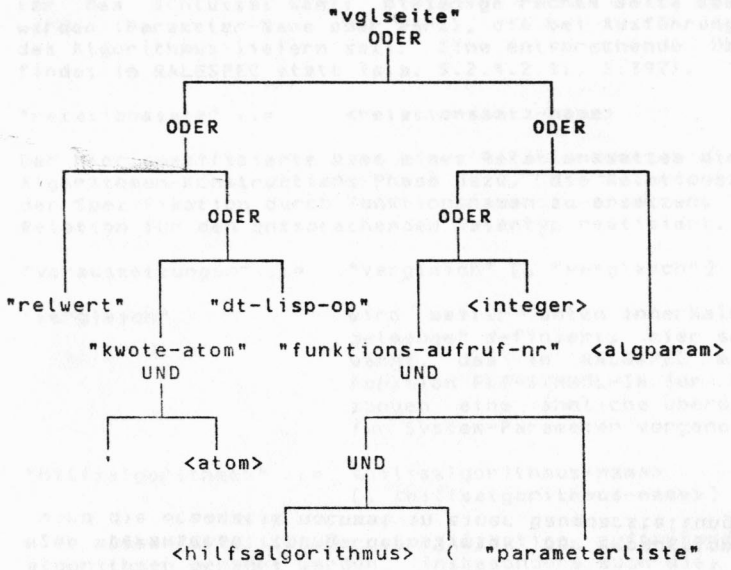
Die Grammatik wird durch ein erkennendes System, den Regelparser, realisiert, der aus der Funktion REGELPARSER und ihren Unterfunktionen besteht. Dabei entspricht jedem nichtterminalem Symbol der Grammatik genau eine LISP-Funktion gleichen Namens (mit angehängtem "p" für Prädikat). Näheres in Abschnitt 5.2.1.2.2., Seite 406.
Die Regelmenge wird mit "⟩" (in Spezifikationen nicht unterstrichen) abgeschlossen, jede Regel innerhalb der Regelmenge mit ";". Linke und rechte Seiten einer Regel werden durch "——>" getrennt.
Linke Seiten bestehen aus einem Vergleich oder mehreren Vergleichen, die dann entweder konjunktiv (UND) oder disjunktiv (ODER) verknüpft sind.
Ein Vergleich setzt sich aus zwei Vergleichsseiten (vglseite) und einem der sechs zulässigen Relationszeichen =,/=,<,>,<=,>= zusammen (s.a. 5.3.2. Relationssätze, Seite 509).
Eine Vergleichsseite ist entweder eine Datentyp- oder LISP-Operation (dt-lisp-op), ein nicht-rekursiver Funktionsaufruf (funktions-aufruf-nr), d.h. ein Hilfsalgorithmusaufruf und kein Aufruf des gerade spezifizierten Algorithmus, einer der Werte UNDEF, TRUE oder FALSE (relwert), einer der Parameter des spezifizierten Algorithmus (algparam), eine ganze Zahl (integer) oder ein gekwotetes Atom (kwote-atom). Letztere sind notig, da alle Algorithmen von ALGCOMP als LAMBDA-Formen implementiert werden, d.h. die Argumente werden beim Aufruf des Algorithmus evaluiert. Will man die Evaluierung unterdrücken, mussen die entsprechenden Argumente gekwotet werden. Dies muß, z.B. bei Aufrufen von Hilfsalgorithmen, schon in der Spezifikation mittels gekwoteter Atome (Schreibweise: 'atom) festgelegt werden.

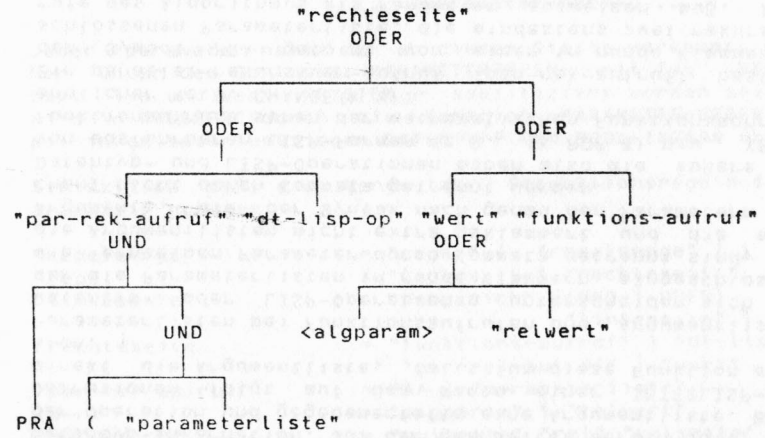
Eine rechte Seite ist entweder ein Funktions-Aufruf, eine Datentyp- oder LISP-Operation, ein parallel-rekursiver Aufruf des Algorithmus (par-rek-aufruf) oder ein Wert.
Ein Funktions-Aufruf kann ein rekursiver Aufruf des Algorithmus sein, dann beginnt der Aufruf mit dem Namen des Algorithmus, oder ein Hilfsalgorithmusaufruf, dann steht der Name des Hilfsalgorithmus an erster Stelle. Auf den Namen folgt eine in runde Klammern eingeschlossene Parameterliste, die aus einem oder mehreren Parametern besteht. Falls mehrere Parameter auftreten, werden diese durch Kommata getrennnt.

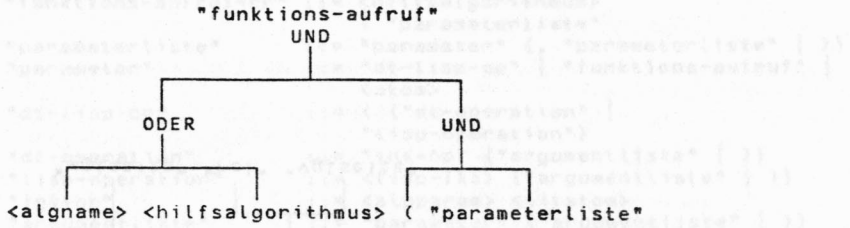
Natürlich muß die Anzahl der Parameter der Parameterliste des jeweils aufgerufenen Algorithmus oder Hilfsalgorithmus entsprechen. Jeder Parameter kann seinerseits wieder eine Datentyp- oder LISP-Operation, ein Funktions-Aufruf oder ein Atom sein und muß, bis auf den letzten Fall, entsprechend geklammert sein.
Datentyp- bzw. LISP-Operationen sind ganz in runde Klammern eingeschlossen. Es folgt bei Datentyp-Operationen der Name der Datentyp-Inkarnation, auf der gearbeitet werden soll, der Name der Operation und gegebenenfalls eine Argumentliste. Bei LISP-Operationen folgt auf den Namen einer INTERLISP-Funktion direkt die Argumentliste, falls für diese Funktion erforderlich.
Parameterlisten bei Funktionsaufrufen und Argumentlisten bei Datentyp- oder LISP-Operationen unterscheiden sich dadurch, daß die Parameterlisten in runde Klammern eingeschlossen und die einzelnen Parameter durch Kommata getrennt sind, während die Argumentlisten nicht extra geklammert und die einzelnen Argumente, die der Syntax nach genau den Parametern entsprechen, nicht durch Kommata getrennt werden.
Datentyp- und LISP-Operationen haben also die äußere Gestalt von ausführbaren LISP-Formen (z.B. (K POP 3) bzw. (EQ A B)), Funktionaufrufe sehen der mathematischen Funktionsschreibweise ähnlicher (z.B. CHANGE(X Y)).
Ein parallel-rekursiver-Aufruf (par-rek-aufruf) besteht aus dem Symbol PRA gefolgt von einer in runde Klammern eingeschlossenen Parameterliste, die mindestens zwei rekursive Aufrufe des Algorithmus als Parameter ausweisen muß. Ansonsten gilt für die Parameter das oben gesagte.
Außerdem kann die rechte Seite einer Regel auch nur aus einem Parameter des Algorithmus oder einem der Werte UNDEF, TRUE oder FALSE bestehen.

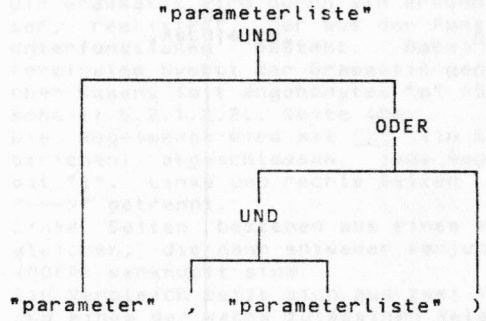
Zur Veranschaulichung der Syntax der Produktionsregelmenge wird die Grammatik nochmal in einer Baumdarstellung angegeben:

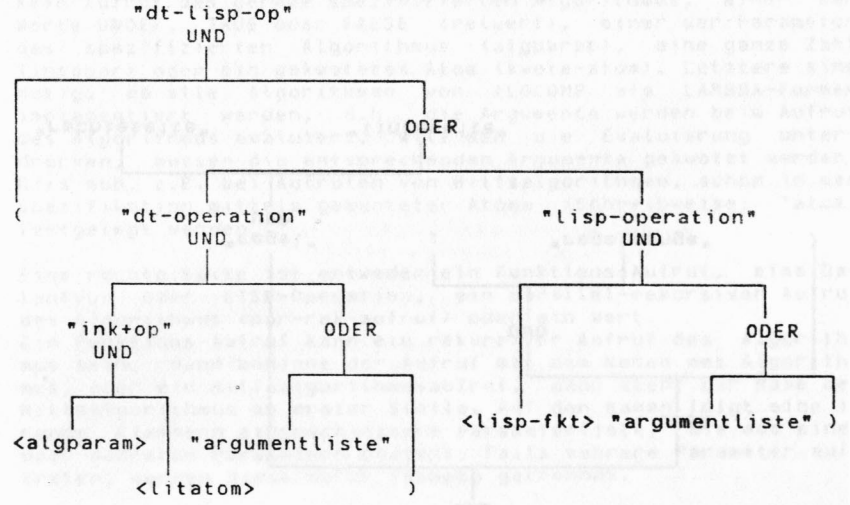


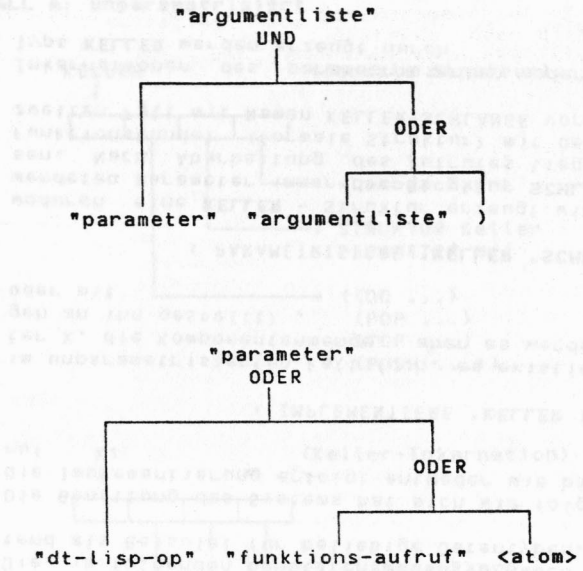

"vglseite"
ODER
ODER
ODER
ODER
ODER
"relwert"
"dt-lisp-op"
<integer>
"kwote-atom"
"funktions-aufruf-nr"
<algparam>
UND
UND
'
<atom>
UND
<hilfsalgorithmus>
(
"parameterliste"


"rechteseite"
ODER
ODER
ODER
"par-rek-aufruf"
"dt-lisp-op"
"wert"
"funktions-aufruf"
UND
ODER
UND
<algparam>
"relwert"
PRA
(
"parameterliste"


"funktions-aufruf"
UND
ODER
UND
<algname>
<hilfsalgorithmus>
(
"parameterliste"


"parameterliste"
UND
ODER
UND
"parameter"
,
"parameterliste"
)


"dt-lisp-op"
UND
(
ODER
"dt-operation"
UND
"lisp-operation"
UND
"ink+op"
UND
ODER
ODER
<algparam>
"argumentliste"
<lisp-fkt>
"argumentliste"
)
<litatom>
)

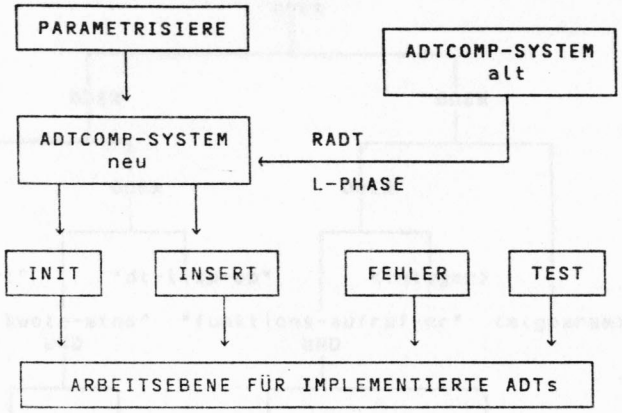


3. Parametrisierte Abstrakte Datentypen

In diesem Kapitel wird die Realisierung des in 2.2. aufgeführten Konzepts für Parametrisierungen abstrakter Datentypen in ADTCOMP beschrieben.
In 3.1. geben wir eine Einführung in das ADTCOMP-Parametrisierungskonzept mit Darstellungen der erzeugten Strukturen, gefolgt von einer Dokumentation der an ADTCOMP vorgenommenen Erweiterungen gegenüber der in [EKT 79] beschriebenen Fassung.
Um das Konzept zu veranschaulichen folgt in 3.2. die vollständige Implementierung des parametrisierten abstrakten Datentyp "OFFENELISTE-INTEGER" sowie einige Beispiele für seine Benutzung.

3.1. Das ADTCOMP-Parametrisierungskonzept für ADTen

Mit der gegenüber [EKT 79] erweiterten Version des ADTCOMP-Systems lassen sich nunmehr sowohl Implementierungen von Spezifikationen von parametrisierten ADTen als auch sämtliche alten, unparametrisierten Implementierungen erzeugen. Die Einbeziehung eines Parameters geschieht auf einer rein syntaktischen Ebene, d.h. es findet zur Zeit noch keine Überprüfung der Zulässigkeit einer Parametrisierung statt, etwa im Sinne eines Nachweises von Eigenschaften eines Funktors zwischen den geeigneten Kategorien der Spezifikationen oder im Sinne eines Nachweises der Extensionseigenschaft des aktuellen Parameters bezüglich des formalen.
Prinzipiell wurde in diesem Teil der Arbeit versucht, das alte ADTCOMP-System unverändert zu lassen, und durch neudefinierte Funktionen die angestrebten Ziele unter Zuhilfenahme des alten Systems zu erreichen. Bis auf die Funktionen RADT und L-PHASE ist dies eingehalten worden.



Die Veränderung des ADTCOMP-Systems durch Modifikation der Funktionen RADT und L-PHASE war notwendig, um auch bei unparametrisierten Datentypen eine Benutzungskontrolle zu gewährleisten: War es bisher möglich, auf Inkarnationen I1 eines Datentyps D1 Operationen eines Datentyps D2 anzuwenden, so wird dies jetzt verhindert; nur datentypspezifische Operationen sind sowohl im parametrisierten wie im unparametrisierten Fall anwendbar.

Die im folgenden benutzten Datentyp-Namen stehen stellvertretend als Beispiel für beliebige Datentypen.

Die Benutzung des Systems hat sich wie folgt geändert:
Die Implementierung erfolgt entweder wie bisher durch den Aufruf

(IMPLEMENTIERE '**KELLER**)

im unparametrisierten Fall (d.h. es existiert zwar ein Parameter X, die Komponentenmenge, aber es werden keine Anforderungen an ihn gestellt) ,
oder mit

(PARAMETRISIERE '**KELLER** '**SCHLANGE**)

wodurch eine **KELLER** - Struktur erzeugt wird, jedoch die verwendeten Parameter immer die Struktur **SCHLANGE** aufweisen müssen. Nach Abarbeitung des Aufrufes liegt im ersten Fall ein Funktionsbundel (formale Struktur) mit dem Namen KELLER , im zweiten Fall mit Namen KELLER-SCHLANGE vor.

Inkarnationen des parametrisierten oder unparametrisierten Typs **KELLER** werden erzeugt durch

(INIT K **KELLER**)

bzw.

(INIT KS **KELLER-SCHLANGE**) ,

wobei K und KS die Bezeichner der Inkarnationen sind.

Soll anstelle einer **KELLER-SCHLANGE** - Inkarnation eine mit einem anderen Parameter erzeugt werden, so wird zunächst in die formale parametrisierte Struktur "KELLER-SCHLANGE" der neue Parameter, etwa MENGE , eingefügt:

(INSERT **MENGE** IN **KELLER-SCHLANGE**) .

Das Ergebnis des Aufrufs ist eine formale parametrisierte Struktur "KELLER-MENGE", die jetzt zum Initiieren von **KELLER-MENGEN** - Inkarnationen herangezogen wird :

(INIT KM **KELLER-MENGE**) .

Das Arbeiten mit diesen Inkarnationen geschieht in etwas anderer Syntax als bisher. Anstelle von

(<DT-Name> <OP-Name> <INK-Name> <ARG-Liste>)

wird jetzt

(<INK-Name> <OP-Name> <ARG-Liste>)

aufgerufen; beispielsweise geht

(KELLER PUSH K 1)

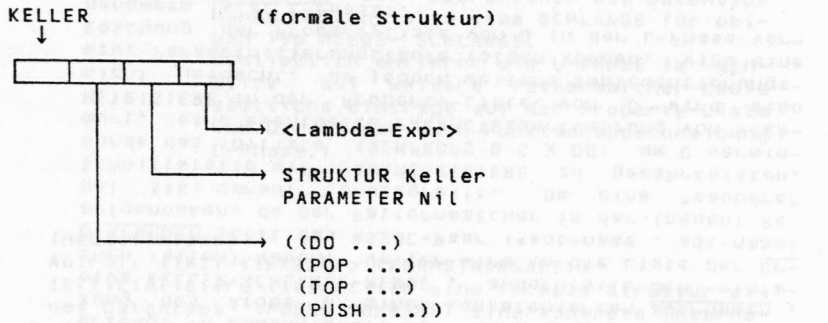
über in

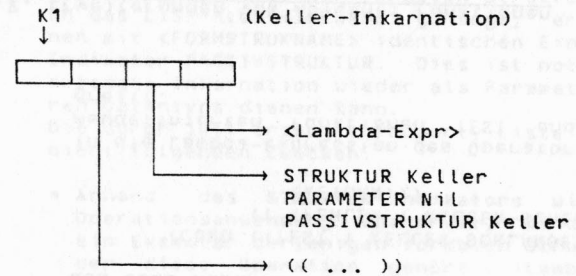
(K PUSH 1)

d.h. (nach INTERLISP-Syntax): der Inkarnationsname wird als Funktionaufruf aufgefaßt, innerhalb dessen allerdings wieder der ursprüngliche Aufruf erzeugt wird (Konservierung des alten ADTCOMP-Systems). Zusätzlich wird jedoch durch diese Neudefinition der Aufrufsyntax die Möglichkeit zu bestimmten Anwendungs- und Strukturüberprüfungen bereitgestellt und genutzt (Funktionen INIT, TEST).

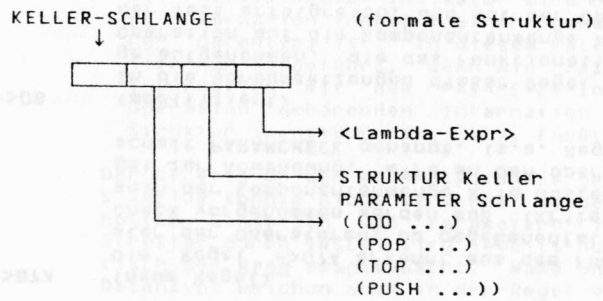
Nach geeigneten Implementierungs- und Initialisierungsläufen sind an die aufgeführten LISP-Atome folgende Daten gebunden:
Wir betrachten als Beispiel den Datentyp Keller:

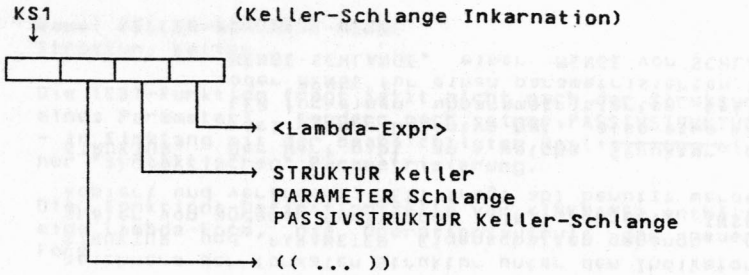
FALL A: unparametrisiert



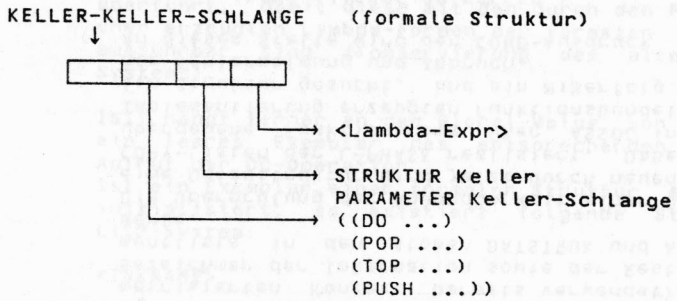


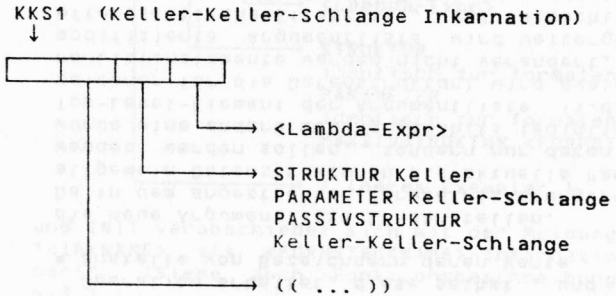
FALL B: einfach parametrisiert





FALL C: zweifach parametrisiert





usw.

3.1.1. Erweiterungen von ADTCOMP: Regeln

->07A (neue Regel)
Die Regel ->07A erkennt aus dem Funktionalitätenmuster der Operatoren, ob gegebenenfalls ein Parametercheck vorgenommen werden muß (Kriterium: Vorhandensein der Komponentenmenge X im Muster). Kommt die Regel zur Anwendung, wird an den Operator die Eigenschaft PARAMCHECK gehängt. (s.a. Regel ->08)

->08 (modifiziert)
In die Voraussetzungen dieser Regel wurde eine Abfrage aufgenommen, die das Funktionalitätenmuster einer Operation auf die Komponentenmenge X matcht. Verläuft der Test erfolgreich, und hat der Operator die Eigenschaft PARAMCHECK, so wird in der Folgerung der "I-atomare-Schlüssel" CHECK gesetzt. (s.a. I38)

I38 (neue Regel)
Erkennt den Schlüssel CHECK, der darauf hinweist, daß bei mindestens einem der Parameter der Operation eine Strukturüberprüfung stattfinden muß, d.h. ob bei Anwendung der Operation ein aktueller Parameter tatsächlich die vereinbarte Struktur hat.
Die Regel realisiert diese Überprüfung durch Einbau des COND-Ausdrucks

```
(COND ((TEST 1 KELLER-SCHLANGE))
      (T (FEHLER 1 KELLER-SCHLANGE)
         (RETURN)))
```

in die Lambda-Expression des Operators, der auf die neudefinierten Funktionen TEST und FEHLER zurückgreift.

3.1.2. Erweiterungen VON ADTCOMP: Funktionen

RADT (modifiziert)
erzeugt in ~~Abhängigkeit~~ von einer PARAMETERwertsetzung des Atoms "D" einen konkatenierten adt-namen, etwa KELLER-SCHLANGE-MENGE, andernfalls den einfachen (alten) Namen. Weiter wird in die Liste der Urbindungen jetzt das ASSOC-Paar (*adt-name . adt-name) aufgenommen, da der Patternmatcher in der (neuen) Regel I38 darauf zurückgreift. Um eine "saubere" Schnittstelle mit PARAMETRISIERE zu gewährleisten, wurde das initiale (REMPROPS D S X DO) um D vermindert; eine eventuelle PARAMETERwertsetzung von PARAMETRISIERE in der Property-Liste von D wird also nicht gelöscht. Da jedoch weitere Implementierungs- bzw. Parametrisierungsläufe folgen können, wird eine Loschung der Propertyliste von D in der L-Phase vorgenommen (s.a. L-PHASE).

L-PHASE (modifiziert)
Die Funktion L-PHASE kopiert eine Lambda-Expression, die an den Namen der erzeugten formalen (parametrisierten) Struktur als deren Aufruffunktion gebunden wird. Aus dieser Lambda-Expression wurde die Funktion EVLIS entfernt: EVLIS evaluiert die Argumentliste L eines Operationsaufrufes derart, daß

- anstelle des Namens der Datenstruktur, auf der die Operation arbeitet, diese selbst , und
- anstelle von Bezeichnern deren Werte

die neue Argumentliste L' darstellen.
Da in dem angestrebten Parametrisierungskonzept nicht allgemein Datenstrukturen als aktuelle Parameter verwendet werden sollen, sondern nur deren Bezeichner, wurde eine andere Lösung gewählt: lediglich das erste Top-Level-Element der Argumentliste (i.d.R. der Bezeichner für die Datenstruktur) wird evaluiert, andere Listenelemente werden nicht verändert, und die so modifizierte Argumentliste wird weitergegeben. Der offensichtliche Effekt ist, daß tatsächlich Bezeichner in einen **KELLER** K gePUSHt oder aus einer **MENGE** M ENTFERNt werden. Aus Gründen der einfachen Referenz bei fehlerhaften Funktionsaufrufen (Parameter hat fehlerhafte Struktur bzw. ist in einem anderen parametrisierten Kontext bereits verwendet) werden der Bezeichner der Inkarnation sowie der Rest der Argumentliste in den Atomen DATSTRUK und ARGUMENTE gemerkt.
Die Überprüfung der Anwendbarkeit einer Operation auf eine Datenstruktur wird jetzt durch neuen Code in den COPY-Listen der L-PHASE realisiert. Dabei wird der übergebene Funktionsname über ASSOC in dem bei der Implementierung erzeugten Funktionsbündel der formalen Struktur gesucht, und ein Mißerfolg führt zu einer Fehlermeldung und Abbruch.
An dieser Stelle wird der COND-Ausdruck

```
(COND ((ASSOC OP KELLER )
       NIL)
      (T (PRINT "Fehlermeldung")
         (RETURN)))
```

eingebaut.
Weiterhin werden jetzt an die Property-Liste des Bezeichners der formalen Struktur unter den Indikatoren STRUKTUR und PARAMETER Eigenschaften gehängt, die später von anderen Funktionen (INIT kopiert, INSERT kopiert und verändert, TEST fragt ab) benutzt werden:

STRUKTUR: Der Wert gibt an, welche Struktur der vorliegende Datentyp hat, also etwa KELLER für einen unparametrisierten Keller oder MENGE für einen parametrisierten Typ MENGE-SCHLANGE, einer MENGE von SCHLAN-

GEN.

PARAMETER: Der Wert gibt an, welche Struktur die Argumente der Operationen des Datentyps haben müssen, also etwa SCHLANGE für obigen Typ MENGE- SCHLANGE.
Schließlich entfernt die L-PHASE im Hinblick auf weitere Pattermatcher-Läufe sämtliche Einträge auf der Property-Liste von D. (Siehe auch Funktionsbeschreibung L-Phase.)

INIT (neudefiniert)
Aufruf: (INIT <INKNAME> <FORMSTRUKNAME>)
INITitialisiere liefert für eine formale Struktur eines Datentyps (FORMSTRUKNAME) eine konkrete Inkarnation mit dem Namen <INKNAME> gemäß den oben beschriebenen Bindungen.
Die Property-Liste der formalen Struktur wird kopiert an das LISP-Atom <INKNAME> gehängt, erweitert um einen mit <FORMSTRUKNAME> identischen Eintrag unter dem Indikator PASSIVSTRUKTUR. Dies ist notwendig, da die erzeugte Inkarnation wieder als Parameter eines anderen Datentyps dienen kann.
Die durch INIT erzeugte Property-Liste von <INKNAME> dient folgenden Zwecken:

- Anhand des STRUKTUR-Indikators wird bei einer Operationsanwendung überprüft, ob die Inkarnation ein Exemplar derjenigen formalen Struktur ist, zu der diese Operation gehört (Lambda-Expr. von <FORMSTRUKNAME>).
- Anhand des PARAMETER-Indikators wird überprüft, ob die Argumente einer auf eine Inkarnation angewandten Operation mit dem für diesen Datentyp vereinbarten Parameter übereinstimmen. (s.a. Funktion TEST)
  Bei unparametrisierten Datentypen wird der PARAMETER-Indikator auf NIL gesetzt. Eine Überprüfung dieser Eigenschaft in Regel ->08 führt dann in diesem Fall zu einem negativen Resultat.
- Der PASSIVSTRUKTUR-Indikator wird benutzt, wenn eine Inkarnation Argument einer Operation eines anderen Datentyps ist. In diesem Fall wird die Operation nur ausgeführt, wenn die PASSIVSTRUKTUR der Inkarnation mit dem PARAMETER-Indikator der zur Operation gehörenden Inkarnation bzw formalen Struktur identisch ist. (s.a. Funktion TEST)

Der bereits an den <FORMSTRUKNAME> gehängte Indikator STRUKTUR konnte nicht verwendet werden. Beim wiederholten Einsetzen von Parametern in eine formale Struktur durch INSERT wird jeweils nur rein syntaktisch der Name ausgetauscht. Name und Struktur eines Datentyps weichen aber in der Regel voneinander ab:

Name: KELLER-SCHLANGE-MENGE
Struktur: Keller

Die TEST-Funktion fragt jetzt nicht nach der Struktur eines Parameters, sondern nach seiner PASSIVSTRUKTUR - in Einklang mit der beabsichtigten Realisierung einer "syntaktischen" Parametrisierung.

Die Funktions-Definitionszelle von <INKNAME> enthält eine Lambda-Form, die Operationsaufrufe der neuen Form

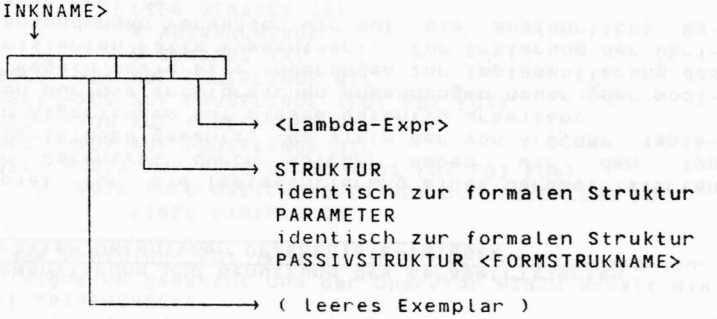
(<INKNAME> <OP-NAME> <ARG-LIST>)

in Operationsaufrufe der alten Form

(< **FORMSTRUKNAME** > <OP-NAME> <INKNAME> <ARG-LIST>)

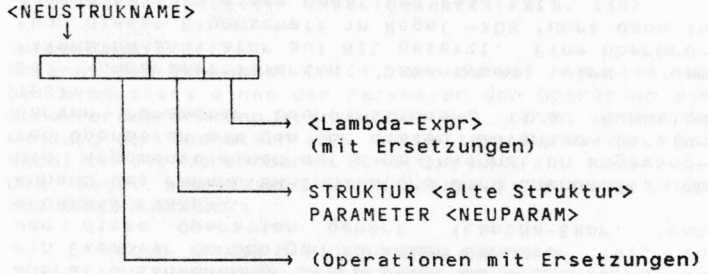
überführt, damit diese mit den durch den Patternmatcher erzeugten Lambda-Formen der formalen Strukturen ausführbar sind (Konservierung des alten ADTCOMP-Systems).

INIT hängt ferner an den Global-Value von <INKNAME> ein leeres Exemplar des entsprechenden Typs durch Aufruf der DO-Operation.
Ist ein Exemplar einer formalen Struktur erfolgreich initialisiert, so existiert folgende Struktur im LISP-System:



und INIT verabschiedet sich mit der Meldung:
"<INKNAME> als <FORMSTRUKNAME>-Inkarnation verfügbar!" (Siehe auch Funktionsbeschreibung INIT in 5.3.1.)

INSERT (neudefiniert)
Aufruf: (INSERT <NEUPARAM> IN <FORMSTRUKNAME>)
INSERT überprüft zunächst, ob unter dem Namen <NEUSTRUKNAME>, der sich durch Konkatenation aus dem Wert des STRUKTUR-Indikators von <FORMSTRUKNAME> und <NEUPARAM> zusammensetzt, schon eine formale Struktur im LISP-System existiert. Ist dies der Fall, wird INSERT mit einer entsprechenden Meldung beendet. Andernfalls kopiert INSERT die formale Struktur <FORMSTRUKNAME> (wenn es sich um eine parametrisierte formale Struktur handelt; Fehlermeldung sonst) und ersetzt überall, im Global-Value, in der Property-Liste und in der Funktions-Definitionszelle den alten durch den neuen Parameter <NEUPARAM>.
Ergebnis von INSERT ist die folgende LISP-Struktur:



und die Meldung:
"<NEUSTRUKNAME> als formale Struktur verfügbar!".
(Siehe auch Funktionsbeschreibung INSERT in 5.3.1.)

TEST (neudefiniert)
Aufruf: durch die Schreiboperatoren parametrisierter formaler Strukturen
Bei der Implementierung von Schreiboperatoren parametrisierter Datentypen wird durch die erweiterte Regel ->08 der "I-atomare-Schlüssel" CHECK gesetzt. Die Regel I38 ersetzt diesen Schlüssel durch den COND-Ausdruck

```
(COND ((TEST) (T (FEHLER) (RETURN)))
```

in der Definition des Schreiboperators.
Bei Benutzung der Schreiboperation prüft die Funktion TEST, ob der Wert des PARAMETER-Indikators der Inkarnation mit dem Wert des PASSIVSTRUKTUR-Indikators der Argumentstruktur übereinstimmt. Ist dies der Fall, wird die Schreiboperation ausgeführt, sonst erfolgt eine Fehlermeldung (durch Aufruf der Funktion FEHLER) und die Schreiboperation wird abgebrochen.
(Siehe auch Funtionsbeschreibung TEST in 5.3.1.)

FEHLER (neudefiniert)
Aufruf: durch die Schreiboperatoren parametrisierter formaler Strukturen (s.a. Funktion TEST). Geht der Test bei Ausführung einer Schreiboperation negativ aus, wird die Funktion FEHLER aufgerufen. Sie erzeugt folgende Fehlermeldung:

"Strukturfehler: Das Argument <Argument> hat keine <Parameter>-Struktur, wie sie von <Inkarnation> gefordert wird." (siehe auch Funktionsbeschreibung FEHLER in 5.3.1.)

3.2. Implementierung und Benutzung des parametrisierten abstrakten Datentypen OFFENELISTE-INTEGER

Als Beispiel für die Implementierung eines parametrisierten abstrakten Datentyps durch ADTCOMP haben wir den Typ OFFENELISTE-INTEGER gewählt, da viele der von ALGCOMP implementierten Algorithmen auf diesem Datentyp arbeiten.
Hier werden nur die erfolgreichen Anwendungen neuer oder modifizierter Regeln sowie alle Änderungen zur Implementierung des unparametrisierten Falls kommentiert. Zur Erklärung der übrigen Regelanwendungen verweisen wir auf die ausführliche Beschreibung in [EKT 79].

Der Typ OFFENELISTE ist wie folgt spezifiziert:

```
DATENTYP:                    OFFENELISTE D;

KOMPONENTENMENGE:            X;

SONSTIGE MENGEN:             A = NATZAHL;

OPERATOREN:                  HINZU:   D + A + X -> D,
                             ENTFERN: D + A     -> D,
                             LIES:    D + A     -> X;

HILFSOPERATOREN:             KONS:    D +     X -> D;

AXIOME:    (HINZU DO A X) = (KONS DO X) FALLS (EQ A 1),
                          = UNDEF FALLS (NOT (EQ A 1)),
 (HINZU (KONS D2 X2) A X) = (KONS (HINZU D2 (SUB1 A) X) X2)
                             FALLS (NOT (EQ A 1)),
                          = (KONS (KONS D2 X2) X)
                             FALLS (EQ A 1),
           (ENTFERN DO A) = UNDEF,
 (ENTFERN (KONS D2 X2) A) = X2 FALLS (EQ A 1),
                          = (KONS (ENTFERN D2 (SUB1 A)) X2)
                             FALLS (NOT (EQ A 1)),
             (LIES DO A) = UNDEF,
    (LIES (KONS D2 X2) A) = X2 FALLS (EQ A 1),
```

```
                    = (LIES D2 (SUB1 A))
                      FALLS (NOT (EQ A 1))  ;;
```

Sei obige Spezifikation in der Datei "OFFENELISTE" abgelegt. Dann bewirkt der Aufruf der Form

```
(PARAMETRISIERE 'OFFENELISTE 'INTEGER)
```

die Implementierung des mit Integer parametrisierten Typs Offene-Liste.

Es folgt das vollständige Protokoll der Implementierung. Zunächst die Aufbau-Phase (Einlesen der Spezifikation):

```
ADT-EINGABE UEBER DATEI: OFFENELISTE
ERGEBNIS    :       D NAME (OFFENELISTE-INTEGER)
ERGEBNIS    :       X KOMPONENTENMENGE
ERGEBNIS    :       A MENGE
ERGEBNIS    :       A MENGE (NATZAHL)
ERGEBNIS    :       HINZU OP (IMPLEM)
ERGEBNIS    :       ENTFERN OP (IMPLEM)
ERGEBNIS    :       LIES OP (IMPLEM)
ERGEBNIS    :       KONS OP (HILFE)
ADT-EINGABE BEENDET.

  ((X A)
   NIL
   (ENTFERN HINZU KONS LIES)
   ((FO1 HINZU D A X -%> D)
    (FO2 ENTFERN D A -%> D)
    (FO3 LIES D A -%> X)
    (FO4 KONS D X -%> D))
   ((AO1 %(HINZU DO A X %) = %(KONS DO X %)
                             FALLS %(EQ A 1 %))
    (AO2 %(HINZU DO A X %) = UNDEF
                             FALLS %(NOT %(EQ A 1 %) %))
    (AO3 %(HINZU %(KONS D2 X2 %) A X %) =
                 %(KONS %(HINZU D2 %(SUB1 A %) X %) X2 %)
                 FALLS %(NOT %(EQ A 1 %) %))
    (AO4 %(HINZU %(KONS D2 X2 %) A X %) =
                 %(KONS %(KONS D2 X2 %) X %)
                 FALLS %(EQ A 1 %))
    (AO5 %(ENTFERN DO A %) = UNDEF)
    (AO6 %(ENTFERN %(KONS D2 X2 %) A %) = D2
                                          FALLS %(EQ A 1 %))
    (AO7 %(ENTFERN %(KONS D2 X2 %) A %) =
         %(KONS %(ENTFERN D2 %(SUB1 A %) %) X2 %)
         FALLS %(NOT %(EQ A 1 %) %))
    (AO8 %(LIES DO A %) = UNDEF)
    (AO9 %(LIES %(KONS D2 X2 %) A %) = X2
                                       FALLS %(EQ A 1 %))
    (A10 %(LIES %(KONS D2 X2 %) A %) =
                %(LIES D2 %(SUB1 A %) %)
                FALLS %(NOT %(EQ A 1 %) %)))
```



```
  ((*ADT . D)
   (*ADT-NAME . OFFENELISTE-INTEGER)
   (S . S)
   (D . D)
   (X . X)))
```

Es folgt die Funktionalitäts-Phase (->-Phase):

```
"START  -> PHASE"
PROBIERREGEL: ->01 AUF PRUEFLINGE (FO1 FO2 FO4)
---REGEL    : ->01 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO1
ERGEBNIS    :      HINZU FORTSETZBAR (EINFACH)
---REGEL    : ->01 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO2
ERGEBNIS    :      ENTFERN FORTSETZBAR (EINFACH)
---REGEL    : ->01 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO4
ERGEBNIS    :      KONS FORTSETZBAR (EINFACH)
PROBIERREGEL: ->02 AUF PRUEFLINGE (FO1 FO2 FO4)
PROBIERREGEL: ->02A AUF PRUEFLINGE (FO3)
---REGEL    : ->02A ERFOLGREICH ANGEWANDT AUF PRUEFLING FO3
ERGEBNIS    :      LIES LIEST
PROBIERREGEL: ->03 AUF PRUEFLINGE (FO1 FO2 FO3)
---REGEL    : ->03 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO1
ERGEBNIS    :      HINZU GEMAESS (A)
ERGEBNIS    :      A AUSWAHLMENGE
ERGEBNIS    :      D GEMAESS (A)
---REGEL    : ->03 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO2
ERGEBNIS    :      ENTFERN GEMAESS (A)
ERGEBNIS    :      A AUSWAHLMENGE
ERGEBNIS    :      D GEMAESS (A)
---REGEL    : ->03 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO3
ERGEBNIS    :      LIES GEMAESS (A)
ERGEBNIS    :      A AUSWAHLMENGE
ERGEBNIS    :      D GEMAESS (A)
PROBIERREGEL: ->04 AUF PRUEFLINGE NIL
PROBIERREGEL: ->05 AUF PRUEFLINGE (FO1 FO2 FO3)
PROBIERREGEL: ->06 AUF PRUEFLINGE NIL
PROBIERREGEL: ->07 AUF PRUEFLINGE NIL
PROBIERREGEL: ->07A AUF PRUEFLINGE (FO1 FO2 FO3 FO4)
---REGEL    : ->07A ERFOLGREICH ANGEWANDT AUF PRUEFLING FO1
ERGEBNIS    :      HINZU PARAMCHECK
```

Im Muster des Prüflings FO1 (HINZU) wird das "X" (Komponentenmenge) erfolgreich gematcht und der Operator HINZU erhält die Eigenschaft "PARAMCHECK".
Das gleiche gilt für den Hilfsoperator KONS:

```
---REGEL    : ->07A ERFOLGREICH ANGEWANDT AUF PRUEFLING FO4
ERGEBNIS    :      KONS PARAMCHECK
PROBIERREGEL: ->08 AUF PRUEFLINGE (FO1 FO2 FO3 FO4)
---REGEL    : ->08 ERFOLGREICH ANGEWANDT AUF PRUEFLING FO1
ERGEBNIS    :      HINZU FUNK ((LAMBDA (D A X) CHECK FORM D))
ERGEBNIS    :      HINZU PARAMS (A X)
```

Aufgrund der von Regel ->07A für den Operator HINZU gesetzten Eigenschaft "PARAMCHECK" und weil wieder erfolgreich das "X" (Komponentenmenge) im Muster gematcht werden kann, fügt die

Regel ->08 den "I-atomaren-Schlüssel" "CHECK" in die vorläufige Form des Operators HINZU ein.
Da es sich bei dem Prüfling F04 (KONS) um einen Hilfsoperator handelt, der nicht implementiert werden soll, kommt in diesem Fall die Regel ->08 nicht zur Anwendung; wohl aber für die zu implementierenden Operatoren ENTFERN und LIES:

```
---REGEL    : ->08 ERFOLGREICH ANGEWANDT AUF PRUEFLING F02
ERGEBNIS    :      ENTFERN FUNK ((LAMBDA (D A) FORM D))
ERGEBNIS    :      ENTFERN PARAMS (A)
---REGEL    : ->08 ERFOLGREICH ANGEWANDT AUF PRUEFLING F03
ERGEBNIS    :      LIES FUNK ((LAMBDA (D A) FORM))
ERGEBNIS    :      LIES PARAMS (A)
-> PHASE BEENDET
```

Es folgt die Axiomen-Phase (=-Phase):
In den Axiomen A01 und A04 wird der Operator HINZU durch den elementaren Hilfsoperator KONS ersetzt. Unter bestimmten Bedingungen haben HINZU und KONS also die gleiche Wirkung, d.h. "HINZU SCHREIBT" (Regel =-06A und =-06B).

```
"START  = PHASE"
PROBIERREGEL: =01 AUF PRUEFLINGE (A09)
---REGEL    : =01 ERFOLGREICH ANGEWANDT AUF PRUEFLING A09
ERGEBNIS    :      LIES LIEST (FALLS (EQ A 1))
ERGEBNIS    :      KONS SCHREIBT (ELEM)
ERGEBNIS    :      *E BEZEICHNET (KONS)
PROBIERREGEL: =03 AUF PRUEFLINGE (A02 A05 A08)
---REGEL    : =03 ERFOLGREICH ANGEWANDT AUF PRUEFLING A02
ERGEBNIS    :      HINZU UNDEFINIERT
                   (FALLS (AND (EQ D DO)(NOT (EQ A 1))))
---REGEL    : =03 ERFOLGREICH ANGEWANDT AUF PRUEFLING A05
ERGEBNIS    :      ENTFERN UNDEFINIERT (FALLS (EQ D DO))
---REGEL    : =03 ERFOLGREICH ANGEWANDT AUF PRUEFLING A08
ERGEBNIS    :      LIES UNDEFINIERT (FALLS (EQ D DO))
PROBIERREGEL: =04 AUF PRUEFLINGE (A03 A04 A07 A10)
---REGEL    : =04 ERFOLGREICH ANGEWANDT AUF PRUEFLING A03
ERGEBNIS    :      HINZU DURCHLAEUFT (FALLS (NOT (EQ A 1)))
---REGEL    : =04 ERFOLGREICH ANGEWANDT AUF PRUEFLING A07
ERGEBNIS    :      ENTFERN DURCHLAEUFT (FALLS (NOT (EQ A 1)))
---REGEL    : =04 ERFOLGREICH ANGEWANDT AUF PRUEFLING A10
ERGEBNIS    :      LIES DURCHLAEUFT (FALLS (NOT (EQ A 1)))
PROBIERREGEL: =05 AUF PRUEFLINGE (A03 A04 A07 A10)
PROBIERREGEL: =05A AUF PRUEFLINGE (A03 A04 A07 A10)
PROBIERREGEL: =05B AUF PRUEFLINGE (A03 A04 A07 A10)
PROBIERREGEL: =06A AUF PRUEFLINGE (A01)
---REGEL    : =06A ERFOLGREICH ANGEWANDT AUF PRUEFLING A01
ERGEBNIS    :      HINZU SCHREIBT
                   (FALLS (AND (EQ D DO)(EQ A 1)))
PROBIERREGEL: =06B AUF PRUEFLINGE (A03 A04 A07 A10)
---REGEL    : =06B ERFOLGREICH ANGEWANDT AUF PRUEFLING A04
ERGEBNIS    :      HINZU SCHREIBT (FALLS (EQ A 1))
GEODERTE BEDINGUNG ERGEBNIS: HINZU SCHREIBT (FALLS (EQ A 1))
PROBIERREGEL: =06C AUF PRUEFLINGE (A03 A06 A07)
---REGEL    : =06C ERFOLGREICH ANGEWANDT AUF PRUEFLING A06
ERGEBNIS    :      ENTFERN LOESCHT (FALLS (EQ A 1))
```



```
PROBIERREGEL: =07 AUF PRUEFLINGE (A03 A07 A10)
PROBIERREGEL: =08 AUF PRUEFLINGE (A03 A07 A10)
PROBIERREGEL: =09 AUF PRUEFLINGE (A06)
PROBIERREGEL: =10 AUF PRUEFLINGE (A03 A07 A10)
---REGEL    : =10 ERFOLGREICH ANGEWANDT AUF PRUEFLING A03
ERGEBNIS    :      A AUSWAHLMENGE (ZAEHLER)
ERGEBNIS    :      HINZU ZAEHLT (FALLS (NOT (EQ A 1)))
---REGEL    : =10 ERFOLGREICH ANGEWANDT AUF PRUEFLING A07
ERGEBNIS    :      A AUSWAHLMENGE (ZAEHLER)
ERGEBNIS    :      ENTFERN ZAEHLT (FALLS (NOT (EQ A 1)))
---REGEL    : =10 ERFOLGREICH ANGEWANDT AUF PRUEFLING A10
ERGEBNIS    :      A AUSWAHLMENGE (ZAEHLER)
ERGEBNIS    :      LIES ZAEHLT (FALLS (NOT (EQ A 1)))
PROBIERREGEL: =11 AUF PRUEFLINGE (A03 A07 A10)
PROBIERREGEL: =16 AUF PRUEFLINGE (A03 A07 A10)
PROBIERREGEL: =17 AUF PRUEFLINGE (A03 A07 A10)
PROBIERREGEL: =19 AUF PRUEFLINGE NIL
PROBIERREGEL: =20 AUF PRUEFLINGE NIL
= PHASE BEENDET
```

Es folgt die Zusammenfassungs-Phase (&-Phase):

```
"START  & PHASE"
NOCH MOEGLICHE REGELN:  &01 &02 &04 &05 &06 &07 &08 &09A &10
                        &10A &11A &11B &12 &13A &13B &14A &14B
                        &15A &15B &16A &16B &17A &17B &18A
                        &18B &19A &19B &21 &22 &24 &25 &27 &29
                        &29A &30 &31 &32 &33
PROBIERREGEL: &01 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
---REGEL    : &01 ERFOLGREICH ANGEWANDT
ERGEBNIS    :      D LISTE
NOCH MOEGLICHE REGELN:  &04 &05 &06 &07 &08 &09A &10 &10A &11A
                        &11B &12 &13A &13B &14A &14B &15A &15B
                        &16A &16B &17A &17B &18A &18B &19A
                        &19B &21 &22 &24 &25 &27 &29 &29A &31
                        &32 &33
PROBIERREGEL: &04 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &05 AUF PRUEFLINGE (ENTFERN HINZU LIES)
PROBIERREGEL: &06 AUF PRUEFLINGE (ENTFERN HINZU LIES)
--- AUFSPALTUNG ENTFERN FUER *OP1 :
    ---REGEL    : &06 ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                  ENTFERN
ERGEBNIS    :      ENTFERN ZAEHLT
                   (NEUSTES FALLS (NOT (EQ A 1)))
--- AUFSPALTUNG HINZU FUER *OP1 :
    ---REGEL    : &06 ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                  HINZU
ERGEBNIS    :      HINZU ZAEHLT (NEUSTES FALLS (NOT (EQ A 1)))
--- AUFSPALTUNG LIES FUER *OP1 :
    ---REGEL    : &06 ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                  LIES
ERGEBNIS    :      LIES ZAEHLT (NEUSTES FALLS (NOT (EQ A 1)))
PROBIERREGEL: &07 AUF PRUEFLINGE (ENTFERN HINZU LIES)
PROBIERREGEL: &08 AUF PRUEFLINGE (ENTFERN HINZU LIES)
PROBIERREGEL: &09A AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &10 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
```

```
PROBIERREGEL: &10A AUF PRUEFLINGE NIL
PROBIERREGEL: &11A AUF PRUEFLINGE (ENTFERN)
PROBIERREGEL: &11B AUF PRUEFLINGE (ENTFERN HINZU LIES)
—— AUFSPALTUNG LIES FUER *OP1 :
      ——REGEL    : &11B ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                    LIES
ERGEBNIS     :       D AUFGEBAUT
ERGEBNIS     :       LIES ZAEHLT (VORNE FALLS (NOT (EQ A 1)))
PROBIERREGEL: &12 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &13A AUF PRUEFLINGE (ENTFERN)
PROBIERREGEL: &13B AUF PRUEFLINGE (ENTFERN HINZU LIES)
—— AUFSPALTUNG LIES FUER *OP1 :
      ——REGEL    : &13B ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                    LIES
ERGEBNIS     :       KONS SCHREIBT (VORNE)
NOCH MOEGLICHE REGELN:  &15A &15B &17A &17B &19A &19B &21 &22
                        &24 &25 &27 &29 &29A &31 &32 &33
PROBIERREGEL: &15A AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &15B AUF PRUEFLINGE (ENTFERN HINZU LIES)
—— AUFSPALTUNG ENTFERN FUER *OP1 :
      ——REGEL    : &15B ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                    ENTFERN
ERGEBNIS     :       ENTFERN ZAEHLT (VORNE FALLS (NOT (EQ A 1)))
—— AUFSPALTUNG HINZU FUER *OP1 :
      ——REGEL    : &15B ERFOLGREICH ANGEWANDT AUF PRUEFLING
                                                    HINZU
ERGEBNIS     :       HINZU ZAEHLT (VORNE FALLS (NOT (EQ A 1)))
PROBIERREGEL: &17A AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &17B AUF PRUEFLINGE (ENTFERN HINZU LIES)
PROBIERREGEL: &19A AUF PRUEFLINGE (ENTFERN)
PROBIERREGEL: &19B AUF PRUEFLINGE (ENTFERN HINZU LIES)
PROBIERREGEL: &21 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
——REGEL     : &21 ERFOLGREICH ANGEWANDT
ERGEBNIS     :       D LISTE (EV)
NOCH MOEGLICHE REGELN:  &29 &31 &32 &33
PROBIERREGEL: &29 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
——REGEL     : &29 ERFOLGREICH ANGEWANDT
ERGEBNIS     :       D DO-FUNK ((NLAMBDA (OBEN)
                                (SETQ OBEN (CONS))
                                (RPLACA OBEN OBEN)))
NOCH MOEGLICHE REGELN:  &31 &32 &33
PROBIERREGEL: &31 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
PROBIERREGEL: &32 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
——REGEL     : &32 ERFOLGREICH ANGEWANDT
ERGEBNIS     :       DO FUNK ((NLAMBDA (OBEN)
                               (SETQ OBEN (CONS))
                               (RPLACA OBEN OBEN)))
PROBIERREGEL: &33 AUF PRUEFLINGE (ENTFERN HINZU KONS LIES)
& PHASE BEENDET
```


Es folgt die Implementierung-Phase (I-Phase):

```
"START  I PHASE"
NOCH MOEGLICHE REGELN:  I01 I01A I02 I05 I06 I07 I08 I08B I08C
                        I09 I10 I10B I13 I13A I15 I18 I20 I21
                        I22A I22B I22C I22D I23 I27 I29 I30
                        I30A I30B I30C I31 I32 I33 I34 I36 I37
                        I38

—— ICH BEARBEITE HINZU ——
```

Der von Regel ->08 in die Form des Operators HINZU eingefügte "I-atomare-Schlüssel" "CHECK" wird von Regel I38 erkannt und durch einen COND-Ausdruck ersetzt, der durch Aufruf der Funktion TEST bei Anwendung des Operator die Übereinstimmung des angegebenen Parametertyps Integer mit dem Typ des übergebenen Parameters X überprüft.

```
ATOMARER SCHLUESSEL CHECK  , DAZU EVTL. ANWENDBARE REGELN :
                                                      (I38)
——REGEL     : I38 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    ***
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    FORM D>>
ATOMARER SCHLUESSEL FORM  , DAZU EVTL. ANWENDBARE REGELN :
                                          (I01 I01A I02 I05)
——REGEL     : I02 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    ***
    (COND
      UNDEFINIERT
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL UNDEFINIERT  ,
         DAZU EVTL. ANWENDBARE REGELN : (I08 I10 I10B)
——REGEL     : I08 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
```

```
        (RETURN)))
    (COND
      ***
      ((AND (EQ D DO)
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      UNDEFINIERT
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
---REGEL    : I08B ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
        (RETURN)))
    (COND
      ((AND *** (NULL VORNE)
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      UNDEFINIERT
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                      (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
        (RETURN)))
    (COND
      ((AND (NULL *** (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      UNDEFINIERT
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
ATOMARER SCHLUESSEL UNDEFINIERT  ,
        DAZU EVTL. ANWENDBARE REGELN : (I08 I10 I10B)
---REGEL    : I10 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
```



```
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
        (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ***
      ((NULL VORNE)
        SCHREIBT VORNE)
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                      (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
        (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL *** (CDR D))
        SCHREIBT VORNE)
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL SCHREIBT  , DAZU EVTL. ANWENDBARE REGELN :
                                    (I15 I18 I20 I21 I22C I22D)
---REGEL    : I15 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
        (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        ***
        (REPLACE VORNE (CONS X VORNE))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD VORNE))
```

```
        D)
       ((EQ A 1)
         SCHREIBT VORNE)
       (T (APPLY* HINZU VORNE PARAMS)))
     D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE *** (CDR D)
                 (CONS X VORNE))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD VORNE))
        D)
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X *** (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD VORNE))
        D)
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
```



```
                                                (I07 I08B I08C I09)
NICHTATOMARER SCHLUESSEL RPLACA ,
              DAZU EVTL. ANWENDBARE REGELN (I30A I31 I32)
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD *** (CDR D)))
        D)
      ((EQ A 1)
        SCHREIBT VORNE)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                                (I07 I08B I08C I09)
ATOMARER SCHLUESSEL SCHREIBT  , DAZU EVTL. ANWENDBARE REGELN :
                                  (I15 I18 I20 I21 I22C I22D)
---REGEL    : I15 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        ***
        (REPLACE VORNE (CONS X VORNE))
        (AND (EQ (CAR URD)
                 D)
```

```
            (RPLACA URD VORNE))
          D)
        (T (APPLY* HINZU VORNE PARAMS)))
      D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (REPLACE *** (CDR D)
                 (CONS X VORNE))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD VORNE))
        D)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
```



```
        (REPLACE (CDR D)
                 (CONS X *** (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD VORNE))
        D)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
NICHTATOMARER SCHLUESSEL RPLACA , DAZU EVTL. ANWENDBARE REGELN
                                                (I30A I31 I32)
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD *** (CDR D)))
        D)
      (T (APPLY* HINZU VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
```

```
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
       ((EQ A 1)
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
       (T (APPLY* HINZU *** (CDR D)
                  PARAMS)))
    D>>
ATOMARER SCHLUESSEL PARAMS  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (IO6)
---REGEL     : IO6 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T (APPLY* HINZU (CDR D)
                 *** A X)))
    D>>
ERGEBNIS DER I PHASE FUER HINZU :

<ERG-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
```



```
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T (APPLY* HINZU (CDR D)
                 A X)))
    D>>


--- ICH BEARBEITE ENTFERN ---
ATOMARER SCHLUESSEL FORM  , DAZU EVTL. ANWENDBARE REGELN :
                                          (IO1 IO1A IO2 IO5)
---REGEL     : IO2 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    ***
    (COND
      UNDEFINIERT
      ((EQ A 1)
        LOESCHT VORNE)
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL UNDEFINIERT  ,
         DAZU EVTL. ANWENDBARE REGELN : (IO8 I10 I10B)
---REGEL     : IO8 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ***
      ((EQ D DO)
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LOESCHT VORNE)
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (IO7 IO8B IO8C IO9)
---REGEL     : IO8B ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
```

```
  <LAMBDA (D A)
    (COND
      (*** (NULL VORNE)
           (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LOESCHT VORNE)
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL *** (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LOESCHT VORNE)
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
ATOMARER SCHLUESSEL LOESCHT  , DAZU EVTL. ANWENDBARE REGELN :
                                          (I22A I22B I23 I27)
---REGEL    : I23 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        ***
        (AND (EQ (CAR URD)
                 VORNE)
             (RPLACA URD D))
        (REPLACE VORNE (CDR VORNE)))
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (AND (EQ (CAR URD)
                 ***
                 (CDR D))
             (RPLACA URD D))
```



```
        (REPLACE VORNE (CDR VORNE)))
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
NICHTATOMARER SCHLUESSEL RPLACA , DAZU EVTL. ANWENDBARE REGELN
                                          (I30A I31 I32)
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (REPLACE *** (CDR D)
                 (CDR VORNE)))
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (REPLACE (CDR D)
                 (CDR *** (CDR D>
      (T (APPLY* ENTFERN VORNE PARAMS)))
    D>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (I33)
---REGEL    : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (REPLACE (CDR D)
                 (CDR (CDR D>
      (T (APPLY* ENTFERN *** (CDR D)
                 PARAMS)))
```

```
    D>>
ATOMARER SCHLUESSEL PARAMS  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I06)
---REGEL     : I06 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (REPLACE (CDR D)
                 (CDR (CDR D>
      (T (APPLY* ENTFERN (CDR D)
                 *** A)))
    D>>
ERGEBNIS DER I PHASE FUER ENTFERN :

<ERG-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (REPLACE (CDR D)
                 (CDR (CDR D>
      (T (APPLY* ENTFERN (CDR D)
                 A)))
    D>>


--- ICH BEARBEITE LIES ---
ATOMARER SCHLUESSEL FORM  , DAZU EVTL. ANWENDBARE REGELN :
                                              (I01 I01A I02 I05)
---REGEL     : I02 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    ***
    (COND
      UNDEFINIERT
      ((EQ A 1)
        LIEST VORNE)
      (T (APPLY* LIES VORNE PARAMS>>
ATOMARER SCHLUESSEL UNDEFINIERT  ,
        DAZU EVTL. ANWENDBARE REGELN : (I08 I10 I10B)
---REGEL     : I08 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
```



```
    (COND
      ***
      ((EQ D DO)
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LIEST VORNE)
      (T (APPLY* LIES VORNE PARAMS>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
---REGEL     : I08B ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      (*** (NULL VORNE)
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LIEST VORNE)
      (T (APPLY* LIES VORNE PARAMS>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL     : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL *** (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        LIEST VORNE)
      (T (APPLY* LIES VORNE PARAMS>>
NICHTATOMARER SCHLUESSEL EQ , DAZU EVTL. ANWENDBARE REGELN
                                          (I07 I08B I08C I09)
ATOMARER SCHLUESSEL LIEST  , DAZU EVTL. ANWENDBARE REGELN :
                                                  (I13 I13A)
---REGEL     : I13 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        ***
        (KOMP VORNE))
      (T (APPLY* LIES VORNE PARAMS>>
ATOMARER SCHLUESSEL KOMP  , DAZU EVTL. ANWENDBARE REGELN :
                                                  (I36 I37)
---REGEL     : I36 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
```

```
        (*** CAR VORNE))
       (T (APPLY* LIES VORNE PARAMS>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL     : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR *** (CDR D)))
      (T (APPLY* LIES VORNE PARAMS>>
ATOMARER SCHLUESSEL VORNE  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I33)
---REGEL     : I33 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR (CDR D)))
      (T (APPLY* LIES *** (CDR D)
                 PARAMS>>
ATOMARER SCHLUESSEL PARAMS  , DAZU EVTL. ANWENDBARE REGELN :
                                                        (I06)
---REGEL     : I06 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR (CDR D)))
      (T (APPLY* LIES (CDR D)
                 *** A>>
ERGEBNIS DER I PHASE FUER LIES :

<ERG-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR (CDR D)))
      (T (APPLY* LIES (CDR D)
                 A>>
I PHASE BEENDET
```


Es folgt die Bereinigungs-Phase (B-Phase):

```
"START  B PHASE"

--- ICH BEARBEITE HINZU ---
NICHTATOMARER SCHLUESSEL COND , DAZU EVTL. ANWENDBARE REGELN
                                                        (B04)
KEINE REGEL ANWENDBAR AUF PRUEFLING HINZU UND SCHLUESSEL COND
NICHTATOMARER SCHLUESSEL COND , DAZU EVTL. ANWENDBARE REGELN
                                                        (B04)
KEINE REGEL ANWENDBAR AUF PRUEFLING HINZU UND SCHLUESSEL COND
ATOMARER SCHLUESSEL REPLACE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (B01 B02)
---REGEL     : B02 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (*** RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (REPLACE (CDR D)
                 (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T (APPLY* HINZU (CDR D)
                 A X)))
    D>>
ATOMARER SCHLUESSEL REPLACE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (B01 B02)
---REGEL     : B02 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
```

```
      ((NULL (CDR D))
        (RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (*** RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T (APPLY* HINZU (CDR D)
                 A X)))
    D>>
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                        (B07 B08)
---REGEL     : B07 ERFOLGREICH ANGEWANDT AUF PRUEFLING HINZU

<NEUE-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T *** (APPLY* HINZU (CDR D)
                     (SUB1 A)
                     X)))
    D>>
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                        (B07 B08)
KEINE REGEL ANWENDBAR AUF PRUEFLING HINZU UND
                          SCHLUESSEL APPLY*
ERGEBNIS DER B PHASE FUER HINZU :

<ERG-FUNK
  <LAMBDA (D A X)
    (COND
      ((TEST 1 OFFENELISTE-INTEGER))
      (T (FEHLER 1 OFFENELISTE-INTEGER)
```



```
         (RETURN)))
    (COND
      ((AND (NULL (CDR D))
            (NOT (EQ A 1)))
        (RETURN (QUOTE UNDEF)))
      ((NULL (CDR D))
        (RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      ((EQ A 1)
        (RPLACD D (CONS X (CDR D)))
        (AND (EQ (CAR URD)
                 D)
             (RPLACA URD (CDR D)))
        D)
      (T (APPLY* HINZU (CDR D)
                 (SUB1 A)
                 X)))
    D>>


--- ICH BEARBEITE ENTFERN ---
NICHTATOMARER SCHLUESSEL COND , DAZU EVTL. ANWENDBARE REGELN
                                                        (B04)
KEINE REGEL ANWENDBAR AUF PRUEFLING ENTFERN UND
                           SCHLUESSEL COND
ATOMARER SCHLUESSEL REPLACE  , DAZU EVTL. ANWENDBARE REGELN :
                                                    (B01 B02)
---REGEL     : B02 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (*** RPLACD D (CDR (CDR D>
      (T (APPLY* ENTFERN (CDR D)
                 A)))
    D>>
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                        (B07 B08)
---REGEL     : B07 ERFOLGREICH ANGEWANDT AUF PRUEFLING ENTFERN

<NEUE-FUNK
  <LAMBDA (D A)
    <COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
```

```
                (CDR D))
            (RPLACA URD D))
        (RPLACD D (CDR (CDR D>
      (T *** (APPLY* ENTFERN (CDR D)
                     (SUB1 A>
   D>>
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                    (B07 B08)
KEINE REGEL ANWENDBAR AUF PRUEFLING ENTFERN UND
                          SCHLUESSEL APPLY*
ERGEBNIS DER B PHASE FUER ENTFERN :

<ERG-FUNK
  <LAMBDA (D A)
    <COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      <(EQ A 1)
        (AND (EQ (CAR URD)
                 (CDR D))
             (RPLACA URD D))
        (RPLACD D (CDR (CDR D>
      (T (APPLY* ENTFERN (CDR D)
                 (SUB1 A>
    D>>


—— ICH BEARBEITE LIES ——
NICHTATOMARER SCHLUESSEL COND , DAZU EVTL. ANWENDBARE REGELN
                                                    (B04)
KEINE REGEL ANWENDBAR AUF PRUEFLING LIES UND SCHLUESSEL COND
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                    (B07 B08)
——REGEL    : B07 ERFOLGREICH ANGEWANDT AUF PRUEFLING LIES

<NEUE-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR (CDR D)))
      (T *** (APPLY* LIES (CDR D)
                     (SUB1 A>>
NICHTATOMARER SCHLUESSEL APPLY* , DAZU EVTL. ANWENDBARE REGELN
                                                    (B07 B08)
KEINE REGEL ANWENDBAR AUF PRUEFLING LIES UND SCHLUESSEL APPLY*
ERGEBNIS DER B PHASE FUER LIES :

<ERG-FUNK
  <LAMBDA (D A)
    (COND
      ((NULL (CDR D))
        (RETURN (QUOTE UNDEF)))
      ((EQ A 1)
        (CAR (CDR D)))
```



```
      (T (APPLY* LIES (CDR D)
                 (SUB1 A>>
B PHASE BEENDET
```

Damit sind die drei Operatoren von OFFENELISTE-INTEGER fertig verfeinert und müssen noch in echte LAMBDA-Ausdrücke (also LISP-Funktions-Definitionen) überführt werden. Dies geschieht in der LAMBDA-Phase (L-Phase):

```
"START L PHASE"
L PHASE BEENDET
```

VERWENDET WURDEN DIE REGELN: ->01 ->02A ->03 ->07A ->08 =01 =03 =04 =06A =06B =06C =10 &01 &06 &11B &13B &15B &21 &29 &32 I38 I02 I08 I08B I33 I10 I15 I06 I23 I13 I36 B02 B07

Damit ist die Implementierung beendet.

Der fertig implementierte Datentyp sieht folgendermaßen aus:

```
(DEFINEQ

<OFFENELISTE-INTEGER
  <NLAMBDA (OP . DLISTE)
    (SETQ ARGUMENTE (CDR DLISTE))
    (SETQ DATSTRUK (CAR DLISTE))
    (SETQ DLISTE (APPEND (LIST (EVAL (CAR DLISTE)))
                         (CDR DLISTE)))
    (PROG ((URD (CAR DLISTE)))
          (COND
            ((ASSOC OP OFFENELISTE-INTEGER)
              NIL)
            (T (PRINTL "ANWENDUNGSFEHLER:" OP "IST KEINE"
                       (QUOTE OFFENELISTE-INTEGER)
                       "OPERATION!")
               (TERPRI)
               (RETURN)))
          (RETURN (EVALA (QUOTE (APPLY (EVAL OP)
                                       DLISTE))
                         OFFENELISTE-INTEGER>>
)
  (RPAQQ OFFENELISTE-INTEGER ((DO NLAMBDA (OBEN)
                                  (SETQ OBEN (CONS))
                                  (RPLACA OBEN OBEN))
          <LIES LAMBDA (D A)
                (COND
                  ((NULL (CDR D))
                    (RETURN (QUOTE UNDEF)))
                  ((EQ A 1)
                    (CAR (CDR D)))
                  (T (APPLY* LIES (CDR D)
                             (SUB1 A>
```

```
      (ENTFERN LAMBDA (D A)
               <COND
                 ((NULL (CDR D))
                   (RETURN (QUOTE UNDEF)))
                 <(EQ A 1)
                   (AND (EQ (CAR URD)
                            (CDR D))
                        (RPLACA URD D))
                   (RPLACD D (CDR (CDR D>
                  (T (APPLY* ENTFERN (CDR D)
                             (SUB1 A>
               D)
     (HINZU LAMBDA (D A X)
            (COND
              ((TEST 1 OFFENELISTE-INTEGER))
              (T (FEHLER 1 OFFENELISTE-INTEGER)
                 (RETURN)))
            (COND
              ((AND (NULL (CDR D))
                    (NOT (EQ A 1)))
                (RETURN (QUOTE UNDEF)))
              ((NULL (CDR D))
                (RPLACD D (CONS X (CDR D)))
                (AND (EQ (CAR URD)
                         D)
                     (RPLACA URD (CDR D)))
                D)
              ((EQ A 1)
                (RPLACD D (CONS X (CDR D)))
                (AND (EQ (CAR URD)
                         D)
                     (RPLACA URD (CDR D)))
                D)
              (T (APPLY* HINZU (CDR D)
                         (SUB1 A)
                         X)))
            D)))
   (PUTPROPS OFFENELISTE-INTEGER
             STRUKTUR OFFENELISTE PARAMETER INTEGER)
STOP
```

Zur Benutzung von OFFENELISTE-INTEGER

Am Beispiel verschiedener Operationsaufrufe soll jetzt das Arbeiten mit dem implementierten Datentyp OFFENELISTE-INTEGER demonstriert werden.
Zur Schreibweise: Fettgedruck sind die Benutzereingaben; die folgenden Systemantworten sind wieder normalgedruckt.
Wir gehen davon aus, das die oben erzeugte Implementierung, d.h. die "Formale Struktur" OFFENELISTE-INTEGER dem INTERLISP-System bekannt ist, z.B. durch Laden der Datei, auf die die Implementierung gerettet wurde. Als nachstes muß die Datei

LISP.DATA.ADT-EX.nn mit dem Kommando

```
(LOAD 'ADT-EX)
```

geladen werden. (Diese Datei enthält Funktionen, die zum Arbeiten mit ADTCOMP-Implementierungen nötig sind. Siehe auch 5.3.)

Jetzt wollen wir eine Inkarnation von OFFENELISTE-INTEGER mit dem Namen OL-I erzeugen. Zu diesem Zweck rufen wir die Funktion INIT:

```
(INIT OL-I OFFENELISTE-INTEGER)
"OL-I ALS OFFENELISTE-INTEGER-INKARNATION VERFUEGBAR"
```

Unter dem Bezeichner OL-I ist nun ein leeres Exemplar des Datentyps OFFENELISTE-INTEGER verfügbar, das folgende Bindungen hat:

```
(CAR 'OL-I)
((((((((((((((((((((((((((((((&))))))))))))))))))))))))))))))
```

(Der CAR-Verweis der ersten CONS-Zelle zeigt auf sich selbst. Die Folge ist die obige unendliche Klammerungstiefe)

```
(CDR 'OL-I)
(STRUKTUR OFFENELISTE PARAMETER INTEGER
 PASSIVSTRUKTUR OFFENELISTE-INTEGER)

(GETD ' OL-I)
(NLAMBDA (OP . ARGLIST)
  <SETQ FORM
        (APPEND (LIST (QUOTE OFFENELISTE-INTEGER)
                      OP
                      (QUOTE OL-I))
                (MAPCAR ARGLIST
                  (FUNCTION (NLAMBDA (ARG)
                    (COND
                      ((LISTP ARG)
                        (EVAL ARG))
                      ((GETP ARG
                             (QUOTE STRUKTUR))
                        ARG)
                      (T (EVAL ARG>
   (APPLY (CAR FORM)
          (CDR FORM)))
```

Da OL-I zur Zeit noch leer ist, führen Aufrufe des Lese- und Löschoperators zur Ausgabe des Wertes UNDEF:

```
(OL-I LIES 1)
UNDEF

(OL-I ENTFERN 1)
UNDEF
```

Jetzt wollen wir versuchen, die Offene-Liste OL-I zu füllen:

```
(OL-I PUSH 1 503)
ANWENDUNGSFEHLER:
PUSH IST KEINE OFFENELISTE-INTEGER OPERATION!
```

Wir haben irrtümlicherweise die Keller-Operation PUSH auf eine Offene-Liste angewendet. Fehler dieser Art werden durch die Funktionsdefinitionen der Formalen Strukturen, hier OFFENELISTE-INTEGER, erkannt.
Der nächste Versuch:

```
(OL-I HINZU 1 (QUOTE WERNER-ODER-WAS?))
STRUKTURFEHLER: DAS ARGUMENT WERNER-ODER-WAS? HAT KEINE
INTEGER STRUKTUR, WIE SIE VON OL-I GEFORDERT WIRD !
```

führt auch nicht zum gewünschten Ergebnis; Fehler dieser Art werden durch die Funktion TEST erkannt, die von den Schreiboperatoren parametrisierter Datentypen aufgerufen wird.
Der dritte Versuch liefert nun das Richtige:

```
(OL-I HINZU 1 503)
((503) 503)
```

Dies ist jetzt der Wert von OL-I. Wir schreiben noch einige ganze Zahlen in die Liste:

```
(OL-I HINZU 1 87)
((503) 87 503)

(OL-I HINZU 3 512)
((512) 87 503 512)

(OL-I HINZU 2 61)
((512) 87 61 503 512)
```

Wir wenden nun den Leseoperator LIES an:

```
(OL-I LIES 1)
87

(OL-I LIES 3)
503

(OL-I LIES 6)
UNDEF
```

Durch anwenden des Löschoperator ENTFERN soll OL-I wieder geleert werden:

```
(OL-I ENTFERN 3)
((512) 87 61 512)

(OL-I ENTFERN 1)
((512) 61 512)
```



```
(OL-I ENTFERN 5)
UNDEF

(OL-I ENTFERN 1)
((512) 512)

(OL-I ENTFERN 1)
((((((((((((((((((((((((((((((&))))))))))))))))))))))))))))))
```

Weitere Beispiele für das Arbeiten mit von APE erzeugten Strukturen finden sich in 4.1.4. und 4.6.

4. Abstrakte Algorithmen

In diesem Kapitel wird das ALGCOMP-Subsystem des APE, das abstrakt spezifizierte Algorithmen, die den Regeln der Spezifikationssprache (siehe 2.4.,S. 24) genügen, in ablauffähigen LISP-Code übersetzt, beschrieben. Dazu in 4.1. ein Überblick über die Ziele von ALGCOMP, die Funktionen und Regeln sowie ein ausführliches Beispiel einer Algorithmus-Implementierung.
Es folgen zahlreiche Beispiele von Implementierungen, an Hand derer die Arbeitsweise von ALGCOMP erläutert werden soll. Es werden in

in 4.2. numerische Algorithmen,

in 4.3. Suchalgorithmen,

in 4.4. Sortieralgorithmen und

in 4.5. noch einige andere Algorithmen

von der Spezifikation bis zur fertigen Implementierung beschrieben.
Eine Veranschaulichung des Zusammenspiels der verschiedenen von APE erzeugten Konstrukte gibt das Anwendungsbeispiel in 4.6.

4.1. Überblick über das ALGCOMP-System

Zunächst sollen die beabsichtigten und erreichten Ziele des ALGOMP-System erläutert werden.
Eine Beschreibung der verschiedenen Phasen, die von ALGCOMP während einer Implementierung durchlaufen werden, schließt sich in 4.1.2. an, und zwar

in 4.1.2.1. für die Einlese-Phase,

in 4.1.2.2. für die Algorithmus-Konstruktions-Phase und

in 4.1.2.3. für die LISP-Phase.

Es folgen Erläuterungen zu den zur Zeit existierenden Regeln, mit denen alle in dieser Arbeit dokumentierten Algorithmen von ALGCOMP implementiert wurden, in 4.1.3..
Schließlich geben wir ein ausführliches Beispiel einer Algorithmus-Implementierung in 4.1.4. an.

4.1.1. Ziel des Verfahrens

ALGCOMP konstruiert aus abstrakten Algorithmen im Sinne der Spezifikationssprache (siehe 2.4.2.) ausführbaren LISP-Code.

Dabei muß ein abstrakter Algorithmus folgender Syntax genügen:

```
<abstr.-Alg.>   ::= <header><Prod.-System><Kommentar>
<header>        ::= ALGORITHMUS-NAME <alg-name>
                    PARAMETERTYPEN <param-dekl>
                    [ RELATION <relation> ]
                    [ VORAUSSETZUNGEN <voraus.> ]
                    [ HILFSALGORITHMEN <hilfs-alg> ]
<Prod.-System>  ::= ALGORITHMUS
                    <alg-name><param-list> =
                    <Produktionsregelmenge>
<Kommentar>     ::= <erklärender Text>
```

(vereinfacht, genaue Definition siehe 2.4.2.)

Ein abstrakter Algorithmus besteht also aus drei Teilen: dem header, dem Produktionsregelsystem und dem Kommentar.
Der header beinhaltet

- den Namen des Algorithmus, also den Namen der zu implementierenden LISP-Funktion,
- einen Deklarationsteil für formale Parameter, der für jeden Parameter seinen Datentyp, seinen Namen und, für System-Parameter, eine Wertzuweisung definiert und
- weitere Angaben über den zu verwendenden Relationssatz, Vorbedingungen, Hilfsalgorithmen etc.

Das Produktionsregelsystem besteht im wesentlichen aus der Produktionsregelmenge, wobei jede Produktionregel ein Paar <Test> ⟶ <Aktion> ist. <Test> ist ein Boolscher Ausdruck, wird also zu einem Wahrheitswert ausgewertet; <aktion> ist ein Term, der sich aus den formalen Parametern sowie Operations- und Funktionssymbolen der verwendeten Datentypen, Aufrufen von Hilfsalgorithmen und rekursiven Algorithmus-Aufrufen zusammensetzt.

Der Kommentar soll ein die Arbeitsweise des Algorithmus erklärender Text sein.

ALGCOMP ist insbesondere zur Verarbeitung rekursiv definierter Algorithmen ausgelegt. Ein Algorithmus ist dann rekursiv, wenn auf mindestens einer rechten Seite einer Regel der Produktionsregelmenge ein Aufruf des Algorithmus auftritt. Gibt es rechte Seiten, die mehrere Algorithmus-Aufrufe parallel aufweisen, heißt der Algorithmus parallel-rekursiv (Beispiel: QUICK-SORT).

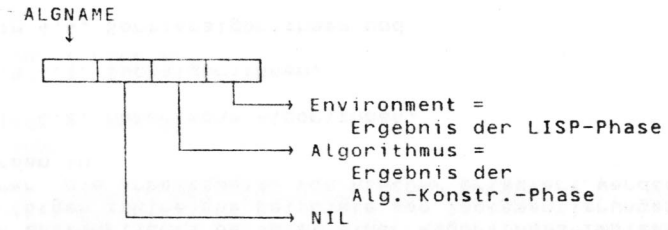
Ein abstrakter Algorithmus wird von ALGCOMP in drei Phasen verarbeitet:

- Eingabe-Phase:
  Die Algorithmus-Spezifikation wird mittels der Funktion RALGSPEC eingelesen und in eine interne Darstellung überführt. Werden hierbei Fehler erkannt, so erfolgt eine entsprechende Fehlermeldung und der Implementierungslauf wird beendet. Der Benutzer muß seine Spezifikation berichtigen und erneut an ALGCOMP übergeben.

- Algorithmus-Konstruktions-Phase:
  In dieser Phase wird aus der internen Darstellung der Produktionsregelmenge ein LISP-COND-Ausdruck konstruiert, der den abstrakten Algorithmus realisiert. Dies geschieht regelgesteuert durch den Pattern-Matcher in sechs Schritten.
  Eine genaue Beschreibung der Algorithmus-Konstruktions-Phase findet sich in dem anschließenden Abschnitt 4.1.2.2.
  Das Ergebnis der Algorithmus-Konstruktion-Phase, der implementierte Algorithmus, wird an der Property-List des Atoms ALGNAME unter dem Indikator ALGORITHMUS abgelegt.

- LISP-Phase:
  Das Ergebnis der vorangegangenen Phasen wird um ein Environment erweitert, das aus den im <header> der Algorithmus-Spezifikation gemachten Angaben konstruiert wird. Resultat ist eine LISP-Funktion mit Namen ALGNAME , die bei Anwendung des Algorithmus vom Benutzer mit aktuellen Parametern aufgerufen wird.

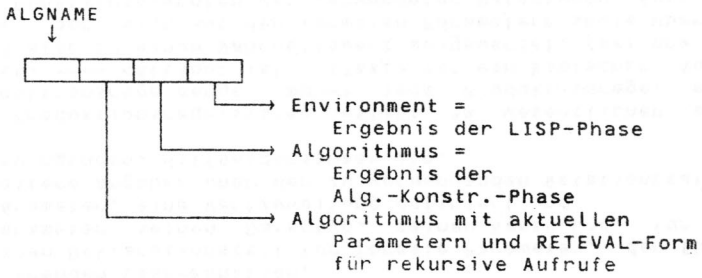
Der fertig implementierte Algorithmus hat folgende LISP-Struktur:



Das Environment ist eine LISP-LAMBDA-Form mit einer Liste der formalen Benutzer-Parameter, die bei Aufruf des Algorithmus gemäß LISP-Konvention durch aktuelle Parameter ersetzt werden, und besteht aus geschachtelten COND-Ausdrücken, die die im <header> der Spezifikation gemachten Angaben über Voraussetzungen, Hilfsalgorithmen, Relation und Parametertypen testen.

Jeder COND-Ausdruck hat nur eine Alternative: entweder ist der Test wahr, dann wird zum nächsten COND-Ausdruck übergegangen, andernfalls wird eine entsprechende Fehlermeldung ausgegeben und der Algorithmus geht in den normalen INTERLISP-Break-Mode über, von dem aus dem Benutzer alle Möglichkeiten zur Fehlersuche und Korrektur offenstehen.

Sind alle COND-Ausdrücke erfolgreich abgearbeitet, werden den System-Parametern mittels SETQ-Formen die spezifizierten Werte zugewiesen.
Es folgt ein RPLACA-Konstrukt, das eine mit aktuellen Werten versehene Kopie des Algorithmus an die Global-Value-Zelle des Atom ALGNAME bindet.
Nach Aufruf des Algorithmus ist die Ausführung der RPLACA-Form die letzte Environment-Anweisung vor der eigentlichen Abarbeitung des Algorithmus. Die Bindungen sind dann wie folgt:



Die als Wert an ALGNAME gebundene Liste unterscheidet sich von der an den Indikator ALGORITHMUS von ALGNAME gebundenen dadurch, daß rekursive Aufrufe des Algorithmus durch eine RETEVAL-Form substituiert worden sind. Mit Hilfe dieses Konstruktes werden in den Kontrollstacks des LISP-Systems gezielt Bereinigungen vorgenommen und letztlich rekursive Aufrufe in eine Folge dieser Stacksprünge überführt.

Durch das RETEVAL-Konstrukt werden somit Rekursionen in Iterationen umgewandelt, wodurch sich noch der folgende Seiteneffekt ergibt: Bei jedem rekursiven Aufruf des Algorithmus würden im Parameter- und Kontrollstack des INTERLISP-Systems neue Stack-Positionen aufgebaut, an die die aktuellen Parameter und die auszuführenden Formen gebunden sind. Da zum Beispiel die Tiefe der Rekursion bei Sortier-Algorithmen von der Anzahl der zu sortierenden Elemente abhängt, kann es in der Praxis vorkommen, daß der (natürlich) endliche Parameter- und Kontrollstack für große Mengen zu sortierender Elemente überlauft, was zu einem Abbruch der Algorithmus-Ausführung, ja sogar zum System-Zusammenbruch führen kann. Bei der iterativen Lösung, die ALGCOMP erzeugt, wird bei jedem rekursiven Aufruf im Parameter- bzw. Kontrollstack durch das RETEVAL-Konstrukt direkt unter den ersten Aufruf des Algorithmus (bzw. das letzte PROG bei parallel-rekursiven rechten Seiten) zurückgesprun-

gen und die Ausführung mit den neuberechneten Parameterwerten von hieraus fortgesetzt, so daß immer nur die obersten Stack-Positionen belegt sind und ein overflow ausgeschlossen ist. Ferner werden auf diese Weise alle überflüssigen Alternativen von COND's aus dem Kontroll-Stack entfernt, während diese bei Verwendung von Rekursionen bis zur endgültigen Terminierung des Algorithmus erhalten bleiben.

Bei der implementierten Lösung wird zwischen zwei Realisierungen dieses Iterationskonzepts unterschieden:

- Es treten rekursive, aber keine parallel-rekursiven rechten Seiten in der Spezifikation auf.
  Dann wird Code erzeugt, der die Stackbereinigungen gemäß Abb.1 (siehe unten) durchführt.

- Mindestens eine parallel-rekursive rechte Seite tritt in der Spezifikation auf.
  Die Auswertung eines derartigen Konstruktes verlangt, daß der unveränderte Satz Parameterwerte für jeden der parallelen Aufrufe des Algorithmus zur Verfügung steht und somit nicht auf gemeinsame Variablenbezeichner zurückgegriffen werden darf. Andernfalls könnte eine vorangehende Abarbeitung bereits die Werte der Variablen ändern.
  Deswegen werden zunächst für jeden der parallelen Aufrufe die individuellen Parametersätze in einen privaten FUNARG-Bereich (siehe dazu [GK 78]) kopiert, und diese Bereiche dann geeignet den einzelnen Aufrufen zur Verfügung gestellt.
  Die Stackbereinigung darf in diesem Fall nur soweit gehen, daß eventuelle noch auszuführende parallele Aufrufe nicht vernichtet werden können. Aus diesem Grund wird nicht der Algorithmusname als Sprungziel definiert, sondern das letzte PROG, und zwar in allen RETEVAL-Formen.
  Die Erzeugung des Codes für diesen Rücksprung aus einer parallel-rekursiven rechten Seite wird bereits in der Algorithmus-Konstruktions-Phase (siehe 4.1.2.2.) vorgenommen und befindet sich bereits in der an den Indikator ALGORITHMUS gehangten Form.

Graphisch läßt sich die Wirkung der Stacksprünge folgendermaßen darstellen:

Abb.1: Für rekursive Aufrufe:

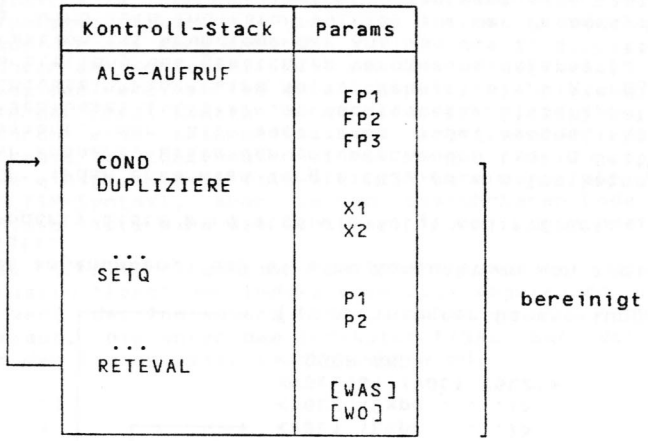
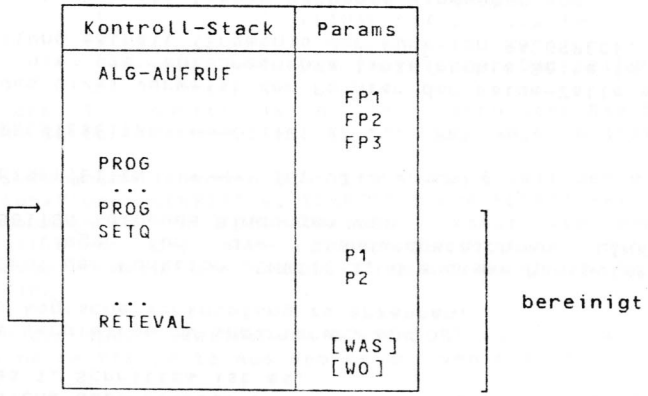


Abb.2: Für parallel-rekursive Aufrufe



Die FPi bezeichnen die formalen Parameter des Algorithmusaufrufs und die Xi bzw. Pi die Parameter der Funktionsaufrufe.
Zur weiteren Information über die Systemstacks des LISP-Systems siehe [GK 78,Kap.12].

4.1.2. Die Phasen des ALGCOMP-Systems

4.1.2.1. Die Einlese-Phase

Die Einlese-Phase des ALGCOMP-System hat die Aufgabe, Spezifikationen abstrakter Datentypen vom Bildschirm oder von einer Datei zu lesen, Übereinstimmung mit der Grammatik der Spezifikationssprache (s.a. 2.4.2.) zu überprüfen, die globalen Variablen URBINDUNGEN, ALGPARAM, RELATION, ALGORITHMUS-WERT, HILFSALGORITHMEN, HILFSFUNKTIONEN (partiell), ALGORITHMUS, PARAM-TYP-LISTE, VORAUSSETZUNGEN, KOMMENTAR, LINKESEITEN, RECHTESEITEN, L1,...,Ln, R1,...Rn und PRA zu definieren und die Spezifikation in interner Darstellung den nachfolgenden Phasen zur weiteren Verarbeitung zur Verfügung zu stellen.

Die Einlese-Phase wird im wesentlichen durch die Funktion RALGSPEC mit Unterfunktionen, insbesondere REGELPARSER, realisiert. Wir verweisen an dieser Stelle auf die ausführlichen Beschreibungen der Funktionen RALGSPEC und REGELPARSER sowie die Erlauterungen zum Parsing-Prozess in 5.2.1.2.

Die oben erwähnte interne Darstellung der Spezifikation unterscheidet sich von der Spezifikationssprache hauptsächlich durch hinzufügen von Klammern und einer LISP-adäquateren Schreibweise: insbesondere werden rechte und linke Seiten von Produktionsregeln geklammert sowie jeder Vergleich, jeder Parameter und jedes Argument eines Funktions- bzw. Datentyp- oder LISP-Funktions-Aufrufs.
Die zusätzliche Klammerung ist notwendig, um den Erkennungsmechanismen des Pattern-Matchers zu genügen.

Nach Durchlaufen der Einlese-Phase ist die Spezifikation soweit systemintern aufbereitet, daß der Pattern-Matcher mit der Konstruktion des Algorithmus beginnen kann.
Werden in der Einlese-Phase Fehler in der Spezifikation erkannt, erfolgt eine entsprechende Fehlermeldung und der Implementierungslauf wird abgebrochen.

4.1.2.2. Die Algorithmus-Konstruktions-Phase

Die Algorithmus-Konstruktions-Phase (AK-Phase) stellt im wesentlichen einen regelgesteuerten Übersetzer dar, der aus dem Produktionsregelsystem, in der von der Einlese-Phase erzeugten internen Form, einen COND-Ausdruck in der Zielsprache INTERLISP erzeugt. Der COND-Ausdruck realisiert die einer Produktionenmenge inharenten logischen Verknüpfungen der {TEST} - konstrukte durch Pradikate und die {AKTION} - Konstrukte durch eine Folge von LISP-Formen. Außerdem wird ein Standard-Fehlerausgang generiert, der den Fall abdeckt, daß keine der spezifizierten Alternativen zutrifft (was einem guten Spezifikationsschreiber natürlich nicht unterläuft...).

Die AK-Phase gliedert sich in sechs Schritte, bei denen je-

weils verschiedene Teile des Produktionssystems nach verschiedenen Zielsetzungen gematcht, manipuliert oder in ausführbaren LISP-Code transformiert werden. Jede linke Seite einer Produktion (d.h. jedes {TEST} - Konstrukt) wird dabei als aus einer Anzahl von Vergleichen der Form

<AUSDRUCK> <RELATION> <AUSDRUCK>

bestehend angenommen, die untereinander entweder durch das Schlüsselwort "UND" oder das Schlüsselwort "ODER" verknüpft sind (siehe dazu 2.4.).
Ziel des 1. Schrittes ist es,

- die Vergleiche zu separieren, und
- die logische Verknüpfung zu erkennen.

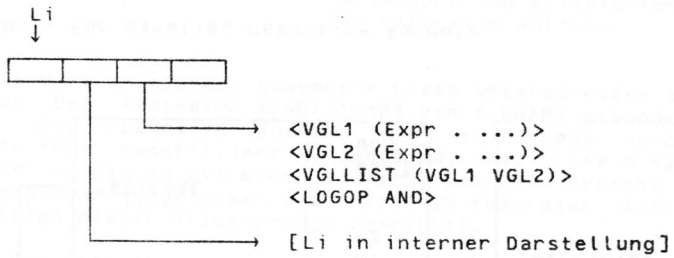
Vor Aufruf der Funktion SCHRITT-1, die diese Manipulation vornimmt, liegen für die Standardbezeichner LINKESEITEN, RECHTESEITEN folgende Bindungen vor:

LINKESEITEN ⟵ (L1 L2 .... Ln)

RECHTESEITEN ⟵ (R1 R2 .... Rn)

Für jedes Li/Ri verweist der Pointer der Value-Zelle auf eine Liste, die die entsprechende linke/rechte Seite in interner Darstellung enthält (Ergebnis der Funktion RALGSPEC).

Nach dem 1. Schritt liegen folgende Bindungen vor:



(Es ist angenommen, daß Li eine Konjunktion von zwei Vergleichen ist).
Die Property-Liste wird also wie folgt modifiziert:

- Für jeden erkannten Vergleich wird ein Indikator VGLj auf der Property-Liste der entsprechenden linken Seite Li initiiert, der als Wert eine Assoziationsliste der Form ((Schlüssel . Liste) ... (Schlüssel . Liste)) hat. Für den Schlussel "Expr" wird im 1. Schritt als Liste die Interndarstellung des gematchten Vergleichs definiert; diese Extraktion ist Grundlage der Analyse des 2. Schrittes.
- Es wird ein Indikator VGLLIST auf der Property-Liste der linken Seite Li initiiert, dessen Wert eine Liste der Indikatoren ist, die den beim Matchen erkannten Vergleichen

zugeordnet sind. Diese Liste dient zur Steuerung des Match-Vorgangs im 2. Schritt, wenn die Vergleiche einzeln bearbeitet werden.

- Es wird ein Indikator LOGOP auf der Property-Liste der linken Seite Li initiiert, dessen Wert die erkannte logische Verknüpfung der Vergleiche ist. Diese Information wird im 3. Schritt aufgegriffen.

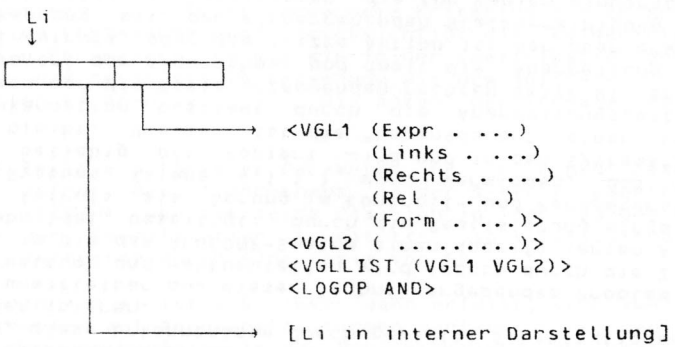
Diese Manipulationen werden für alle Elemente der Liste LINKESEITEN (= für alle Li) durchgeführt.

Der 2. Schritt, gesteuert durch die Funktion SCHRITT-2, leistet zweierlei:

- Zerlegung aller Vergleiche in deren Konstituenten
- Erzeugung von LISP-Code für diese Konstrukte

Jeder Vergleich VGLj jeder linken Seite Li wird syntaktisch geprüft (bzgl. der Grammatik der Spezifikationssprache) und in seine Konstituenten zerlegt. Dabei ist es aus prinzipiellen Gründen notwendig, daß für die Konstituenten in den Regeln bereits feste Muster vorgegeben sind. Nur wenn sich ein vorhandenes Muster ausreichend mit einem zu prüfenden matchen läßt, kann die entsprechende Regel angewandt werden. Im anderen Fall müssen die Regelmuster geeignet erweitert werden, damit syntaktisch korrekt gebildete Ausdrücke der Spezifikationssprache auch über den Pattern-matcher erkannt werden können.

Nach dem 2. Schritt sind folgende Bindungen vorhanden:



Erkannte Konstituenten eines Vergleichs

<AUSDRUCK> <RELATION> <AUSDRUCK>

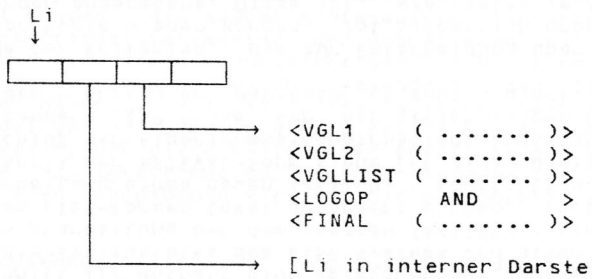
werden der dem Indikator VGLj zugeordneten Assoziationsliste zugefügt:

- unter dem Schlüssel "Links" die linke Seite des Vergleichs

in interner Darstellung, jedoch mit gleichzeitiger Syntaxmodifikation, so daß eine elementare LISP-ausführbare Form entsteht.

- unter dem Schlüssel "Rechts" die rechte Seite des Vergleichs in interner Darstellung, jedoch mit gleichzeitiger Syntaxmodifikation, so daß eine elementare LISP-ausführbare Form entsteht.
- unter dem Schlüssel "Rel" der Name der Relation, die sich aus dem vereinbarten Relationssatz (siehe S. 509) und dem in der Spezifikation verwendeten Symbol ergibt.
- Durch eine zusammenfassende Regel werden die drei extrahierten Komponenten eines Vergleichs zu einer Liste vereinigt, die bereits als LISP-Form aufgefaßt werden kann (LISP-Syntax), aber keinen ausführbaren Code darstellt. Diese Liste wird dem Schlüssel "Form" zugeordnet.

Der 3. Schritt (Funktion SCHRITT-3) komponiert die unter den Schlüsseln "Form" der Indikatoren VGLj abgelegten Listen mit dem Wert des Indikators LOGOP, d.h. es wird eine LISP-Form aufgebaut, die unter dem Indikator FINAL auf der Property-Liste der linken Seite Li abgelegt wird:



Mit dem 3. Schritt ist die mehr technische Bearbeitung der linken Seiten (der {TEST} - Konstrukte) abgeschlossen.

Die sich anschließenden Schritte 4 bis 6 (mit den entsprechenden Funktionen SCHRITT-4, SCHRITT-5 und SCHRITT-6) analysieren die rechten Seiten (oder {AKTION} - Konstrukte) der einzelnen Produktionen. Die dabei auftretenden Probleme sind von gewichtigerer Natur als bisher. Dies sei durch ein Beispiel verdeutlicht:

In der Spezifikation von MERGE-SORT (siehe 4.4.1.) besteht eine rechte Seite aus dem rekursiven Aufruf

MERGE-SORT((A POP),B,(C PUSH (A POP))).

(A,B,C seien formale Parameter von MERGE-SORT.)

Das Auftreten des formalen Parameters A in der ersten Argumentposition beeinflußt in der abstrakten Spezifikation nicht das Auftreten in der dritten Position, d.h. die einzelnen Argumente eines rekursiven Funktionsaufrufs werden

als unabhängig voneinander angenommen.
Ist A jedoch eine bestimmte, implementierte Inkarnation (etwa des Datentyps KELLER), kann das Evaluieren des ersten Arguments den Wert des dritten Arguments beeinflußen, d.h. es wird der bereits gePOPte Keller A zu Berechnung herangezogen, entgegen der Intention.

Eine Möglichkeit, derartige unerwünschte Seiteneffekte zu vermeiden, besteht darin, geeignet Kopien anzufertigen und auf diese anstelle von Inkarnationen zurückzugreifen.
Daher liegt ein Hauptgewicht der Analyse der Schritte 4 und 5 darin, festzustellen, wann wo welche Kopien notwendig sind, und darin, diese Erkenntnisse durch gezielte Substitutionen im Code zu realisieren.
Ein sich unmittelbar aus dieser Technik ergebendes Problem der Kopienverwaltung und -akkumulation wird jedoch durch die Zielstruktur, in die das ALGCOMP-System einen spezifizierten Algorithmus abbildet, beseitigt: durch die Realisierung eines rekursiven Aufrufs als Sprung im Kontroll- und Parameterstack des LISP-Systems (siehe 4.1.1.) wird vermieden, daß eine Struktur beliebig oft kopiert wird und in den Systemstacks resident bleibt. Vielmehr ist die "Lebensdauer" einer Kopie einer Inkarnation bestimmt durch die Abarbeitungszeit der LISP-Formen der die Kopie erzeugenden rechten Seite Ri: entweder terminiert der Algorithmus und damit die Abarbeitung des Top-Level-Aufrufs, oder die letzte Aktion ist der oben erwähnte Stacksprung mit der einhergehenden Stackbereinigung einschließlich aller Stackpositionen, die für Kopien eingerichtet wurden.

Die Entscheidung, wann ein formaler Parameter kopiert werden muß, wird in der Regel auf Grund syntaktischer Beobachtungen beim Pattern-Matching gefällt, da die Semantik der Konstrukte nicht als bekannt vorausgesetzt werden kann. (Zudem haben gleichnamige Operanden, die in verschiedenen Algorithmenspezifikationen auftauchen, nicht notwendig gleiche Semantiken, denn ausschlaggebend ist, für welchen Datentyp sie jeweils spezifiziert wurden). Allerdings gibt es einige Sonderfälle, in denen Programmierwissen des Regelschreibers mit in die Regelmuster eingefloßen ist, und insbesondere das Wissen über seiteneffekt-ungefährliche Konstrukte wird an einigen Stellen zur Entscheidung gegen eine Kopie benutzt.
Bevor die im ALGCOMP-System realisierten Detektionstechniken besprochen werden, sei zum Verständnis noch die Zielstruktur erlautert, in die die {AKTION} - Konstrukte der Spezifikationssprache, wenn sie rekursive Aufrufe des Algorithmus darstellen, abgebildet werden. Ein rekursiver Aufruf der Form

MERGE-SORT((A POP),B,(C PUSH(A POP)))

wird uberfuhrt in eine Folge von LISP-Formen, in der drei Abschnitte unterschieden werden:


```
I.   [ (DUPLIZIERE ...)
     [ .......
     L (DUPLIZIERE ...)
II.  [ (SETQ ...)
     [ ...
     L (SETQ ...)
III. [ (RETEVAL ....)
```

Im Abschnitt I befinden sich die LISP-Formen, die von bestimmten formalen Parametern des Algorithmus Kopien herstellen und an Standardbezeichner binden. Hier sind also die Ergebnisse der "Kopien-Untersuchungen" der Schritte 4 und 5 explizit ablesbar. Erzeugt wird dieser Abschnitt des Codes im sechsten Schritt.
Im Abschnitt II wird jedem formalen Parameter des Algorithmus, dessen Wert durch das betrachtete {AKTION} - Konstrukt geändert wird, über die LISP-Funktion SETQ ein neuer Wert zugewiesen. Innerhalb dieser LISP-Formen sind an bestimmten Stellen formale Parameter durch Standardbezeichner (Xi) substituiert, und verhindern so die oben erwähnten unerwünschten Seiteneffekte, insbesondere, daß bereits mit einem neuen Wert versehene formale Parameter in Wertberechnungen miteinbezogen werden. Dieser Abschnitt wird in den Schritten 4 und 5 erzeugt.
Der Abschnitt III enthält eine RETEVAL-Form, durch deren Ausführung die Systemstacks des LISP-Systems von allen Bindungen, die durch Abarbeitung der dem letzten {AKTION} - Konstrukt zugeordneten LISP-Formen lokal generiert wurden, bereinigt und die Abarbeitung eines neuen {AKTION} - Konstruktes einleitet. (Zur Semantik der RETEVAL-Form siehe [EK 77]). Diese Form wird nach Abschluß der Algorithmus-Konstruktions-Phase in der sich anschließenden LISP-Phase von der gleichnamigen Funktion in den bis dahin erzeugten COND-Ausdruck substituiert.

Die Menge der Kriterien, die zur Entscheidung über die Erzeugung einer Kopie dienen können, ist sicherlich größer als die im folgenden angegebene. Diese ist, wie vieles im APE-System, empirisch entstanden aus der Betrachtung einer steigenden Anzahl von Beispielen und ist ausreichend für die Überdeckung der darin auftretenden Fälle.
Das erste Kriterium lautet:

* Tritt ein formaler Parameter FP des Algorithmus n Mal (n>1) in einer rechten Seite Ri auf, so substituiere das zweite bis n-te Mal durch eine Kopie von FP.

In dieser Rigorosität angewandt, führt das Kriterium * in der Praxis jedoch zu vielen überflüssigen Kopien, so daß es in zwei Richtungen abgeschwacht wird: einmal betrachtet man die Fälle, daß es sich bei FP um eine Inkarnation eines vom ADTCOMP-System implementierten Datentyps handelt:

*' Tritt ein formaler Parameter FP des Algorithmus, der Inkarnation eines ADTCOMP-Datentyps ist, n Mal (n>1) in einer rechten Seite Ri auf, so substituiere das zweite bis n-te Mal durch eine Kopie von FP, sofern uner-

wünschte Seiteneffekte nicht ausgeschlossen werden können.

Der letzte Zusatz zum Kriterium *' berücksichtigt die oben gemachte Anmerkung, daß die Semantik gewisser Konstrukte mit in die Regeln miteingeflossen ist und die Entscheidung über eine Kopie beeinflußt (siehe 4.1.3. und 5.2.2.).

*'' Treten in n Funktionsaufrufen (n>1) innerhalb einer rechten Seite Ri formale Parameter FPj des Algorithmus auf, die nicht Inkarnationen eines ADTCOMP-Datentyps sind, dann substituiere im zweiten bis n-ten Funktionsaufruf jedes Auftreten durch eine Kopie von FPj, sofern unerwünschte Seiteneffekte nicht ausgeschlossen werden können.

Die Implementierung einer Inkarnation eines abstrakten Datentyps geschieht im ADTCOMP-System in Form eines LISP-Atoms mit Funktionszelle ≠ NIL (siehe 3.1.). Insofern werden Inkarnationen als Funktionen interpretiert, und die Unterscheidungen in den Kriterien *' und *'' reflektieren darauf, daß Inkarnationen im Gegensatz zu "anderen" Funktionen gesondert behandelt werden müssen. (Der (Inkarnations-) Bezeichner A ist gleichzeitig Funktions- und Variablenbezeichner; als Funktion läßt er sich daher nur durch Substitution übergeben, als Variable genügen die LISP-Übergabemechanismen.) Außerdem lassen sich semantische Aspekte bei der Entscheidung über eine Kopie bei Konstrukten, die aus ADTCOMP - Inkarnationen aufgebaut sind, leichter miteinbeziehen als bei Konstrukten, die aus Nicht-Inkarnationen aufgebaut sind.

Das zweite, in ALGCOMP verwendete Kriterium zur Entscheidung über eine Kopienerzeugung berücksichtigt die Reihenfolge der formalen Parameter, so wie sie in der Spezifikation angegeben wurde, und setzt sie in Beziehung mit dem Auftreten in einer aktuellen rechten Seite Ri. Die Anzahl der formalen Parameter sei n.

** Tritt ein formaler Parameter FP in einem rekursiven Aufruf des Algorithmus in der s-ten Argumentposition (1≤s≤n) und ist seine Position in der Parameterliste der Spezifikation t mit t<s, so substituiere dieses Auftreten durch eine Kopie von FP.

Das Kriterium ** ist z.B. immer dann erfüllt, wenn zwei formale Parameter auf Argumentpositionen vertauscht werden bezüglich der Reihenfolge in der Parameterliste der Spezifikation und somit immer einer bereits einen neuen Wert zugewiesen bekommen hat, wenn er in die Wertberechnung für den anderen einbezogen wird.
Dieses Kriterium wird im ALGCOMP-System ähnlich modifiziert wie das Kriterium *, also in Fälle des Vorliegens von Inkarnationen und Nicht-Inkarnationen aufgeteilt. Zusätzlicher Grund ist hier, daß bei Zuweisung der Wertes einer Inkarnation A an eine Inkarnation B eine Reihe von Überprüfungen bzgl. der Zulässigkeit durchgeführt werden, die im "Nicht-Inkarnations-

Fall" nicht stattfunden (z.B. darf man einer KELLER-Inkarnation K nicht den Wert einer KELLER-SCHLANGE-Inkarnation KS zuweisen).

Die Kriterien *, *' und *'' überdecken sich teilweise; daher wurde durch die realisierte Kontrollstruktur sichergestellt, daß keine redundanten Doppelkopien erzeugt werden und der generierte Code bzgl. dieser Anforderungen minimal bleibt. Der Hauptgedanke dabei ist, die Kriterien in unterschiedlichen Phasen disjunkt anzuwenden und geeignete Regelanwendungspfade zu definieren (siehe 4.1.3., S. 97).

Mit den drei angegebenen Kriterien arbeitet ALGCOMP zur Zeit und implementiert dabei die dokumentierten Algorithmen. Dabei kommt es in Ausnahmefällen zu Entscheidungen für eine Kopie, die zwar nach den angegebenen Kriterien gerechtfertigt sind, aber bei Kenntnis der Semantik der beteiligten Konstrukte unnötig wären. Die Elemination solcher Fälle ist durch Hinzufügen weiteren Programmierwissens zu den Regeln erreichbar. Aber die damit einhergehende Steigerung der Komplexität und Verringerung der Verständlichkeit einer Regel machen dieses Verfahren fragwürdig. Andererseits könnten sophistische Kriterien gesucht werden, die sowohl Syntaxmuster der Konstrukte wie auch Funktionalitätsmuster der Operationen bei der Entscheidung über eine Kopie berücksichtigen (z.B. kann die Information, daß es sich um einen "lesenden" oder "schreibenden" Operator handelt, aus dem Funktionalitätsmuster abgelesen und verwendet werden). Für die Entwicklung des ALGCOMP - Systems galt jedoch, daß diese Fälle einer überflüssigen Kopie eine so geringe Häufigkeit hatten, daß der Aufwand für eine Behebung in der oben angegebenen Richtung nicht gerechtfertigt gewesen wäre.

Die Kopiendetektion und -erzeugung ist die Hauptaufgabe der Schritte 4 und 5.

Aufgaben des 4. Schrittes, der realisiert wird durch die Funktion SCHRITT-4, sind:

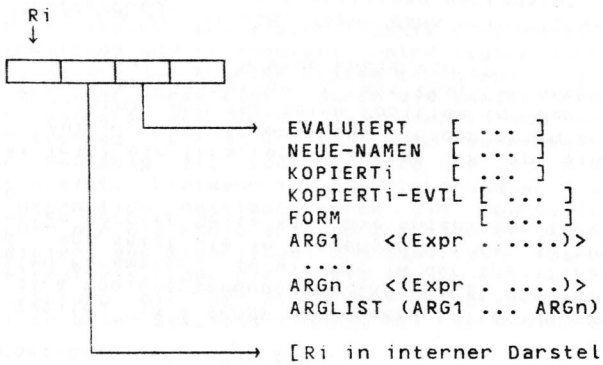
- Feststellung der Struktur einer rechten Seite Ri, d.h. ob es sich um
  - einen rekursiven Aufruf
  - einen parallel-rekursiven Aufruf
  - einen Inkarnationsaufruf
  - einen Funktionsaufruf
  - eine Parameterevaluation

  handelt und der Initiierung der entsprechend zugeordneten Bearbeitung.
- Entscheidung darüber, wo eine Kopie anstelle eines Originals substituiert werden soll.
- Vorentscheidung darüber, wo eine Kopie anstelle eines Originals substituiert werden soll. Die endgültige Entscheidung fällt Schritt 5.
- Zerlegung der rechten Seiten Ri in die sie konstituierenden Bestandteile (Argumente eines rekursiven Aufrufs), oder, falls die Bearbeitung von Ri bereits im vierten

Schritt abgeschlossen wird, die Erzeugung von LISP-Code unter geeigneter Syntaxmodifikation.

Die Funktion SCHRITT-4 arbeitet auf der Liste aller rechten Seiten (R1 .... Rn), die unter dem Standardbezeichner RECHTESEITEN von der Funktion RALGSPEC abgelegt worden ist. Dabei ist der Wert jedes Ri eine Liste, die die entsprechende rechte Seite der Produktion in interner Darstellung enthält (Ergebnis der Funktion RALGSPEC).

Nach Abarbeitung von SCHRITT-4 liegen folgende Bindungen vor, in Abhängigkeit von dem betrachteten Konstrukt:



(Diese Graphik enthält alle, im 4. Schritt auf der Property-Liste von Ri erzeugbaren Indikatoren. Allerdings schließen sich einige Indikatoren wechselseitig aus. Für eine bestimmte Seite Ri ist die Menge ihrer Indikatoren nach dem 4. Schritt eine Teilmenge der oben angegebenen.)

Die Property-Liste des Standardbezeichners Ri wird wie folgt modifiziert:

- Für jede rechte Seite Ri wird auf der Property-Liste der Indikator EVALUIERT initialisiert. Sein Wert (ein literales Atom) gibt Hinweis auf die Struktur von Ri. Mögliche Werte sind:

  Rekursiv: Ri ist ein rekursiver Aufruf des Algorithmus. (In gewissem Sinne der "Normalfall".)
  Parampass: Die rechte Seite besteht nur aus der Auswertung eines formalen Parameters. Diese Seite braucht daher nicht weiter bearbeitet zu werden, und es kann geeigneter LISP-Code erzeugt werden, der unter dem Indikator FORM auf der Property-Liste von Ri abgelegt wird.
  Parallel-rekursiv: Ri besteht aus mehreren, parallel zueinander ausführbaren, rekursiven Aufrufen des Algorithmus. Diese Konstituenten müssen geeignet extrahiert werden, mit neuen, vom System erzeugten Namen versehen werden und einzeln als rekursive Aufrufe bearbeitet werden.

  Die Liste der neuen Namen wird unter dem Indikator NEUE-NAMEN auf der Property-Liste von Ri abgelegt.
  Fun-Arg-Rekursion: Bei Ri handelt es sich um einen rekursiven Aufruf des Algorithmus, der innerhalb einer parallel-rekursiven rechten Seite auftritt. Die Realisierung von Ri wird durch ein LISP-FUNARG-Konstrukt erreicht. Deswegen muß diese rechte Seite durch einen besonderen Satz von Regeln bearbeitet werden, der diese Zielstruktur berücksichtigt.
  Alg-Aufruf: Bei Ri handelt es sich um einen (Hilfs)-Algorithmus-Aufruf, der mit gewissen Syntaxmodifikationen direkt in LISP-Code übersetzt werden kann. Dieser Code wird unter dem Indikator FORM auf der Property-Liste von Ri abgelegt. Eine weitere Bearbeitung ist nicht nötig.
  Lisp-Aufruf: Es handel sich um den Aufruf einer Funktion des LISP-Systems. Siehe "Alg-Aufruf" entsprechend.
  Ink-Aufruf: Es handelt sich um den Aufruf einer Datentyp-Inkarnation (die gleichzeitig formaler Parameter des Algorithmus sein muß).
  Siehe "Alg-Aufruf" entsprechend.

  Die Wertbindungen an EVALUIERT bewirken, daß alle Ri mit EVALUIERT-Bindungen an "rekursiv", "parallel-rekursiv" oder "fun-arg-rekursion" im 5.Schritt weiter bearbeitet werden, alle sonstigen Ri erst im Schritt 6 .

- Für diejenigen Ri, die parallel-rekursive Aufrufe des Algorithmus darstellen, wird auf der Property-Liste der Indikator NEUE-NAMEN initialisiert, dessen Wert eine Liste der vom System erzeugten neuen Namen für die einzelnen rekursiven Aufrufe ist. Dabei erhält jeder rekursive Aufruf innerhalb einer parallel-rekursiven rechten Seite Ri einen eigenen Namen über ein ihm zugeordnetes LISP-Atom (siehe auch die NEUE-NAMEN erzeugende Funktion NEUE-RSEITEN). Diese Liste wird im Schritt 5 verwendet.
- Für diejenigen Ri, bei denen eine Entscheidung für eine Kopie gefällt wird, wird auf der entsprechenden Property-Liste der Indikator KOPIERTj initialisiert, dessen Wert der Name desjenigen formalen Parameters ist, der kopiert werden soll. Der Index j bezeichnet dabei die Argumentposition, in die die Kopie substituiert werden soll. Dieser Indikator wird im 5.Schritt weiterverwendet.
- Für diejenigen Ri, bei denen eine Vor - Entscheidung für eine Kopie gefällt wird, wird auf der entsprechenden Property-Liste der Indikator KOPIERTj-EVTL (= ...-eventuell) initialisiert, dessen Wert der Name des formalen Parameters ist, für den diese Vorentscheidung gelten soll, und j die Argumentposition, in die diese Substitution möglicherweise durchgeführt werden soll. Die endgültige Entscheidung wird im 5.Schritt gefällt, der diese Information dazu aufgreift.
- Für diejenigen Ri, deren EVALUIERT-Indikator als Wert entweder "rekursiv" oder "fun-arg-rekursion" aufweist, wird für jedes erkannte Argument des rekursiven Aufrufs auf der Property-Liste ein Indikator ARGj initialisiert, dessen

Wert eine Assoziationsliste der Form ((Schlüssel . Liste) ... (Schlüssel . Liste)) ist. Für den Schlüssel "Expr" wird im 4. Schritt als Liste die Interndarstellung des gematchten Arguments definiert. Diese Extraktion ist Grundlage der Analyse des 5. Schritts.

- Für diejenigen Ri, deren EVALUIERT-Indikator als Wert entweder "rekursiv" oder "fun-arg-rekursion" aufweist, wird auf der entsprechenden Property-Liste ein Indikator ARGLIST initialisiert, dessen Wert eine Liste der Indikatoren ist, denen beim Matchen erkannte Argumente zugeordnet sind. Diese Liste dient zur Steuerung des Match-Vorgangs im 5. Schritt, wenn die Argumente einzeln bearbeitet werden.
- Für diejenigen Ri, deren EVALUIERT-Indikator einen der Werte "parampass", "alg-aufruf", "lisp-aufruf" oder "ink-aufruf" aufweist, wird auf der entsprechenden Property-Liste ein Indikator FORM initialisiert, dessen Wert eine Liste ist, die den LISP-Code für das erkannte Konstrukt darstellt. Diese Liste wird im 6. Schritt weiterverwendet.

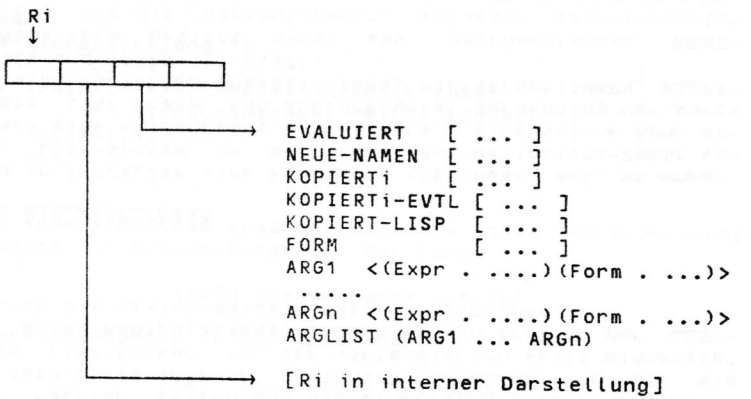
Weitere Modifikationen werden im 4. Schritt nicht durchgeführt.

Im 5. Schritt, realisiert durch die Funktion SCHRITT-5, werden jetzt die Argumente der rekursiven Aufrufe bearbeitet:

- Jedes Argument wird auf syntaktische Zulässigkeit (bzgl. der Grammatik der Spezifikationssprache) untersucht. Dabei ist es, analog dem Match-Vorgang der Vergleiche in Schritt 2, aus prinzipiellen Gründen notwendig, daß für das ganze Argument oder für bestimmte Sub-Strukturen feste Muster in den Regeln vorgegeben sind, wenn erfolgreich gematcht werden soll. Anderfalls muß ein entsprechendes Regelmuster oder eine neue Regel zugefügt werden.
- Wenn bereits feststeht, daß ein formaler Parameter FP in einem vorliegenden Argument ARGj substituiert werden soll, so wird diese Substitution vorgenommen und für FP ein Standardbezeichner Xj eingesetzt.
- Wenn bisher lediglich eine Vorentscheidung bezüglich einer Kopie gefallen ist (Indikator KOPIERTj-EVTL), so wird eine endgültige Entscheidung gefällt. Falls danach kopiert werden muß, wird der entsprechende formale Parameter durch den Standardbezeichner Xj ersetzt.
- Für bestimmte Konstrukte (LISP-Funktionsaufrufe) wird entschieden, ob in den aktuellen Argumenten auftretende formale Parameter des Algorithmus durch Kopien ersetzt werden müssen. Die daraus resultierenden Substitutionen werden sofort durchgeführt.
- Es wird für die Argumente ARGj LISP-ausführbarer Code generiert.
- Es werden Zuweisungen zu den formalen Parametern FPj erzeugt, d.h. (SETQ FPj Liste) - Formen, wobei FPj der j-ten Argumentposition zugeordnete formale Parameter und "Liste" das ARGj zugeordnete LISP-Codestück bezeichnen.
- Es werden Redundanzen aus dem erzeugten Code entfernt.

Um die beiden, in Schritt 5 denkbaren Fälle für den Indikator EVALUIERT, nämlich "Ri EVALUIERT rekursiv" und "RI EVALUIERT fun-arg-rekursion", geeignet zu trennen, werden sowohl durch die Struktur der Regelmenge als auch durch die modifizierte Kontrollstruktur der Funktion SCHRITT-5, für beide Möglichkeiten eigene "Pfade" realisiert.

Nach Ausführung von SCHRITT-5 sind folgende Bindungen (bzw. eine Teilmenge davon) vorhanden:



Die Property-Liste von Ri wird dabei wie folgt manipuliert:

- Unter dem Schlüssel "Form" wird auf der dem Indikator ARGj zugeordneten Assoziationsliste eine Liste abgelegt, die den ARGj entsprechenden LISP-Code enthält. In diesem Codestück sind sämtliche Bereinigungen und Substitutionen durchgeführt.
- Auf der Property-Liste von Ri wird (fallweise) ein Indikator KOPIERT-LISP initialisiert, dessen Wert eine Assoziationsliste von Paaren (Schlüssel . Liste) ist. "Schlüssel" bezeichnet dabei ein Argument ARGj eines rekursiven Aufrufs, und "Liste" ist eine Liste von Paaren (FP K), die ein durch die Kopie K substituiertes Auftreten des formalen Parameters FP bezeichnen. Die so definierte Assoziationsliste wird im sechsten Schritt weiterverwendet.

Da der 5. Schritt im wesentlichen eine Verknüpfung bisher gemachter Beobachtungen beinhaltet, werden keine weiteren Indikatoren erzeugt oder andere als angegeben modifiziert.

Der sechste, die AK-Phase abschließende Schritt, realisiert durch die Funktion SCHRITT-6, leistet folgendes:

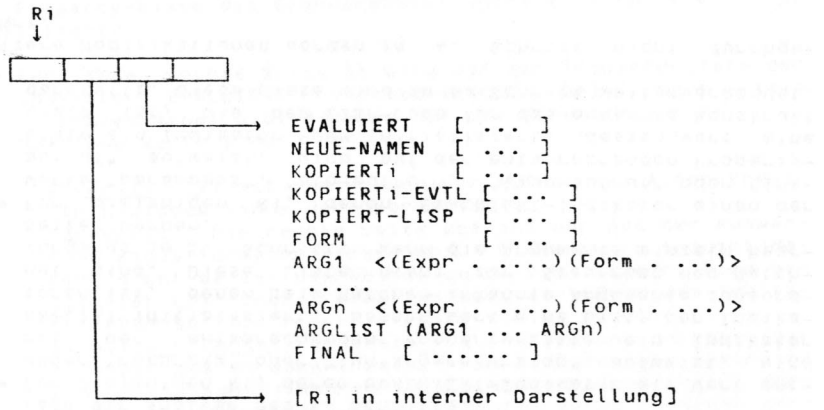
- Es werden für diejenigen Ri, deren EVALUIERT-Indikator den Wert "rekursiv" aufweist, die DUPLIZIERE-Formen für die entsprechenden formalen Parameter erzeugt. Für die Ri mit "Ri EVALUIERT parallel-rekursiv" sind auf Grund der Ziel-

struktur, in die sie abgebildet werden, keine DUPLIZIERE-Formen zu erstellen (siehe Vorbemerkung zu Quick-Sort, Abschnitt 4.4.4.).

- Es werden für diejenigen Ri, deren KOPIERT-LISP-Indikator als Wert eine Assoziationsliste aufweist, DUPLIZIERE-Formen entsprechend den in dieser Liste angegebenen Daten erstellt.
- Der den Ri zugeordnete, LISP-ausführbare Code wird
  - entweder
    a) als der Wert des Indikators FORM aus der zugeordneten Property-Liste extrahiert, falls der EVALUIERT-Indikator einen der Werte "parampass", "alg-aufruf", "lisp-aufruf" oder "ink-aufruf" aufweist,
    b) aus den Assoziationslisten, die den Indikatoren ARGj auf der entsprechenden Property-liste zugeordnet sind, unter dem Schlüssel "Form" extrahiert, falls "Ri EVALUIERT rekursiv" gilt,

    und zusammen mit den bereits erzeugten DUPLIZIERE-Formen (sofern es welche gibt) und einem Zwischen-Code für den rekursiven Aufruf zu einem Codestück verschmolzen. Der Zwischen-Code wird in der sich der AK-Phase anschließenden LISP-Phase durch eine RETEVAL-Form substituiert (siehe Funktionsbeschreibung LISP-PHASE).
  - oder
    mit Hilfe der Funktion PARALLEL-REK-AUFBAU aus der den Indikatoren ARGj zugeordneten Assoziationslisten unter dem Schlüssel "Form" extrahiert und zu einem FUN-ARG-Konstrukt aufgebaut, in dem die parallel ausführbaren rekursiven Aufrufe des Algorithmus durch LISP-PROGs realisiert werden.
- Komposition des COND-Ausdrucks aus allen erzeugten Codestücken.

Nach Ausführung des 6. Schrittes liegen folgende Bindungen (bzw. Teilmengen davon) vor:



Die Property-Liste des Standardbezeichners Ri wird dabei wie

folgt manipuliert:

- Unter dem Indikator FINAL wird der aus den DUPLIZIERE-Formen, der den Argumenten zugeordneten Formen und dem Zwischen-Code für die RETEVAL-Form zusammengesetzte LISP-Code abgelegt. Dieser für Ri generierte Code unterscheidet sich von dem in der lauffähigen Implementierung verwendeten nur noch durch die noch nicht durchgeführte Substitution des RETEVAL-Konstruktes.
- Für die rechten Seiten Ri, für die nicht "Ri EVALUIERT rekursiv" oder "Ri EVALUIERT parallel-rekursiv" gilt, wird als Wert des FINAL-Indikators der Wert des FORM-Indikators definiert.

Andere Indikatoren werden nicht erzeugt oder manipuliert.

Abschließend werden die zusammengehörigen FINAL-Listen für Li und Ri verbunden und in einem COND-Ausdruck zusammen mit einem Standardfehlerausgang komponiert. Dieser COND-Ausdruck wird an den Indikator ALGORITHMUS der Property-Liste des Standardbezeichners ALGNAME übergeben.

Damit ist die AK-Phase beendet.

Es sei noch ausdrücklich darauf hingewiesen, daß der bis hierhin durchlaufende Übersetzungsvorgang in der Regel (d.h. wenn die Eingabespezifikation mindestens eine rekursive rechte Seite aufweist), noch keine brauchbare Abbildung der Spezifikation in LISP darstellt, da der COND-Ausdruck für sich keine Implementierung darstellt. Sehr wesentlich für die Lauffähigkeit sind die Substitution des RETEVAL-Konstruktes in die rekursiven rechten Seiten und die Einbettung dieses so modifizierten Ausdruckes in eine geeignete Umgebung, in der die erzeugten LISP-Formen in der intendierten Weise einsetzbar werden. Diese Manipulationen jedoch werden erst in der LISP-Phase durch die gleichnamige Funktion angestoßen.

4.1.2.3. Die LISP-Phase

Ruft man in INTERLISP eine Funktions <fkt-name> auf, so erwartet das LISP-System an der Funktions-Defintions-Zelle von <fkt-name> eine ausführbare LAMBDA-Form. Eine solche Form erzeugt die LISP-Phase von ALGCOMP unter Verwendung der Werte der globalen Variablen VORAUSSETZUNG, HILFSALGORITHMEN, HILFSFUNKTIONEN und PARAM-TYP-LISTE.
Die LAMBDA-Form besteht dabei aus geschachtelten COND-Ausdrücken, die die verschiedenen, entweder spezifizierten oder während der Algorithmus-Konstruktions-Phase erkannten Bedingungen für eine korrekte Ausführung des Algorithmus überprüfen.

Die einzelnen COND-Ausdrücke für die verschiedenen Bedingungen sehen folgendermaßen aus:

- Benutzer-Parameter werden auf ihren Datentyp hin überprüft; der Datentyp des beim Aufruf des Algorithmus übergebenen aktuellen Parameters bzw. das Ergebnis der evaluierten übergebenen Form muß mit dem in der Spezifikation angegebenen Typ übereinstimmen. Für von ADTCOMP implementierte Datentypen lautet die Form:

```
(COND
 ((EQ (QUOTE <datentyp-name>)
      (GETP <param-name> (QUOTE PASSIVSTRUKTUR)))
  ( ..... ))
 (T (PRINTL "***ERROR***" <param-name> "NICHT VOM TYP"
            <datentyp-name>)
    (BREAK1 NIL T)
    NIL))
```

Für von INTERLISP definierte Datentypen wird auf die systeminterne Typkennung (s.a. [GK 78,p.3-9]) abgefragt, also zum Beispiel für Integer:

```
(COND
 ((OR (TYPEP <param-name> 22)
      (TYPEP <param-name> 23))
  ( ..... ))
 (T (PRINTL "***ERROR***" <param-name> "NICHT VOM TYP"
            INTEGER")
    (BREAK1 NIL T)
    NIL))
```

- Für während der Algorithmus-Konstruktions-Phase oder der Einlese-Phase durch die Funktionen RALGSPEC bzw. FKT-SYMBOL-IN erkannte Hilfsfunktionen wird ihre Definiertheit zum Aufrufzeitpunkt überprüft, wobei allerdings die Funktionen EQ, NEQ, LESSP und GREATERP ausgenommen sind, da sie schon durch das INTERLISP-System zur Verfügung gestellt werden. Es wird folgende Form verwendet:

```
(COND
 ((GETD (QUOTE <hilfsfkt-name>)
  ( ..... ))
 (T (PRINTL "***ERROR*** HILFSFUNKTION"
            <hilfsfkt-name> "UNDEFINIERT")
    (BREAK1 NIL T)
    NIL))
```

- Dasselbe gilt für Hilfsalgorithmen:

```
(COND
 ((GETD (QUOTE <hilfsalg-name>)
  ( ..... ))
 (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
            <hilfsalg-name> "UNDEFINIERT")
    (BREAK1 NIL T)
    NIL))
```


- Voraussetzungen werden in der spezifizierten Form übernommen, wobei Relationszeichen durch entsprechende LISP-Funktionen gemäß dem gewählten Relationssatz ersetzt werden; zum Beispiel geht N<0 über in

```
(LESSP N 0).
```

DER COND-Ausdruck für Voraussetzungen sieht folgendermaßen aus:

```
(COND
 (( <voraussetzung >)
  ( ..... ))
 (T (PRINTL "***ERROR*** VORAUSSETZUNG"
            <voraussetzung> "NICHT ERFUELLT")
    (BREAK1 NIL T)
    NIL))
```

An den innersten COND-Ausdruck schließen sich Wertzuweisungs-Statements für System-Parameter der Form:

```
(SETQ <param-name> <wert>)
```

für Wertzuweisungen an Variable oder

```
(INIT <ink-name> <dt-name>)
```

für Initialisierungen von Datentyp-Inkarnationen (Wertzuweisung an <ink-name>) an.
Es folgt das RPLACA-Statement mit dem RETEVAL-Konstrukt (s.a. 4.1.1.), das nach erfolgreicher Abarbeitung der COND-Ausdrücke und Wertzuweisungen eine mit aktuellen Parameterwerten versehene Kopie des Algorithmus an die Global-Value-Zelle von "ALGNAME" hängt.
Parallel zur RPLACA-Form liegt ein APPLY-Statement, das die soeben als Wert von "ALGNAME" erzeugte Algorithmus-Version mit den aktuellen Parametern zur Ausführung bringt. Bei parallelrekursiven Algorithmen ist dieses APPLY-Statement in ein PROG-Konstrukt eingebettet, da in diesem Fall derjenige Parameter, der Wert der Ausführung einer solchen Aktion sein soll, explizit in der RETURN-Anweisung angegeben werden muß (siehe Quick-Sort,4.4.4.).

Das Environment eines Algorithmus hat folgende Struktur:

```
(<alg-name>
 (LAMBDA (formale Benutzer-params System-params)
  (COND
   ((TEST1)
        .
        .
        .
       (COND
        ((TESTn)
         (Wertzuweisungen an System-params)
         (RPLACA (QUOTE <alg-name>)
           (DSUBST akt-param form-param  .......
            (DSUBST <QUOTE (RETEVAL (STKPOS (QUOTE <alg-name>)
                                            -1)
                                    (QUOTE (APPLY <alg-name>
                                                  NIL>
                    (QUOTE (<alg-name> param-list))
                    (COPY (GETP (QUOTE <alg-name>)
                                (QUOTE ALGORITHMUS))))))
         (APPLY <alg-name> NIL))
        (T (PRINTL "Fehlermeldung")
           (BREAK1 NIL T)
           NIL))
           .
           .
           .
   (T (PRINTL "Fehlermeldung")
      (BREAK1 NIL T)
      NIL))
 )))
```

4.1.3. ALGCOMP-Pattern-Matcher und Regelbasen

Der dem ALGCOMP-System zugrundeliegende Gedanke ist, den Übersetzungsvorgang aus der Spezifikationssprache in die Zielsprache mit Hilfe eines Produktionsregelsystems (PR-System) zu realisieren, das seinerseits von einem Satz von Meta-Regeln gesteuert wird. Im folgenden werden die Systemkomponenten des PR-Systems besprochen.

PR-Systeme unterstützen eine "empirische" Arbeitsweise bei der Entwicklung eines Übersetzungs- und Implementierungssystems in folgendem Sinne:
Bei einem Vorgehen, das mit elementaren Beispielen von Spezifikationen und Implementierungen einsetzt und im Laufe der Zeit komplexere und speziellere Exempel mit in die Betrachtung einbezieht, kann es zu Beginn keine Festlegungen, etwa über die Struktur der Spezifikationssprache oder die Zielstrukturen, in die Worte der Spezifikationssprache abgebildet werden sollen, geben. Denn jedes neue Problem berührt meist an mindestens einer Stelle die Grenzen, die für die bisherigen Aufgaben ausreichend waren; häufig überschreitet es sie. Hat man

dann starre Systemstrukturen, so bleibt die Leistungsfähigkeit auf eine Teilklasse beschränkt oder jedes neue Beispiel erfordert ein partielles oder vollständiges Re-Design. Auf die vorliegende Arbeit angewandt, bedeutete diese Erkenntnis, daß z.B. die Spezifikationssprache für abstrakt spezifizierte Algorithmen nicht auf Grund theoretischer Überlegungen oder Trockenübungen definierbar war, wollte man eine Varietät von möglichen Strukturen nicht von vornherein ausschließen. Vielmehr ergab sie sich aus der Menge der betrachteten Beispiele und konvergierte schließlich derart, daß die Häufigkeit von nicht korrekt implementierten Spezifikationen in dem Maße abnahm, wie die Anzahl der erfolgreich implementierten Spezifikationen stieg.

Das geeignete Hilfsmittel für diese Art der Entwicklung ist ein PR-System. Die syntaktischen Strukturen, die bei der Spezifikation des ersten Beispiels vorliegen, werden zunächst zerlegt und auf (Struktur)-Muster reduziert, die ihrerseits in einzelne Regeln der Regelbasen einfließen. Die meisten der ALGCOMP-Regeln bestehen aus einem derartigen Muster und einer Reihe von Bedingungen und Konsequenzen, durch die ein erfolgreiches Matching definiert und geeignete Transformationen durchgeführt werden. Jede weitere Spezifikation enthält eine Menge von Konstrukten, die entweder eine Teilmenge derjenigen ist, für die in den bereits geschriebenen Regeln schon Muster existieren, oder eine Erweiterung darstellt. Im letzten Fall werden einfach neue Regeln geschrieben, durch die bisher nicht erfolgreich matchbare Konstrukte erkannt und bearbeitet werden können.

Weiter lassen sich in einem gewissen Zustand des Systems viele Spezialregeln zu wenigen generalisieren, wenn man die ihnen zugrundeliegenden Muster vereinigen kann. Das Hinzufügen bzw. Manipulieren von Regeln eines PR-Systems stellt im Gegensatz zu einem partiellen Re-Design eines Systems eine einfachere und praxisnähere Arbeit dar.

Das im ALGCOMP - System eingesetzte PR-System besteht aus drei Komponenten:

- dem Patternmatcher,
- den Regelbasen, und
- dem Kontrollmodul.

Der Patternmatcher wird dabei realisiert durch die bereits für das ADTCOMP-System entwickelte Funktion REGELPRUEF und etwa 20 Unterfunktionen (enthalten in der Datei LISP.DATA.PMREST.nn). Die Verwendung von REGELPRUEF und dem damit verbundenen Pattern-Matching Mechanismus hat hauptsachlich pragmatische Gründe, denn es handelt sich dabei um ein einsatzbereites Pattern-Matching System, das ohne weiteres für die oben beschriebenen Arbeiten der Regelanwendungen im ALGCOMP-System (Muster-Matching) herangezogen werden konnte.

Der Mechanismus der Regelanwendung sei hier nur kurz wiedergegeben (siehe dazu [EKT 79]):

Sämtliche Voraussetzungen einer Regel werden bearbeitet und es wird versucht, sie zu T auszuwerten, d.h. ob Indikatoren von Property-Listen bestimmte Werte aufweisen oder ob Muster gematcht werden können.
Gelingt das, werden alle in der Folgerung der Regel angegebenen Aktionen ausgeführt und eine neue Regel wird aus der Basis ausgewählt.
Andernfalls wird sofort eine neue Regel aus der Basis ausgewählt.

Die Miteinbeziehung der Funktion REGELPRUEF und des damit verbundenen Regelanwendungsmechanismus in das ALGCOMP-System hat vier Konsequenzen:

1) Die Informationshaltung und -manipulation muß ausschließlich über Indikatoren von Property-Listen literaler Atome abgewickelt werden.
2) Der verwendete Muster-Matching-Mechanismus erzwingt starke Anforderungen an die in den Regeln niedergeschriebenen Muster. Ein Matching eines aktuellen Prüflings mit einem Regelmuster ist nur dann erfolgreich in dem Sinne, daß eine Übersetzung eines Spezifikationssprachenkonstrukt in ein Zielsprachenkonstrukt möglich ist, wenn eine "eins-zu-eins" - Zuordnung zwischen Prüfling und Muster besteht bzw. wenn die nicht fest im Regelmuster gebundenen Teile des Prüflings keine Modifikation beim Transfer in die Zielsprache benötigen.
3) Der Kontrollmechanismus für die Regelanwendung muß auf die Grenzen der in einer Regel ausdrückbaren Bedingungen oder Aktionen Rücksicht nehmen und unter Umständen Regelanwendungspfade verzweigen oder wiederholen.
4) Syntax und Semantik der Regeln mußten durch die unveränderte Übernahme von REGELPRUEF ebenfalls unverändert bleiben. Durch das verlagerte Anwendungsgebiet (weniger Verknüpfung von Informationen, mehr Mustererkennung und -manipulation) wurde gelegentlich die Grenze der Expressivität der Regeln erreicht. Abhilfe innerhalb dieses Systems konnte nur durch eine Anzahl von Hilfsfunktionen geschaffen werden, die in Regelvoraussetzungen aufgerufen werden und die erforderlichen Manipulationen gewissermaßen "extern" ausführen.

Besonders der 2. Punkt verringert die einem PR-System innewohnenden Freiheitsgrade bzgl. einer sukzessiven Erweiterung von Fähigkeiten erheblich, da für jedes neue Konstrukt zwar leicht ein entsprechendes Muster neu entworfen werden kann, aber bei der kleinsten Variation des Prüflings (zusätzlicher Parameter einer Operation oder vertauschte Stellung der Terme einer Gleichung) nicht mehr "eins-zu-eins" gematcht werden und die Regel nicht mehr angewendet werden kann. Mit zunehmender Kompliziertheit der Konstrukte gewinnt diese Schwäche an Bedeutung und ist entweder nur durch ein anders strukturiertes Matchingsystem, das etwa einen Prüfling nicht als Ganzes betrachtet, sondern dessen Struktur durch rekursives Zerlegen von Teilen aufspaltet, oder durch starke Verlagerung von Über-

setzungsaufgaben in den Parsingteil des ALGCOMP-Systems zu beheben. Für das vorliegende implementierte System waren allerdings die durch den Transfer von REGELPRUEF erzeugten Randbedingungen nicht von der Größenordnung, daß ein anderes Konzept von Vorteil gewesen wäre.

Die zweite Komponente des ALGCOMP-PR-Systems sind die Regelbasen. Wie im Abschnitt 4.1.2. beschrieben, besteht die Algorithmus-Konstruktionsphase aus sechs Schritten, denen je eine Regelbasis zugeordnet ist. Grund für diese Organisation der Regeln sind die unterschiedlichen Anforderungen an die Regelmuster, die andererseits auch Regeln eines Schrittes (einer Basis) unanwendbar für andere Schritte machen.

Die Standardbezeichner für die Regelbasen heißen entsprechend der Schritte S1-REGELN, S2-REGELN, ... , S6-REGELN. Die Erzeugung, Veränderung oder Vernichtung von Regeln muß "per Hand" vorgenommen werden. Das bereits existierende Regelmanipulationspaket RMP (siehe Dateiverzeichnis im Anhang 3 oder [EKT 79]) ist ausschließlich für Manipulationen von Regeln des ADTCOMP-Sytems konzipiert und nicht kompatibel mit den Anforderungen von ALGCOMP. Ein eigenes Regelmanipulationssystem ist aber auf Grund der Struktur und des derzeitigen Umfangs der ALGCOMP-Regelbasen nicht angemessen.

Eine direkte Folge ist, daß es bei ALGCOMP-Regeln keine Unterscheidung zwischen "externer" und "interner" Darstellung gibt, da die Manipulation der verschiedenen Darstellungen eben eine Aufgabe eines Regelmanipulationspaketes wäre.

Jede einzelne Regel genügt der Syntax, wie sie in [EKT 79] festgelegt wurde. Insbesondere die dort definierten Axiomenvoraussetzungen (= "Muster") sind wesentlicher Bestandteil der meisten ALGCOMP-Regeln, da ein Hauptaugenmerk auf Syntaxerkennung und Modifikation liegt. Zeitweilig war der Umfang der in den sechs Regelbasen vorhandenen Regeln ca. 75 Stück, die von elementarer Struktur und geringer Komplexität waren (wenig Voraussetzungen und Eigenschaftsüberprüfungen). Durch starke Zusammenfassung von Mustern liegt in der implementierten Version ein Destillat von 31 Regeln vor, das zur Bearbeitung der dokumentierten Algorithmen ausreicht. Durch die Verdichtung wird allerdings in viele Regeln eine Komplexität induziert, die die zugrundeliegenden Muster verdeckt und den Anwendungsmechanismus weniger durchschaubar macht. Aus diesem Grund sind in 5.2.2. zu dem dokumentierten Code der einzelnen Regeln noch ausführliche Erläuterungen eingestreut.

Gründe für die Zusammenfassung von Regeln sind neben der Verkleinerung von Regelbasen hauptsächlich die vereinfachte Konstruktion von sogenannten Regelanwendungspfaden (RA-Pfade), wie sie weiter unten beschrieben werden, und die bessere Überprüfungsmöglichkeit, ob ein aktuell vorliegendes Konstrukt bereits von einem Muster einer existierenden Regel gematcht werden kann.

Die dritte Komponente des ALGCOMP-PR-Systems ist der Kontroll-

modul, der im Programmcode auf mehrere Funktionen verteilt ist und im folgenden beschrieben wird.

Aufgaben des Kontrollmoduls sind

- die geeignete Auswahl von Regeln aus der dem aktuellen Schritt zugeordnete Regelbasis,
- die Initiierung des Anwendungsmechanismus für eine ausgewählte Regel (Funktion SCHRITT-i), und
- die Verarbeitung der bei der Regelanwendung gewonnenen Information bei der Auswahl der nächsten Regel (Funktion Si-STEUERUNG).

Die oberste Ebene der Kontrolle stellt die Zuordnung der Regelbasen zu den Schritten der Algorithmus-Konstruktionsphase dar, die durch die Funktionen SCHRITT1, ... ,SCHRITT-6 realisiert wird. Danach folgt die weitere Regelauswahl durch Meta-Regeln, die innerhalb der Funktionen SCHRITT-i und Si-STEUERUNG angewendet werden.

Die Regeln jeder einzelnen Regelbasis sind geordnet. Kriterien für eine Ordnungsrelation auf einer Regelbasis sind die Teilaufgaben, für die die einzelnen Regeln jeweils geschrieben wurden. Zum Beispiel gibt es im 4. Schritt bei der Analyse einer rechten Seite Ri die Teilaufgaben "Erkennung der Struktur", "Erkennung der Notwendigkeit von Kopien" und "Erkennung der Konstituenten". Die diesen Teilaufgaben zugeordneten Regeln befinden sich in der Liste der S4-Regeln in der angedeuteten Reihenfolge. Innerhalb der Menge der Regeln, die einer Teilaufgabe zugeordnet ist, ist eine Ordnung nicht notwendig, kann aber zur besseren Traversierung der Regelmenge definiert werden.

Hauptgrund für die Strukturierung der Regelmenge ist die Tatsache, daß bestimmte Regeln die erfolgreiche Anwendung von Vorgängern voraussetzen. Damit nicht mehrmals die ganze Regelmenge durchprobiert werden muß, ist eine dies berücksichtigende Ordnung von Vorteil.

Weiterhin erlaubt die innere Strukturierung der Regelmengen die Definition von Regelanwendungspfaden (RA-Pfaden), die innerhalb der einem Schritt i zugeordneten Si-STEUERUNG die Auswahl der nachsten anzuwendenden Regel beeinflußen. Dabei soll ein RA-Pfad eine Folge von Teilmengen einer Regelmenge sein, wobei jedes Folgenglied die Menge von Regeln ist, die bezüglich einer erfolgreichen Anwendung der "kleinsten" Regel dieser Menge alle möglichen Nachfolger enthält. Das erste Folgenglied eines RA-Pfades ist die Regelmenge selbst, das letzte die leere Menge.

Beispiel: Die Basis Si-REGELN umfasse 9 Regeln {1,...,9}. Eine der Regeln sei in jedem Fall anwendbar.
Ist 1 erfolgreich, so sei die verbleibende Menge möglicher Regeln {5,...,9} . Ist wieder 5 erfolgreich, so verbleibt {9}.
RA-Pfad: {1,...,9} , {5,...,9} , {9} , {} .

Ist erst 2 erfolgreich, so wird erst die kleinste Regel (1) aus der aktuellen Menge entfernt. Nachfolger von 2 sei {}.
RA-Pfad: {1,...,9} , {2,...,9} , {} .

Die Definition dieser Pfade ist eine wesentliche Voraussetzung für eine effektives Durchlaufen der Regelmenge, indem ein "working-set" nur auf das beschränkt wird, was noch möglich ist.

Die Festlegung der Pfade baut auf der inneren Ordnung der Regelmengen auf und muß explizit vom Regelschreiber durchgeführt werden durch Implementierung in den Code der Steuerfunktionen Si-STEUERUNG. Ein so definiertes System ist allerdings sensitv gegenüber Veränderungen der Regelordnung und erfordert bei Modifikationen der Regelmenge meist ein "Updating" der RA-Pfade, wenn man die Effektivität nicht verlieren will.

Die RA-Pfade werden in den Steuerfunktionen angegeben. Sie sind eines der zwei Kriterien, die bei der Festlegung der nächsten anwendbaren Regel eingesetzt werden.

Das zweite Kriterium reflektiert die Länge des Musters einer Regel und die Länge eines aktuellen Prüflings. Für jede Regel, die eine Axiom-Voraussetzung besitzt, läßt sich eine Zahl ML angeben, die die minimale Anzahl der in ihrem Muster auftretenden literalen Atome bezeichnet. Ein Minimum kann deswegen angegeben werden, weil häufig Teile eines Musters optional gematcht werden können.
Andererseits wird für einen aktuellen Prüfling eine Zahl PL berechnet, die die Anzahl der in ihm vorhandenen literalen Atome bezeichnet.

Gilt PL < ML für eine Regel und einen Prüfling, so wird die Regel nicht angewandt, da der Prüfling schon "syntaktisch zu klein" ist.

Die Information über die minimale Länge des Musters einer Regel wird in der Funktion PATTERNLAENGE verwaltet, auf die die Steuerfunktionen Si-STEUERUNG zugreifen. Wie oben ist die Veränderung oder Berechnung von Daten, die für jede Regel in PATTERNLAENGE aufbewahrt werden, Sache des Regelschreibers. Gerade die Festlegung der Zahl ML ist eine diffizile Angelegenheit, wenn das betrachtete Muster aus einer Folge von vorgeschriebenen und optionalen Teile besteht, und aus allen Varianten eine minimale Zahl berechnet werden muß. Eine korrekte Festlegung dieser Zahl ist deswegen wichtig, weil eine "eigentlich" anwendbare Regel aus diesem Grund abgelehnt werden kann.

Neben diesen beiden Auswahlkriterien wird in den Funktionen Si-STEUERUNG noch eine Information der zugeordneten Schrittfunktion SCHRITT-i über den Ausgang der letzten Regelanwendung (erfolgreich / nicht erfolgreich) benutzt, die über die (SCHRITT-i)-lokale Variable FLAG übergeben wird.

Damit ergibt sich folgender Regelauswahlmechanismus:

[1] FLAG = T, d.h. die letzte Regelanwendung war erfolgreich
⇒ Suche in dem der letzten Regel zugeordneten RA-Pfad die nächste Teilmenge;
Wähle die "kleinste" Regel aus.
Gehe zu 3.

[2] FLAG = NIL, d.h. die letzte Regelanwendung war erfolglos.
⇒ Verringere die aktuelle Regelmenge um die erfolglos versuchte Regel.
Wähle die "kleinste" Regel aus.
Gehe zu 3.

[3] PL < ML, d.h. Prüflinglänge ist kleiner als die minimale Musterlänge der Regel.
⇒ Verringere die aktuelle Regelmenge um diese Regel.
Wähle, sofern möglich, die "kleinste" Regel aus. Gehe zu 5.

[4] PL ≥ ML, d.h. Prüflinglänge ist ausreichend.
⇒ Wende die Regel an!

[5] Ist die aktuelle Regelmenge leer?
ja: Ende des Schrittes.
nein: Gehe zu 3.

Die Kriterien der Regelauswahlsteuerung haben sich im Laufe der Entwicklung des Systems ergeben, als insbesondere bei großen Anzahlen von Regeln in einer Basis die ungesteuerte (sukzessive) Auswahl zu Ineffizienzen führte. Ein Maß für die Effektivität des Systems ist der Regelanwendungsfaktor (RA-Faktor), der der Quotient aus der Anzahl der versuchten Regelanwendungen und der Anzahl der erfolgreichen Regelanwendungen ist. Der RA-Faktor wird für jeden Implementierungslauf des ALGCOMP-Systems berechnet. Mit Hilfe der Steuerung der Regelanwendung konnte eine signifikante Verbesserung erreicht werden, so daß der RA-Faktor jetzt bei durchschnittlich 67.83% liegt.
Im Anhang 2 befinden sich für die hier dokumentierten Algorithmenimplementierungen diesbezügliche Daten.

4.1.4. Beispiel: Implementierung und Benutzung des Algorithmus Bubble-Sort

Der Sortieralgorithmus Bubble-Sort arbeitet auf dem Datentyp OFFENELISTE, da ein direkter Zugriff zu beliebigen Listenelementen notwendig ist.
Bubble-Sort liest jeweils zwei aufeinander folgende Elemente der OFFENENLISTE OL und vergleicht diese mittels der Kleiner-Relation auf den ganzen Zahlen (Relationsatz Standard). Stehen die Elemente bereits in der richtigen Reihenfolge, geht Bubble-Sort ein Elementpaar weiter, sonst werden die beiden Elemente Elemente durch Aufruf des Hilfsalgorithmus CHANGE (siehe 4.5.1.,S.242) vertauscht; in diesem Fall wird die Boolsche Systemvariable BUBBLE? auf den Wert TRUE gesetzt.
Bubble-Sort terminiert, wenn die OFFENELISTE ohne Vertauschen von Elementen durchlaufen wurde.

Zunächst die Spezifikation von Bubble-Sort:

```
ALGORITHMUS-NAME: BUBBLE-SORT;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-INTEGER  OL;
                  SYSTEM:   INTEGER  INDEX := 1,
                            BOOL     BUBBLE? := FALSE;

HILFSALGORITHMEN: CHANGE;

ALGORITHMUS:      BUBBLE-SORT (OL,INDEX,BUBBLE?) =
                {     (OL LIES (ADD1 INDEX)) /= UNDEF   UND
                      (OL LIES INDEX) <= (OL LIES (ADD1 INDEX))
                ---> BUBBLE-SORT (OL,(ADD1 INDEX),BUBBLE?);
                      (OL LIES  (ADD1 INDEX)) /= UNDEF   UND
                      (OL LIES  INDEX) > (OL LIES  (ADD1 INDEX))
                ---> BUBBLE-SORT
                           (CHANGE ('OL,INDEX,(ADD1 INDEX)),
                           (ADD1 INDEX),
                           TRUE);
                      (OL LIES (ADD1 INDEX)) = UNDEF     UND
                      BUBBLE? = TRUE
                ---> BUBBLE-SORT (OL,1,FALSE);
                      (OL LIES (ADD1 INDEX)) = UNDEF     UND
                      BUBBLE? = FALSE
                ---> OL     }

KOMMENTAR:  BUBBLE-SORT LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
            OFFENELISTE OL UND SORTIERT DIESE. WERT VON
            BUBBLE-SORT IST DIE SORTIERTE OFFENELISTE OL,D.H.
            DAS ALTE OL EXISTIERT NICHT MEHR [KNU 73,P.107]!!
```

Da der Hilfsalgorithmus CHANGE auch als abstrakter Algorithmus formuliert ist und mit ALGCOMP implementiert wurde, d.h. als LAMBDA-Ausdruck realisiert ist, muß auf der Aktions-Seite der zweiten Produktionsregel der Parameter OL des CHANGE-Aufrufs gekwotet werden. Auf diese Weise wird der Bezeichner OL übergeben; andernfalls würde der Wert von OL, also eine Listenstruktur, übergeben.

Es folgt das Protokoll der Implementierung:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: BUB
ALGORITHMUS-NAME: (BUBBLE-SORT)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-INTEGER))
                  SYSTEM:   ((INDEX INTEGER 1)
                             (BUBBLE? BOOL FALSE))
HILFSALGORITHMEN: (CHANGE)
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES (ADD1 INDEX)) /= UNDEF) UND
      ((OL LIES INDEX) <= (OL LIES (ADD1 INDEX)))))
R1 : ((BUBBLE-SORT ((OL),((ADD1 INDEX)),(BUBBLE?))))
L2 : ((((OL LIES (ADD1 INDEX)) /= UNDEF) UND
      ((OL LIES INDEX) > (OL LIES (ADD1 INDEX)))))
R2 : ((BUBBLE-SORT ((CHANGE (('OL),(INDEX),((ADD1 INDEX)))),
                    ((ADD1 INDEX)),
                    (TRUE))))
L3 : ((((OL LIES (ADD1 INDEX)) == UNDEF) UND
      (BUBBLE? == TRUE)))
R3 : ((BUBBLE-SORT ((OL),(1),(FALSE))))
L4 : ((((OL LIES (ADD1 INDEX)) == UNDEF) UND
      (BUBBLE? == FALSE)))
R4 : ((OL))
LINKESEITEN:  (L1 L2 L3 L4)
RECHTESEITEN: (R1 R2 R3 R4)
KOMMENTAR:    (BUBBLE-SORT LIEST DIE MIT GANZEN ZAHLEN
               GEFUELLTE LISTE OL UND SORTIERT DIESE.
               WERT VON BUBBLE-SORT IST DIE SORTIERTE
               OFFENELISTE OL , D.H. DAS ALTE OL EXISTIERT
               NICHT MEHR [KNU 73 , P.107])
ALG-EINGABE BEENDET.
RELATION      = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES (ADD1 INDEX))
                                     /= UNDEF))
ERGEBNIS     :      L1 VGL2 (EXPR . ((OL LIES INDEX) <=
                                     (OL LIES (ADD1 INDEX))))
```



```
ERGEBNIS     :      L1 LOGOP (AND)
ERGEBNIS     :      L1 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES (ADD1 INDEX))
                                     /= UNDEF))
ERGEBNIS     :      L2 VGL2 (EXPR . ((OL LIES INDEX) >
                                     (OL LIES (ADD1 INDEX))))
ERGEBNIS     :      L2 LOGOP (AND)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES (ADD1 INDEX))
                                     == UNDEF))
ERGEBNIS     :      L3 VGL2 (EXPR . (BUBBLE? == TRUE))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((OL LIES (ADD1 INDEX))
                                     == UNDEF))
ERGEBNIS     :      L4 VGL2 (EXPR . (BUBBLE? == FALSE))
ERGEBNIS     :      L4 LOGOP (AND)
ERGEBNIS     :      L4 VGLLIST (VGL1 VGL2)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L1 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (NEQ (OL LIES (ADD1 INDEX))
                                          (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L1 VGL2 (RECHTS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L1 VGL2 (REL . <=)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (FORM . (LESSEQP (OL LIES INDEX)
                                              (OL LIES
                                                   (ADD1 INDEX))
                                     ))
```

```
REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L2 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L2 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (NEQ (OL LIES (ADD1 INDEX))
                                         (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L2 VGL2 (RECHTS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L2 VGL2 (REL . >)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (FORM . (GREATERP (OL LIES INDEX)
                                              (OL LIES
                                                  (ADD1 INDEX))
                                      ))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L3 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (EQ (OL LIES (ADD1 INDEX))
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (LINKS . BUBBLE?)
ERGEBNIS     :      L3 VGL2 (RECHTS . T)
ERGEBNIS     :      L3 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (FORM . (EQ BUBBLE? T))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (LINKS . (OL LIES (ADD1 INDEX)))
ERGEBNIS     :      L4 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L4 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (FORM . (EQ (OL LIES (ADD1 INDEX))
```



```
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (LINKS . BUBBLE?)
ERGEBNIS     :      L4 VGL2 (RECHTS .)
ERGEBNIS     :      L4 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (FORM . (EQ BUBBLE?))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((AND (NEQ (OL LIES (ADD1 INDEX))
                                         (QUOTE UNDEF))
                                    (LESSEQP (OL LIES INDEX)
                                             (OL LIES
                                                 (ADD1 INDEX)))
                               ))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((AND (NEQ (OL LIES (ADD1 INDEX))
                                         (QUOTE UNDEF))
                                    (GREATERP (OL LIES INDEX)
                                              (OL LIES
                                                  (ADD1 INDEX)))
                               ))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((AND (EQ (OL LIES (ADD1 INDEX))
                                       (QUOTE UNDEF))
                                   (EQ BUBBLE? T)))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 FINAL ((AND (EQ (OL LIES (ADD1 INDEX))
                                       (QUOTE UNDEF))
                                   (EQ BUBBLE?)))
SCHRITT-3 ENDE
```

```
SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R1

REGELANWENDUNG:  S4-55 AUF R1

REGELANWENDUNG:  S4-9 AUF R1
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . (OL))
ERGEBNIS     :      R1 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :      R1 ARG3 (EXPR . (BUBBLE?))
ERGEBNIS     :      R1 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-52 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (CHANGE (('OL) ,
                                             (INDEX) ,
                                             ((ADD1 INDEX)))))
ERGEBNIS     :      R2 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :      R2 ARG3 (EXPR . (TRUE))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . (1))
ERGEBNIS     :      R3 ARG3 (EXPR . (FALSE))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R4

REGELANWENDUNG:  S4-2 AUF R4
```



```
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FORM (OL)
ERGEBNIS     :      R4 EVALUIERT (PARAMPASS)
SCHRITT-4 ENDE

SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (EXPR . (SETQ BUBBLE? BUBBLE?))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-32 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-33 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-4 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-4 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ OL
                                     (CHANGE 'OL
                                             INDEX
                                             (ADD1 INDEX))
                                     ))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
```

```
REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG3 (EXPR . (SETQ BUBBLE? T))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (EXPR . (SETQ INDEX 1))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG3 (EXPR . (SETQ BUBBLE?))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL ((SETQ INDEX (ADD1 INDEX))
                               (BUBBLE-SORT OL INDEX BUBBLE?))

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 FINAL ((SETQ OL
                                     (CHANGE 'OL
                                             INDEX
                                             (ADD1 INDEX)))
                               (SETQ INDEX (ADD1 INDEX))
                               (SETQ BUBBLE? T)
                               (BUBBLE-SORT OL INDEX BUBBLE?))

REGELANWENDUNG:  S6-1 AUF R3
```



```
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 FINAL ((SETQ INDEX 1)
                               (SETQ BUBBLE?)
                               (BUBBLE-SORT OL INDEX BUBBLE?))

REGELANWENDUNG:  S6-1 AUF R4

REGELANWENDUNG:  S6-21 AUF R4
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 FINAL (OL)


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((<AND (NEQ (OL LIES (ADD1 INDEX))
                  (QUOTE UNDEF))
             (LESSEQP (OL LIES INDEX)
                      (OL LIES (ADD1 INDEX>
        (SETQ INDEX (ADD1 INDEX))
        (BUBBLE-SORT OL INDEX BUBBLE?))
      ((<AND (NEQ (OL LIES (ADD1 INDEX))
                  (QUOTE UNDEF))
             (GREATERP (OL LIES INDEX)
                       (OL LIES (ADD1 INDEX>
        (SETQ OL (CHANGE (QUOTE OL)
                         INDEX
                         (ADD1 INDEX)))
        (SETQ INDEX (ADD1 INDEX))
        (SETQ BUBBLE? T)
        (BUBBLE-SORT OL INDEX BUBBLE?))
      ((AND (EQ (OL LIES (ADD1 INDEX))
                (QUOTE UNDEF))
            (EQ BUBBLE? T))
        (SETQ INDEX 1)
        (SETQ BUBBLE?)
        (BUBBLE-SORT OL INDEX BUBBLE?))
      ((AND (EQ (OL LIES (ADD1 INDEX))
                (QUOTE UNDEF))
            (EQ BUBBLE?))
        OL)
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9
 S5-1 S5-2 S5-4 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.671233

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(NEQ LESSEQP GREATERP EQ)

SCHRITT-6 ENDE
```

```
ALG-KONSTRUKTIONS-PHASE BEENDET

START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<BUBBLE-SORT
 <LAMBDA (OL INDEX BUBBLE?)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((GETD (QUOTE LESSEQP))
       (COND
        ((GETD (QUOTE CHANGE))
          (SETQ INDEX 1)
          (SETQ BUBBLE? NIL)
          <RPLACA (QUOTE BUBBLE-SORT)
            (DSUBST OL (QUOTE OL)
              (DSUBST
                <QUOTE
                  (RETEVAL
                    (STKPOS (QUOTE BUBBLE-SORT)
                            -1)
                    (QUOTE (APPLY BUBBLE-SORT NIL>
                (QUOTE (BUBBLE-SORT OL INDEX BUBBLE?))
                (COPY (GETP (QUOTE BUBBLE-SORT)
                            (QUOTE ALGORITHMUS>
          (APPLY BUBBLE-SORT NIL))
         (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                    (QUOTE CHANGE)
                    "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
      (T (PRINTL "***ERROR*** HILFSFUNKTION"
                 (QUOTE LESSEQP)
                 "UNDEFINIERT")
         (BREAK1 NIL T)
         NIL)))
   (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
              (QUOTE OFFENELISTE-INTEGER))
      (BREAK1 NIL T)
      NIL>>
 LISP-PHASE BEENDET

 (DEFINEQ

 <BUBBLE-SORT
   <LAMBDA (OL INDEX BUBBLE?)
             .
             .
             .
     siehe Ergebnis der LISP-Phase
             .
```



```
             .
             .    >>
)
  <PUTPROPS BUBBLE-SORT KOMMENTAR
            (BUBBLE-SORT LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
             OFFENELISTE OL UND SORTIERT DIESE. WERT VON
             BUBBLE-SORT IST DIE SORTIERTE OFFENELISTE OL ,
             D.H. DAS ALTE OL EXISTIERT NICHT MEHR
             [KNU 73 , P.107])
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                    .
                    .
                    .
     siehe Ergebnis der Automatischen Codierung
                    .
                    .
                    .    >
STOP
```

Zur Benutzung von Bubble-Sort

An Hand des Bubble-Sort-Beispiels soll nun das Arbeiten mit von APE implementierten Datenstrukturen und Algorithmen in INTERLISP-Umgebungen demonstriert werden.
Wir nehmen an, daß ein von ADTCOMP implementierter parametrisierter Datentyp OFFENELISTE-INTEGER (s.a. 3.2.), also die "formale Struktur", im LISP-System existiert. Der Algorithmus Bubble-Sort steht auch zur Verfügung, da wir ihn soeben implementiert haben.
Um mit diesen Objekten arbeiten zu können, müssen noch die Dateien mit den Anwendungsfunktionen geladen werden:
[Schreibweise: Fettgedrucktes gibt der Benutzer ein, darauf folgendes Normalgedrucktes sind Antworten des System.]

(LOAD (QUOTE ADT-EX)) und **(LOAD (QUOTE ALG-EX))**.

Durch Aufruf der Initialisierungsfunktion INIT erzeugen wir jetzt eine OFFENELISTE-INTEGER-Inkarnation, also ein Exemplar dieses Datentyp:

```
(INIT OL-I OFFENELISTE-INTEGER)
"OL-I ALS OFFENELISTE-INTEGER-INKARNATION VERFUEGBAR"
```

Das eben erzeugte Exemplar des Typ Offene-Liste wird jetzt durch Anwenden des Schreiboperators HINZU sukzessive mit den natürlichen Zahlen 503, 87, 512, 61, 908, 170, 897, 275, 170, 426, 154 und 509 gefüllt (Testreihe aus [KNU 73]):

```
(OL-I HINZU 1 503)
((503) 503)

(OL-I HINZU 1 87)
((503) 87 503)

(OL-I HINZU 1 512)
((503) 512 87 503)
```

und so weiter bis

```
(OL-I HINZU 1 509)
((503) 509 154 426 170 275 897 170 908 61 512 87 503)
```

Diese Liste soll jetzt durch Bubble-Sort sortiert werden. Um den Ablauf des Sortierens besser verfolgen zu können, tracen wir Bubble-Sort und den Hilfsalgorithmus Change und erhalten so bei jedem Aufruf der Algorithmen die aktuellen Parameterwerte.
(Bemerkung zum Trace: Bei Change wird im folgenden der COUNT-Parameter weggelassen, da an dieser Stelle nur von Interesse ist, welche Elemente von OL-I vertauscht werden, was durch die Werte der Parameter INDEX1 und INDEX2 angegeben wird. Bubble-Sort läßt sich durch die INTERLISP-Funktion TRACE nicht so modifizieren, daß das hier dokumentierte Bild entsteht, da Bubble-Sort ja nur einmal aufgerufen wird und alle rekusiven Aufrufe durch Stack-Sprünge realisiert werden. Um trotzdem bei jeder "Rekursion" die aktuellen Parameterwerte zu erhalten, haben wir vor dem COND-Ausdruck in der NLAMBDA-Form des Algorithmus ein entsprechendes PRINT-Kommando eingefügt.)

```
(TRACE CHANGE BUBBLE-SORT)
(TIME (BUBBLE-SORT (QUOTE OL-I)))
```

Die TIME-Funktion bewirkt, daß nach Ausführung von Bubble-Sort die hierfür benötigte Zeit und die Anzahl der verwendeten CONS-Zellen ausgedruckt wird.
Es folgt das Ablaufprotokoll von Bubble-Sort:

```
BUBBLE-SORT: OL = OL-I
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((503) 509 154 426 170 275 897 170 908 61 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 1      INDEX2 = 2
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = T
OL-I: ((503) 154 509 426 170 275 897 170 908 61 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 2      INDEX2 = 3
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = T
OL-I: ((503) 154 426 509 170 275 897 170 908 61 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 3      INDEX2 = 4
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = T
OL-I: ((503) 154 426 170 509 275 897 170 908 61 512 87 503)
```



```
      CHANGE: OL = OL-I      INDEX1 = 4      INDEX2 = 5
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 897 170 908 61 512 87 503)
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 897 170 908 61 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 6      INDEX2 = 7
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 170 897 908 61 512 87 503)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 170 897 908 61 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 8      INDEX2 = 9
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 170 897 61 908 512 87 503)
      CHANGE: OL = OL-I      INDEX1 = 9      INDEX2 = 10
BUBBLE-SORT: INDEX1 = 10      INDEX2 = 11      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 170 897 61 512 908 87 503)
      CHANGE: OL = OL-I      INDEX1 = 10      INDEX2 = 11
BUBBLE-SORT: INDEX1 = 11      INDEX2 = 12      BUBBLE? = T
OL-I: ((503) 154 426 170 275 509 170 897 61 512 87 908 503)
      CHANGE: OL = OL-I      INDEX1 = 11      INDEX2 = 12
BUBBLE-SORT: INDEX1 = 12      INDEX2 = 13      BUBBLE? = T
OL-I: ((908) 154 426 170 275 509 170 897 61 512 87 503 908)
```

Jetzt ist OL-I einmal bis zum Ende durchlaufen. Da BUBBLE?=T gilt, schließt sich nach der dritten Produktionsregel (siehe Spezifikation) ein neuer Durchlauf an:

```
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((908) 154 426 170 275 509 170 897 61 512 87 503 908)
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = NIL
OL-I: ((908) 154 426 170 275 509 170 897 61 512 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 2      INDEX2 = 3
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = T
OL-I: ((908) 154 170 426 275 509 170 897 61 512 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 3      INDEX2 = 4
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = T
OL-I: ((908) 154 170 275 426 509 170 897 61 512 87 503 908)
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = T
OL-I: ((908) 154 170 275 426 509 170 897 61 512 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 5      INDEX2 = 6
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 897 61 512 87 503 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 897 61 512 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 7      INDEX2 = 8
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 61 897 512 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 8      INDEX2 = 9
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 61 512 897 87 503 908)
      CHANGE: OL = OL-I      INDEX1 = 9      INDEX2 = 10
BUBBLE-SORT: INDEX1 = 10      INDEX2 = 11      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 61 512 87 897 503 908)
      CHANGE: OL = OL-I      INDEX1 = 10      INDEX2 = 11
BUBBLE-SORT: INDEX1 = 11      INDEX2 = 12      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
```

```
BUBBLE-SORT: INDEX1 = 12      INDEX2 = 13      BUBBLE? = T
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
```

Hier ist der zweite Durchlauf beendet. Es gilt wieder BUBBLE?=T, also folgt der nächste Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1       INDEX2 = 2       BUBBLE? = NIL
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
BUBBLE-SORT: INDEX1 = 2       INDEX2 = 3       BUBBLE? = NIL
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
BUBBLE-SORT: INDEX1 = 3       INDEX2 = 4       BUBBLE? = NIL
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
BUBBLE-SORT: INDEX1 = 4       INDEX2 = 5       BUBBLE? = NIL
OL-I: ((908) 154 170 275 426 170 509 61 512 87 503 897 908)
      CHANGE: OL = OL-I       INDEX1 = 4       INDEX2 = 5
BUBBLE-SORT: INDEX1 = 5       INDEX2 = 6       BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 509 61 512 87 503 897 908)
BUBBLE-SORT: INDEX1 = 6       INDEX2 = 7       BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 509 61 512 87 503 897 908)
      CHANGE: OL = OL-I       INDEX1 = 6       INDEX2 = 7
BUBBLE-SORT: INDEX1 = 7       INDEX2 = 8       BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 512 87 503 897 908)
BUBBLE-SORT: INDEX1 = 8       INDEX2 = 9       BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 512 87 503 897 908)
      CHANGE: OL = OL-I       INDEX1 = 8       INDEX2 = 9
BUBBLE-SORT: INDEX1 = 9       INDEX2 = 10      BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 87 512 503 897 908)
      CHANGE: OL = OL-I       INDEX1 = 9       INDEX2 = 10
BUBBLE-SORT: INDEX1 = 10      INDEX2 = 11      BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 11      INDEX2 = 12      BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 12      INDEX2 = 13      BUBBLE? = T
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
```

Es folgt ein neuer Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1       INDEX2 = 2       BUBBLE? = NIL
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 2       INDEX2 = 3       BUBBLE? = NIL
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 3       INDEX2 = 4       BUBBLE? = NIL
OL-I: ((908) 154 170 275 170 426 61 509 87 503 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 3       INDEX2 = 4
BUBBLE-SORT: INDEX1 = 4       INDEX2 = 5       BUBBLE? = T
OL-I: ((908) 154 170 170 275 426 61 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 5       INDEX2 = 6       BUBBLE? = T
OL-I: ((908) 154 170 170 275 426 61 509 87 503 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 5       INDEX2 = 6
BUBBLE-SORT: INDEX1 = 6       INDEX2 = 7       BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 509 87 503 512 897 908)
BUBBLE-SORT: INDEX1 = 7       INDEX2 = 8       BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 509 87 503 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 7       INDEX2 = 8
BUBBLE-SORT: INDEX1 = 8       INDEX2 = 9       BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 87 509 503 512 897 908)
```



```
      CHANGE: OL = OL-I       INDEX1 = 8       INDEX2 = 9
BUBBLE-SORT: INDEX1 = 9       INDEX2 = 10      BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10      INDEX2 = 11      BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11      INDEX2 = 12      BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12      INDEX2 = 13      BUBBLE? = T
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
```

Es folgt der fünfte Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1       INDEX2 = 2       BUBBLE? = NIL
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 2       INDEX2 = 3       BUBBLE? = NIL
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 3       INDEX2 = 4       BUBBLE? = NIL
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 4       INDEX2 = 5       BUBBLE? = NIL
OL-I: ((908) 154 170 170 275 61 426 87 503 509 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 4       INDEX2 = 5

BUBBLE-SORT: INDEX1 = 5       INDEX2 = 6       BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 426 87 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 6       INDEX2 = 7       BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 426 87 503 509 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 6       INDEX2 = 7
BUBBLE-SORT: INDEX1 = 7       INDEX2 = 8       BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8       INDEX2 = 9       BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9       INDEX2 = 10      BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10      INDEX2 = 11      BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11      INDEX2 = 12      BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12      INDEX2 = 13      BUBBLE? = T
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
```

Es schließt sich der nächste Durchlauf an, da BUBBLE? immer noch den Wert T hat:

```
BUBBLE-SORT: INDEX1 = 1       INDEX2 = 2       BUBBLE? = NIL
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 2       INDEX2 = 3       BUBBLE? = NIL
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 3       INDEX2 = 4       BUBBLE? = NIL
OL-I: ((908) 154 170 170 61 275 87 426 503 509 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 3       INDEX2 = 4
BUBBLE-SORT: INDEX1 = 4       INDEX2 = 5       BUBBLE? = T
OL-I: ((908) 154 170 61 170 275 87 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 5       INDEX2 = 6       BUBBLE? = T
OL-I: ((908) 154 170 61 170 275 87 426 503 509 512 897 908)
      CHANGE: OL = OL-I       INDEX1 = 5       INDEX2 = 6
BUBBLE-SORT: INDEX1 = 6       INDEX2 = 7       BUBBLE? = T
```

```
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10     BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10     INDEX2 = 11     BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11     INDEX2 = 12     BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12     INDEX2 = 13     BUBBLE? = T
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
```

Es folgt der siebente Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = NIL
OL-I: ((908) 154 170 61 170 87 275 426 503 509 512 897 908)
      CHANGE: OL = OL-I      INDEX1 = 2      INDEX2 = 3
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = T
OL-I: ((908) 154 61 170 170 87 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = T
OL-I: ((908) 154 61 170 170 87 275 426 503 509 512 897 908)
      CHANGE: OL = OL-I      INDEX1 = 4      INDEX2 = 5
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10     BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10     INDEX2 = 11     BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11     INDEX2 = 12     BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12     INDEX2 = 13     BUBBLE? = T
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
```

Es folgt der achte Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((908) 154 61 170 87 170 275 426 503 509 512 897 908)
      CHANGE: OL = OL-I      INDEX1 = 1      INDEX2 = 2
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = T
OL-I: ((908) 61 154 170 87 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = T
OL-I: ((908) 61 154 170 87 170 275 426 503 509 512 897 908)
      CHANGE: OL = OL-I      INDEX1 = 3      INDEX2 = 4
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = T
```



```
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10     BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10     INDEX2 = 11     BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11     INDEX2 = 12     BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12     INDEX2 = 13     BUBBLE? = T
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
```

Es folgt der neunte Durchlauf:

```
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = NIL
OL-I: ((908) 61 154 87 170 170 275 426 503 509 512 897 908)
      CHANGE: OL = OL-I      INDEX1 = 2      INDEX2 = 3
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10     BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10     INDEX2 = 11     BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11     INDEX2 = 12     BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12     INDEX2 = 13     BUBBLE? = T
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
```

OL-I ist jetzt zwar sortiert, aber es wurden im letzten Durchlauf Elemente vertauscht (BUBBLE?=T). Folglich schließt sich noch ein letzter Durchlauf an:

```
BUBBLE-SORT: INDEX1 = 1      INDEX2 = 2      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 2      INDEX2 = 3      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 3      INDEX2 = 4      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 4      INDEX2 = 5      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 5      INDEX2 = 6      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 6      INDEX2 = 7      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 7      INDEX2 = 8      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 8      INDEX2 = 9      BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 9      INDEX2 = 10     BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 10     INDEX2 = 11     BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 11     INDEX2 = 12     BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
BUBBLE-SORT: INDEX1 = 12     INDEX2 = 13     BUBBLE? = NIL
OL-I: ((908) 61 87 154 170 170 275 426 503 509 512 897 908)
```

Da BUBBLE? jetzt den Wert NIL hat, terminiert Bubble-Sort nach der vierten Produktionsregel (siehe Spezifikation).

```
19740 CONSES
15.525000 SECONDS
```

Der relativ hohe Zeitverbrauch ist in dem Sortierverfahren selbst begründet: schließlich braucht Bubble-Sort immerhin zehn Durchläufe, um OL-I zu sortieren; andere Sortieralgorithmen sind hier möglicherweise wesentlich schneller. Durch Compilieren des Funktionscodes kann die Ausführungszeit drastisch gesenkt werden; in günstigen Fällen etwa um den Faktor 10.

Ferner sei noch bemerkt, daß die ursprüngliche Liste OL-I nach Ausführung von Bubble-Sort nicht mehr existiert, da der Algorithmus direkt auf OL-I arbeitet; es wird also kein zusätzlicher Speicherplatz benötigt. Will man jedoch die ursprüngliche Liste weiterhin zur Verfügung haben, so muß auf einer Kopie von OL-I gearbeitet werden, etwa durch Aufruf der Formen:

```
(DUPLIZIERE OL-I OL-I1)
(BUBBLE-SORT 'OL-I1)
```

Der Aufruf der INTERLISP-Funktion COPY führt nicht zu dem gewünschten Ergebnis, da COPY nur den Global-Value, nicht jedoch die Property-List und die Funktionszelle kopiert.

4.2. Numerische Algorithmen

In diesem Abschnitt werden die ALGCOMP-Implementierungen einiger numerischer Standardalgorithmen besprochen und zwar

in 4.2.1. die FAKultätsfunktion (FAK),

in 4.2.2. der Größte Gemeinsame Teiler (GGT) und

in 4.2.3. das Kleinste Gemeinsame Vielfache (KGV),

wobei, wie bei allen folgenden Implementierungsbeschreibungen, jeweils die Spezifikation des Algorithmus, das von ALGCOMP automatisch erzeugte Implementierungsprotokoll sowie die LISP-Formen des fertig implementierten Algorithmus kommentiert angegeben werden.
Außerdem wird auf bei der Implementierung auftretende Besonderheiten hingewiesen.

4.2.1. Die Fakultätsfunktion (FAK)

Dieser sehr einfache Algorithmus mit nur zwei Produktionsregeln und nur einer einfach-rekursiven rechten Seite realisiert die bekannte Fakultätsfunktion. Der Benutzer muß beim Aufruf des implementierten Algorithmus den Parameter N angeben, während der Systemparameter Speicher intern initialisiert wird; er dient als Ablage für die Zwischenergebnisse.

Spezifikation der Fakultätsfunktion FAK:

```
ALGORITHMUS-NAME: FAK;

PARAMETERTYPEN:   BENUTZER: INTEGER N;
                  SYSTEM:   INTEGER SPEICHER := 1;

VORAUSSETZUNGEN:  N > 0;

ALGORITHMUS:      FAK (N,SPEICHER) =
                  { N = 0   ---->   SPEICHER;
                    N > 0   ---->   FAK ((SUB1 N),
                                         (ITIMES N SPEICHER))  }

KOMMENTAR:        FAKULTAETS-FUNKTION!!
```

Es folgt das Implementierungsprotokoll; zunächst die Einlese-Phase:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: FAK
ALGORITHMUS-NAME: (FAK)
PARAMETERTYPEN:    BENUTZER: ((N . INTEGER))
                   SYSTEM:   ((SPEICHER INTEGER 1))
VORAUSSETZUNGEN:   ((GROESSERP N 0))
ALGORITHMUS:       PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : (((N == 0)))
R1 : ((SPEICHER))
L2 : (((N > 0)))
R2 : ((FAK (((SUB1 N)) , ((ITIMES N SPEICHER)))))
LINKESEITEN:  (L1 L2)
RECHTESEITEN: (R1 R2)
KOMMENTAR:         (FAKULTAETS-FUNKTION)
ALG-EINGABE BEENDET.
RELATION      = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET
```

Bemerkung: Da in der Spezifikation das Schlüsselwort RELATION nicht auftritt, setzt die Funktion RALGSPEC den Standard-Relationssatz ein.
Es folgen die sechs Schritte der Algorithmus-Konstruktion-Phase:

```
START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (N == 0))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . (N > 0))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . N)
ERGEBNIS     :      L1 VGL1 (RECHTS . 0)
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
```



```
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ N 0))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . N)
ERGEBNIS     :      L2 VGL1 (RECHTS . 0)
ERGEBNIS     :      L2 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (GREATERP N 0))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ N 0))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((GREATERP N 0))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (SPEICHER)
ERGEBNIS     :      R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . ((SUB1 N)))
ERGEBNIS     :      R2 ARG2 (EXPR . ((ITIMES N SPEICHER)))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2)
SCHRITT-4 ENDE
```

```
SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (LISP-AUFRUF SUB1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM . (SETQ N (SUB1 N)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (LISP-AUFRUF ITIMES)
ERGEBNIS     :      R2 ARG2 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM . (SETQ SPEICHER
                                     (ITIMES STD1-2
                                             SPEICHER)))
ERGEBNIS     :      R2 KOPIERT-LISP (ARG2 (N STD1-2)
                                     (SPEICHER SPEICHER))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL (SPEICHER)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE N (STD1-2))
                              (SETQ N (SUB1 N))
                              (SETQ SPEICHER
                                    (ITIMES STD1-2 SPEICHER))
                              (FAK N SPEICHER))


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ N 0)
        SPEICHER)
      ((GREATERP N 0)
        (DUPLIZIERE N (STD1-2))
        (SETQ N (SUB1 N))
```



```
        (SETQ SPEICHER (ITIMES STD1-2 SPEICHER))
        (FAK N SPEICHER))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-2 S5-5 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.680000

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(GROESSERP EQ GREATERP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET
```

Aus der Liste der von RALGSPEC und den in der Algorithmus-Konstruktions-Phase erkannten Hilfsfunktionen werden die Funktionen GREATERP und EQ wieder gestrichen, da diese im INTERLISP-System sowieso definiert sind und somit nicht besonders überprüft werden müssen.
Die anschließende LISP-Phase liefert die Funktionsdefinition von FAK:

```
START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<FAK
 <LAMBDA (N SPEICHER)
  (COND
   ((OR (TYPEP N 22)
        (TYPEP N 23))
     (COND
      ((GETD (QUOTE DUPLIZIERE))
       (COND
        ((GETD (QUOTE GROESSERP))
          (COND
           ((GROESSERP N 0)
             (SETQ SPEICHER 1)
             <RPLACA (QUOTE FAK)
                     (DSUBST
                       <QUOTE (RETEVAL (STKPOS (QUOTE FAK)
                                               -1)
                                       (QUOTE (APPLY FAK NIL>
                       (QUOTE (FAK N SPEICHER))
                       (COPY (GETP (QUOTE FAK)
                                   (QUOTE ALGORITHMUS>
             (APPLY FAK NIL))
           (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                      (QUOTE (GROESSERP N 0))
                      "NICHT ERFUELLT")
              (BREAK1 NIL T)
              NIL)))
        (T (PRINTL "***ERROR*** HILFSFUNKTION"
                   (QUOTE GROESSERP)
```

```
                    "UNDEFINIERT")
              (BREAK1 NIL T)
              NIL)))
       (T (PRINTL "***ERROR*** HILFSFUNKTION"
                  (QUOTE DUPLIZIERE)
                  "UNDEFINIERT")
          (BREAK1 NIL T)
          NIL)))
   (T (PRINTL "***ERROR***" N "NICHT VOM TYP INTEGER")
      (BREAK1 NIL T)
      NIL>>
ISP-PHASE BEENDET
```

ach dem Implementierungslauf existiert das Atom FAK im LISP-ystem; das Ergebnis der LISP-Phase wird an die Funktionszelle on FAK, das Ergebnis der Algorithmus-Konstruktions-Phase an ie Property-List unter dem Indikator ALGORITHMUS gebunden. ußerdem werden unter den Indikatoren KOMMENTAR sowie RELATION ie entsprechenden aktuellen Werte abgelegt.
ls Ergebnis der Algorithmen-Implementierung ergibt sich also olgendes Bild:

```
(DEFINEQ

<FAK
  <LAMBDA (N SPEICHER)
          .
          .
          .
    siehe Ergebnis LISP-Phase
          .
          .
          .    >>
       NIL>>
)
  <PUTPROPS FAK KOMMENTAR (FAKULTAETS-FUNKTION)
          RELATION
          (STANDARD ((== . EQ)
                     (/= . NEQ)
                     (%< . LESSP)
                     (%> . GREATERP)
                     (%<= . LESSEQP)
                     (%>= . GREATEREQP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
              .
              .
              .
    siehe Ergebnis der Automatischen Codierung
              .
              .
              .    >
STOP
```


4.2.2. Größter Gemeinsamer Teiler (GGT)

Dieser Algorithmus berechnet nach bekannter Methode den größten gemeinsamen Teiler von zwei natürlichen Zahlen. Der Datentyp, auf dem der Algorithmus arbeitet, sind also die ganzen Zahlen, die durch die Voraussetzungen auf die positiven ganzen Zahlen eingeschränkt werden.
Zunächst die Spezifikation von GGT:

```
ALGORITHMUS-NAME:  GGT;

PARAMETERTYPEN:    BENUTZER: INTEGER N,
                             INTEGER M;

VORAUSSETZUNGEN:   N > 0    ,   M > 0;

ALGORITHMUS:       GGT (N,M) =
                 {   N = M   ----->   N;
                     N > M   ----->   GGT((IDIFFERENCE N M),M);
                     N < M   ----->   GGT(N,(IDIFFERENCE M N))  }

KOMMENTAR:         GROESSTER GEMEINSAMER TEILER!!
```

Die Spezifikation wird wie folgt implementiert:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: GGT
ALGORITHMUS-NAME:  (GGT)
PARAMETERTYPEN:    BENUTZER: ((N . INTEGER) (M . INTEGER))
VORAUSSETZUNGEN:   ((GROESSERP N 0) (GROESSERP M 0))
ALGORITHMUS:       PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : (((N == M)))
R1 : ((N))
L2 : (((N > M)))
R2 : ((GGT (((IDIFFERENCE N M)) , (M))))
L3 : (((N < M)))
R3 : ((GGT ((N) , ((IDIFFERENCE M N)))))
LINKESEITEN:   (L1 L2 L3)
RECHTESEITEN:  (R1 R2 R3)
KOMMENTAR:          (GROESSTER GEMEINSAMER TEILER)
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET
```

```
START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (N == M))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . (N > M))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . (N < M))
ERGEBNIS     :      L3 LOGOP (NONE)
ERGEBNIS     :      L3 VGLLIST (VGL1)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . N)
ERGEBNIS     :      L1 VGL1 (RECHTS . M)
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ N M))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . N)
ERGEBNIS     :      L2 VGL1 (RECHTS . M)
ERGEBNIS     :      L2 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (GREATERP N M))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . N)
ERGEBNIS     :      L3 VGL1 (RECHTS . M)
```



```
ERGEBNIS     :      L3 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (LESSP N M))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ N M))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((GREATERP N M))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((LESSP N M))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (N)
ERGEBNIS     :      R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . ((IDIFFERENCE N M)))
ERGEBNIS     :      R2 ARG2 (EXPR . (M))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (N))
ERGEBNIS     :      R3 ARG2 (EXPR . ((IDIFFERENCE M N)))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2)
```

```
SCHRITT-4 ENDE

SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (LISP-AUFRUF IDIFFERENCE)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM . (SETQ N (IDIFFERENCE N M)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (EXPR . (SETQ M M))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ N N))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (LISP-AUFRUF IDIFFERENCE)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM . (SETQ M (IDIFFERENCE M N)))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL (N)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((SETQ N (IDIFFERENCE N M))
                              (GGT N M))
```



```
REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ M (IDIFFERENCE M N))
                              (GGT N M))


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ N M)
        N)
      ((GREATERP N M)
        (SETQ N (IDIFFERENCE N M))
        (GGT N M))
      ((LESSP N M)
        (SETQ M (IDIFFERENCE M N))
        (GGT N M))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.756757

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(GROESSERP EQ GREATERP LESSP)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<GGT
 <LAMBDA (N M)
  (COND
   ((OR (TYPEP N 22)
        (TYPEP N 23))
     (COND
      ((OR (TYPEP M 22)
           (TYPEP M 23))
        (COND
         ((GETD (QUOTE GROESSERP))
           (COND
            ((GROESSERP M 0)
              (COND
               ((GROESSERP N 0)
                 <RPLACA (QUOTE GGT)
                         (DSUBST
                           <QUOTE (RETEVAL
```

```
                              (STKPOS (QUOTE GGT)
                                      -1)
                              (QUOTE (APPLY GGT NIL>
                       (QUOTE (GGT N M))
                       (COPY (GETP (QUOTE GGT)
                                   (QUOTE ALGORITHMUS>
              (APPLY GGT NIL))
             (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                        (QUOTE (GROESSERP N 0))
                        "NICHT ERFUELLT")
                (BREAK1 NIL T)
                NIL)))
          (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                     (QUOTE (GROESSERP M 0))
                     "NICHT ERFUELLT")
             (BREAK1 NIL T)
             NIL)))
          (T (PRINTL "***ERROR*** HILFSFUNKTION"
                     (QUOTE GROESSERP)
                     "UNDEFINIERT")
             (BREAK1 NIL T)
             NIL)))
       (T (PRINTL "***ERROR***" M "NICHT VOM TYP INTEGER")
          (BREAK1 NIL T)
          NIL)))
    (T (PRINTL "***ERROR***" N "NICHT VOM TYP INTEGER")
       (BREAK1 NIL T)
       NIL>>
LISP-PHASE BEENDET
```

Durch RALGSPEC wurde, wegen fehlender Angaben in der Spezifikation, wieder der Standardrelationssatz für GGT definiert. Aus der Liste der erkannten Hilfsfunktionen wurden die in INTERLISP zur Verfügung stehenden EQ, GREATERP und LESSP gelöscht.

Das Atom GGT besitzt nach der Implementierung folgende Bindungen:

```
(DEFINEQ

<GGT
  <LAMBDA (N M)
          .
          .
          .
    siehe LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS GGT KOMMENTAR (GROESSTER GEMEINSAMER TEILER)
          RELATION
          (STANDARD ((== . EQ)
                     (/= . NEQ)
                     (%< . LESSP)
```



```
                     (%> . GREATERP)
                     (%<= . LESSEQP)
                     (%>= . GREATEREQP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
                .
                .
                .
    siege Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```

4.2.3. Kleinstes Gemeinsames Vielfaches (KGV)

Auch dieser Algorithmus arbeitet auf natürlichen Zahlen, so daß als angemessener Datentyp wieder die durch die Voraussetzungen auf ihren positiven Anteil eingeschränkten ganzen Zahlen benutzt werden. Infolgedessen kommt auch wieder der Standardrelationssatz zur Anwendung.

```
ALGORITHMUS-NAME: KGV;

PARAMETERTYPEN:   BENUTZER: INTEGER N,
                            INTEGER M;
                  SYSTEM:   INTEGER SPEICHER :=
                              (COND ((GREATERP N M) N)
                                    (T M));

VORAUSSETZUNGEN:  N > 0   ,   M > 0;

ALGORITHMUS:      KGV (N,M,SPEICHER) =
                {   N < M    ---->   KGV(M,N,SPEICHER);
                    N >= M   UND     (IREMAINDER SPEICHER M) = 0
                             ---->   SPEICHER;
                    N >= M   UND     (IREMAINDER SPEICHER M) /= 0
                             ---->   KGV(N,M,(IPLUS SPEICHER N)) }

KOMMENTAR:        KLEINSTES GEMEINSAMEN VIELFACHES!!
```

Dem Systemparameter SPEICHER wird unter dem Schlüssel PARAMETERTYPEN: SYSTEM: eine COND-Form als Wert zugewiesen, deren syntaktische Korrektheit, soweit möglich, durch die Funktion FKT-SYMBOL-IN innerhalb von RALGSPEC überprüft wird. Ob die Bedeutung dieses Ausdrucks sinnvoll ist, wird nicht untersucht.

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: KGV
ALGORITHMUS-NAME: (KGV)
PARAMETERTYPEN:    BENUTZER: ((N . INTEGER) (M . INTEGER))
                   SYSTEM:   ((SPEICHER INTEGER
                                        (COND ((GREATERP N M)N)
                                              (T M))))
VORAUSSETZUNGEN:   ((GROESSERP N 0) (GROESSERP M 0))
ALGORITHMUS:       PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : (((N < M)))
R1 : ((KGV ((M) , (N) , (SPEICHER))))
L2 : (((N >= M) UND ((IREMAINDER SPEICHER M) == 0)))
R2 : ((SPEICHER))
L3 : (((N >= M) UND ((IREMAINDER SPEICHER M) /= 0)))
R3 : ((KGV ((N) , (M) , ((IPLUS SPEICHER N)))))
LINKESEITEN:  (L1 L2 L3)
RECHTESEITEN: (R1 R2 R3)
KOMMENTAR:         (KLEINSTES GEMEINSAMES VIELFACHES)
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (N < M))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . (N >= M))
ERGEBNIS     :      L2 VGL2 (EXPR . ((IREMAINDER SPEICHER
                                                 M) == 0))
ERGEBNIS     :      L2 LOGOP (AND)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . (N >= M))
ERGEBNIS     :      L3 VGL2 (EXPR . ((IREMAINDER SPEICHER
                                                 M) /= 0))
ERGEBNIS     :      L3 LOGOP (AND)
```



```
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . N)
ERGEBNIS     :      L1 VGL1 (RECHTS . M)
ERGEBNIS     :      L1 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (LESSP N M))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . N)
ERGEBNIS     :      L2 VGL1 (RECHTS . M)
ERGEBNIS     :      L2 VGL1 (REL . >=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (GREATEREQP N M))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (LINKS . (IREMAINDER SPEICHER M))
ERGEBNIS     :      L2 VGL2 (RECHTS . 0)
ERGEBNIS     :      L2 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (FORM . (EQ (IREMAINDER SPEICHER M)
                                        0))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . N)
ERGEBNIS     :      L3 VGL1 (RECHTS . M)
ERGEBNIS     :      L3 VGL1 (REL . >=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (GREATEREQP N M))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (LINKS . (IREMAINDER SPEICHER M))
```

```
ERGEBNIS     :       L3 VGL2 (RECHTS . 0)
ERGEBNIS     :       L3 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 VGL2 (FORM . (NEQ (IREMAINDER SPEICHER
                                                    M) 0))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :       L1 FINAL ((LESSP N M))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 FINAL ((AND (GREATEREQP N M)
                                    (EQ (IREMAINDER SPEICHER
                                                    M) 0)))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 FINAL ((AND (GREATEREQP N M)
                                    (NEQ (IREMAINDER SPEICHER
                                                    M) 0)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R1
---REGEL     : S4-54 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 KOPIERT2 (N)

REGELANWENDUNG:  S4-9 AUF R1
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG1 (EXPR . (M))
ERGEBNIS     :       R1 ARG2 (EXPR . (N))
ERGEBNIS     :       R1 ARG3 (EXPR . (SPEICHER))
ERGEBNIS     :       R1 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R2

REGELANWENDUNG:  S4-2 AUF R2
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 FORM (SPEICHER)
ERGEBNIS     :       R2 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
```



```
ERGEBNIS     :       R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (N))
ERGEBNIS     :       R3 ARG2 (EXPR . (M))
ERGEBNIS     :       R3 ARG3 (EXPR . ((IPLUS SPEICHER N)))
ERGEBNIS     :       R3 ARGLIST (ARG1 ARG2 ARG3)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG1 (EXPR . (SETQ N M))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG2 (EXPR . (SETQ M X2))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG2

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG3 (EXPR . (SETQ SPEICHER SPEICHER))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (SETQ N N))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (EXPR . (SETQ M M))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG3
```

```
---REGEL    : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG3 (LISP-AUFRUF IPLUS)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG3
---REGEL    : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG3 (FORM . (SETQ SPEICHER
                                      (IPLUS SPEICHER N)))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL    : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 FINAL ((DUPLIZIERE N X2)
                             (SETQ N M)
                             (SETQ M X2)
                             (KGV N M SPEICHER))

REGELANWENDUNG:  S6-1 AUF R2

REGELANWENDUNG:  S6-21 AUF R2
---REGEL    : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 FINAL (SPEICHER)

REGELANWENDUNG:  S6-1 AUF R3
---REGEL    : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 FINAL ((SETQ SPEICHER
                                   (IPLUS SPEICHER N))
                             (KGV N M SPEICHER))


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((LESSP N M)
        (DUPLIZIERE N X2)
        (SETQ N M)
        (SETQ M X2)
        (KGV N M SPEICHER))
      ((AND (GREATEREQP N M)
            (EQ (IREMAINDER SPEICHER M)
                0))
        SPEICHER)
      ((AND (GREATEREQP N M)
            (NEQ (IREMAINDER SPEICHER M)
                 0))
        (SETQ SPEICHER (IPLUS SPEICHER N))
        (KGV N M SPEICHER))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-54 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)
```



```
REGELANWENDUNGSFAKTOR:  0.777778

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(GROESSERP LESSP GREATEREQP EQ NEQ DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<KGV
 <LAMBDA (N M SPEICHER)
  (COND
   ((OR (TYPEP N 22)
        (TYPEP N 23))
     (COND
      ((OR (TYPEP M 22)
           (TYPEP M 23))
       (COND
        ((GETD (QUOTE DUPLIZIERE))
          (COND
           ((GETD (QUOTE GREATEREQP))
             (COND
              ((GETD (QUOTE GROESSERP))
                (COND
                 ((GROESSERP M 0)
                   (COND
                    ((GROESSERP N 0)
                      (SETQ SPEICHER (COND
                           ((GREATERP N M)
                             N)
                           (T M)))
                      <RPLACA (QUOTE KGV)
                              (DSUBST
                                <QUOTE (RETEVAL
                                         (STKPOS (QUOTE KGV)
                                                 -1)
                                         (QUOTE (APPLY KGV
                                                       NIL>
                                (QUOTE (KGV N M SPEICHER))
                                (COPY
                                  (GETP (QUOTE KGV)
                                        (QUOTE ALGORITHMUS>
                        (APPLY KGV NIL))
                    (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                               (QUOTE (GROESSERP N 0))
                               "NICHT ERFUELLT")
                       NIL)))
                (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                           (QUOTE (GROESSERP M 0))
                           "NICHT ERFUELLT")
```

```
                    (BREAK1 NIL T)
                    NIL)))
               (T (PRINTL "***ERROR*** HILFSFUNKTION"
                          (QUOTE GROESSERP)
                          "UNDEFINIERT")
                  (BREAK1 NIL T)
                  NIL)))
            (T (PRINTL "***ERROR*** HILFSFUNKTION"
                       (QUOTE GREATEREQP)
                       "UNDEFINIERT")
               (BREAK1 NIL T)
               NIL)))
         (T (PRINTL "***ERROR*** HILFSFUNKTION"
                    (QUOTE DUPLIZIERE)
                    "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
      (T (PRINTL "***ERROR***" M "NICHT VOM TYP INTEGER")
         (BREAK1 NIL T)
         NIL)))
   (T (PRINTL "***ERROR***" N "NICHT VOM TYP INTEGER")
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET
```

Das Atom KGV besitzt nun folgende Bindungen:

```
(DEFINEQ

<KGV
  <LAMBDA (N M SPEICHER)
          .
          .
          .
     siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS KGV KOMMENTAR (KLEINSTES GEMEINSAMES VIELFACHES)
          RELATION
          (STANDARD ((== . EQ)
                     (/= . NEQ)
                     (%< . LESSP)
                     (%> . GREATERP)
                     (%<= . LESSEQP)
                     (%>= . GREATEREQP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
                .
                .
                .
     siehe Ergebnis der Automatischen Codierung
                .
                .
```



```
                .    >
STOP
```

4.3. Suchalgorithmen

Nach den Numerischen Algorithmen, die ja ausschließlich auf dem Datentyp Integer definiert sind, folgen in diesem Abschnitt einige gängige Suchalgorithmen, die wir [KNU 73] entnommen haben.
Als Datentypen werden hier von ADTCOMP implementierte Speicherstrukturen verwendet, in den meisten Fällen der Typ OFFENELISTE. Dieser eignet sich für viele Such- und Sortieralgorithmen, da der Indexzugriff der Lese-, Schreib- und Löschoperatoren Manipulationen an beliebigen Stellen der Datenstruktur erlaubt.
Es gibt jedoch Ausnahmen, wie der auf Kellern definierte Merge-Algorithmus (siehe 4.4.1.) oder der Heap-Sort-Algorithmus [KNU 73,p.146], für den ein Binärer Baum die angemessene Datenstruktur wäre.

Die Beschreibungen der implementierten Suchalgorithmen sind wie folgt angeordnet:

Sequential-Search	in 4.3.1.,
Sequential-Search-on-ordered-Tables	in 4.3.2.,
Binary-Search	in 4.3.3. und
Fibonaccian-Search	in 4.3.4.

4.3.1. Sequential-Search

Dieser Algorithmus operiert auf dem Datentyp OFFENELISTE-LISPATOM, auch einer Offenen Liste also, deren Elemente vom Typ LISPATOM sein müssen. Ein LISPATOM ist dabei entweder ein literales Atom oder eine ganze Zahl; es gilt also LISPATOM = LITATOM oder INTEGER.
Sequential-Search sucht in der Offenen Liste OL vom kleinsten zum größten Index nach dem Element KEY; es handelt sich hier also um einen einfachen, sequentiellen Suchalgorithmus.

Sequential-Search wird folgendermaßen spezifiziert:

```
ALGORITHMUS-NAME: SEQUENTIAL-SEARCH;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-LISPATOM OL,
                            LISPATOM KEY;
                  SYSTEM:   INTEGER INDEX := 1;

RELATION:         ALPHORDER;

ALGORITHMUS:      SEQUENTIAL-SEARCH (OL,KEY,INDEX) =
                  {   (OL LIES INDEX) = KEY
                    ---> TRUE;
```



```
                      (OL LIES INDEX) /= KEY     UND
                      (OL LIES INDEX) /= UNDEF
                    ---> SEQUENTIAL-SEARCH(OL,KEY,(ADD1 INDEX));
                      (OL LIES INDEX) = UNDEF
                    ---> FALSE   }

KOMMENTAR:          EINFACHES SERIELLES SUCHEN NACH KEY IN OL.
                    OL BRAUCHT NICHT GEORDNET ZU SEIN.
                    [KNU 73,P.393]!!
```

Da die Elemente der Offenen Liste, auf der Sequential-Search arbeitet, vom Typ LISPATOM sein müssen, kommt der Relationssatz ALPHORDER zur Anwendung (s.a. 5.3.2.,S.509). Die Relationen dieses Satzes benutzen die INTERLISP-Funktion ALPHORDER, die literale Atome (auch Sonderzeichen) sowie Zahlen lexikographisch ordnet.

Es folgt das Implementierungsprotokoll:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: SEQS
ALGORITHMUS-NAME: (SEQUENTIAL-SEARCH)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-LISPATOM)
                             (KEY . LISPATOM))
                  SYSTEM:   ((INDEX INTEGER 1))
RELATION:         (ALPHORDER)
RELATIONSSATZ:    ((=  . EQ)(/= . NEQ)(<  . LESS-ALPHP)
                   (>  . GREATER-ALPHP)(<= . LESSEQ-ALPHP)
                   (>= . GREATEREQ-ALPHP))
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES INDEX) == KEY)))
R1 : ((TRUE))
L2 : ((((OL LIES INDEX) /= KEY) UND
       ((OL LIES INDEX) /= UNDEF)))
R2 : ((SEQUENTIAL-SEARCH ((OL) , (KEY) , ((ADD1 INDEX)))))
L3 : ((((OL LIES INDEX) == UNDEF)))
R3 : ((FALSE))
LINKESEITEN:  (L1 L2 L3)
RECHTESEITEN: (R1 R2 R3)
KOMMENTAR:        (EINFACHES SERIELLES SUCHEN NACH KEY IN OL.
                   OL BRAUCHT NICHT GEORDNER ZU SEIN.
                   [KNU 73,P.393])
ALG-EINGABE BEENDET.
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES INDEX) == KEY))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)
```

```
REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 VGL1 (EXPR . ((OL LIES INDEX) /= KEY))
ERGEBNIS     :       L2 VGL2 (EXPR . ((OL LIES INDEX) /= UNDEF))
ERGEBNIS     :       L2 LOGOP (AND)
ERGEBNIS     :       L2 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 VGL1 (EXPR . ((OL LIES INDEX) == UNDEF))
ERGEBNIS     :       L3 LOGOP (NONE)
ERGEBNIS     :       L3 VGLLIST (VGL1)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :       L1 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :       L1 VGL1 (RECHTS . KEY)
ERGEBNIS     :       L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :       L1 VGL1 (FORM . (EQ (OL LIES INDEX) KEY))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :       L2 VGL1 (RECHTS . KEY)
ERGEBNIS     :       L2 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 VGL1 (FORM . (NEQ (OL LIES INDEX) KEY))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 VGL2 (LINKS . (OL LIES INDEX))
ERGEBNIS     :       L2 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :       L2 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 VGL2 (FORM . (NEQ (OL LIES INDEX)
                                          (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :       L3 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :       L3 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
```



```
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 VGL1 (FORM . (EQ (OL LIES INDEX)
                                         (QUOTE UNDEF)))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :       L1 FINAL ((EQ (OL LIES INDEX) KEY))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :       L2 FINAL ((AND (NEQ (OL LIES INDEX) KEY)
                                    (NEQ (OL LIES INDEX)
                                         (QUOTE UNDEF))))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 FINAL ((EQ (OL LIES INDEX)
                                   (QUOTE UNDEF)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FORM (T)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (EXPR . (OL))
ERGEBNIS     :       R2 ARG2 (EXPR . (KEY))
ERGEBNIS     :       R2 ARG3 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :       R2 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R3

REGELANWENDUNG:  S4-2 AUF R3
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 FORM (NOFORM)
SCHRITT-4 ENDE
```

```
SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (FORM . (SETQ INDEX (ADD1 INDEX)))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL (T)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((SETQ INDEX (ADD1 INDEX))
                              (SEQUENTIAL-SEARCH OL KEX INDEX))

REGELANWENDUNG:  S6-1 AUF R3

REGELANWENDUNG:  S6-21 AUF R3
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 FINAL ('NIL)
```



```
START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ (OL LIES INDEX)
           KEY)
        T)
      ((AND (NEQ (OL LIES INDEX)
                 KEY)
            (NEQ (OL LIES INDEX)
                 (QUOTE UNDEF)))
        (SETQ INDEX (ADD1 INDEX))
        (SEQUENTIAL-SEARCH OL KEY INDEX))
      ((EQ (OL LIES INDEX)
           (QUOTE UNDEF))
        NIL)
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(s1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.794118

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ NEQ)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET

Da die Hilfsfunktionen EQ und NEQ in INTERLISP zur Verfügung
stehen, werden sie aus der Liste der Hilfsfunktionen gestri-
chen. Eine Abfrage wird nicht implementiert.

Es folgt die LISP-Phase:

START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<SEQUENTIAL-SEARCH
 <LAMBDA (OL KEY INDEX)
  (COND
   ((EQ (QUOTE OFFENELISTE-LISPATOM)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((OR (TYPEP KEY 30)
          (TYPEP KEY 22)
          (TYPEP KEY 23))
       (SETQ INDEX 1)
       <RPLACA (QUOTE SEQUENTIAL-SEARCH)
               (DSUBST OL (QUOTE OL)
                       (DSUBST
                         <QUOTE
```

```
                    ---(RETEVAL
                        (STKPOS
                          (QUOTE SEQUENTIAL-SEARCH)
                          -1)
                        (QUOTE (APPLY SEQUENTIAL-SEARCH
                                      NIL>
                    (QUOTE (SEQUENTIAL-SEARCH OL
                                              KEY
                                              INDEX))
                    (COPY
                      (GETP (QUOTE SEQUENTIAL-SEARCH)
                            (QUOTE ALGORITHMUS>
                (APPLY SEQUENTIAL-SEARCH NIL))
    (T (PRINTL "***ERROR***" KEY "NICHT VOM TYP LISPATOM")
       (BREAK1 NIL T)
       NIL)))
  (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
             (QUOTE OFFENELISTE-LISPATOM))
     (BREAK1 NIL T)
     NIL>>
LISP-PHASE BEENDET
```

Der fertig implementierte Algorithmus sieht nun folgendermaßen aus:

```
(DEFINEQ

<SEQUENTIEL-SEARCH
  <LAMBDA (OL KEY INDEX)
          .
          .
          .
    siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS SEQUENTIAL-SEARCH KOMMENTAR (EINFACHES SERIELLES
          DURCHSUCHEN VON OL NACH KEY. OL BRAUCHT NICHT
          GEORDNET ZU SEIN. [KNU 73 , P.393])
          RELATION
          (ALPHORDER ((== . EQ)
                      (/= . NEQ)
                      (%< . LESS-ALPHP)
                      (%> . GREATER-ALPHP)
                      (%<= . LESSEQ-ALPHP)
                      (%>= . GREATEREQ-ALPHP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
              .
              .
              .
    siehe Ergebnis der Automatischen Codierung
              .
              .
```



```
              .    >
STOP
```

4.3.2. Sequential-Search-on-ordered-Tables

Im Gegensatz zu dem vorhergehenden Algorithmus setzt Sequential-Search-on-ordered-Tables Datenstrukturen voraus, die aufsteigend geordnet sind. Wieder wird auf dem Datentyp OFFENELISTE-LISPATOM gearbeitet; dabei muß jetzt der Wert des LISPATOMs mit dem Index i größer oder gleich dem Wert des LISPATOMs mit dem Index i+1 sein.
Da bei dieser Version des seriellen Suchens eine bereits sortierte Suchmenge vorausgesetzt wird, arbeitet dieser Algorithmus im allgemeinen schneller als der einfache Sequential-Search.

```
ALGORITHMUS-NAME: SEQUENTIAL-SEARCH-OOT;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-LISPATOM OL,
                            LISPATOM KEY;
                  SYSTEM:   INTEGER INDEX := 1;

RELATION:         ALPHORDER;

ALGORITHMUS:      SEQUENTIAL-SEARCH-OOT (OL,KEY,INDEX) =
                  { (OL LIES INDEX) = KEY
                  ---> TRUE;
                    (OL LIES INDEX) = UNDEF     ODER
                    (OL LIES INDEX) < KEY
                  ---> FALSE;
                    (OL LIES INDEX) /= UNDEF    UND
                    (OL LIES INDEX) > KEY
                  ---> SEQUENTIAL-SEARCH-OOT(OL,KEY,
                                             (ADD1 INDEX)) }

+KOMMENTAR:       SEQUENTIELLES SUCHEN "ON ORDERD TABLES"
                  (OOT). OL WIRD ALS GEORDNET VORAUSGESETZT
                  [KNU 73,P.296]!!
```

Wegen der Verwendung von LISPATOM als Parametertyp wird als Relation ALPHORDER spezifiziert.

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: SEQSOOT
ALGORITHMUS-NAME: (SEQUENTIAL-SEARCH-OOT)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-LISPATOM)
                             (KEY . LISPATOM))
                  SYSTEM:   ((INDEX INTEGER 1))
RELATION:         (ALPHORDER)
RELATIONSSATZ:     ((== . EQ)(/= . NEQ)(< . LESS-ALPHP)
                   (> . GREATER-ALPHP)(<= . LESSEQ-ALPHP)
                   (>= . GREATEREQ-ALPHP))
```

```
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES INDEX) == KEY)))
R1 : ((TRUE))
L2 : ((((OL LIES INDEX) == UNDEF) ODER
      ((OL LIES INDEX) < KEY)))
R2 : ((FALSE))
L3 : ((((OL LIES INDEX) /= UNDEF) UND
      ((OL LIES INDEX) > KEY)))
R3 : ((SEQUENTIAL-SEARCH-OOT ((OL) , (KEY) , ((ADD1 INDEX)))))
LINKESEITEN:  (L1 L2 L3)
RECHTESEITEN: (R1 R2 R3)
KOMMENTAR:        (SEQUENTIELLES SUCHEN " ON ORDERD TABLES "
                   (OOT) . OL WIRD ALS GEORDNET VORAUSGESETZT.
                   [KNU 73 , P.396])
ALG-EINGABE BEENDET.
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES INDEX) == KEY))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2

REGELANWENDUNG:  S1-2 AUF L2
---REGEL     : S1-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES INDEX) == UNDEF))
ERGEBNIS     :      L2 VGL2 (EXPR . ((OL LIES INDEX) < KEY))
ERGEBNIS     :      L2 LOGOP (OR)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES INDEX) /= UNDEF))
ERGEBNIS     :      L3 VGL2 (EXPR . ((OL LIES INDEX) > KEY))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L1 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
```



```
ERGEBNIS     :      L1 VGL1 (FORM . (EQ (OL LIES INDEX) KEY))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L2 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L2 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (EQ (OL LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L2 VGL2 (RECHTS . KEY)
ERGEBNIS     :      L2 VGL2 (REL . <)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (FORM . (LESS-ALPHP (OL LIES INDEX)
                                                 KEY))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L3 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (NEQ (OL LIES INDEX)
                                         (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L3 VGL2 (RECHTS . KEY)
ERGEBNIS     :      L3 VGL2 (REL . >)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (FORM . (GREATER-ALPHP (OL LIES
                                                    INDEX)
                                                  KEY))
SCHRITT-2 ENDE
```

```
SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ (OL LIES INDEX) KEY))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((OR (EQ (OL LIES INDEX)
                                      (QUOTE UNDEF))
                                 (LESS-ALPHP (OL LIES INDEX)
                                             KEY)))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((AND (NEQ (OL LIES INDEX)
                                        (QUOTE UNDEF))
                                   (GREATER-ALPHP (OL LIES
                                                      INDEX)
                                                  KEY)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (T)

REGELANWENDUNG:  S4-1 AUF R2

REGELANWENDUNG:  S4-2 AUF R2
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FORM (NOFORM)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R3 ARG3 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3)
SCHRITT-4 ENDE
```



```
SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM . (SETQ INDEX (ADD1 INDEX)))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL (T)

REGELANWENDUNG:  S6-1 AUF R2

REGELANWENDUNG:  S6-21 AUF R2
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ('NIL)

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ INDEX (ADD1 INDEX))
                              (SEQUENTIAL-SEARCH-OOT OL
                                                     KEY
                                                     INDEX))
```

```
START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ (OL LIES INDEX)
           KEY)
        T)
      ((OR (EQ (OL LIES INDEX)
               (QUOTE UNDEF))
           (LESS-ALPHP (OL LIES INDEX)
                       KEY))
        NIL)
      ((AND (NEQ (OL LIES INDEX)
                 (QUOTE UNDEF))
            (GREATER-ALPHP (OL LIES INDEX)
                           KEY))
        (SETQ INDEX (ADD1 INDEX))
        (SEQUENTIAL-SEARCH-OOT OL KEY INDEX))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S1-2 S2-1 S2-9 S3-1 S4-1 S4-2 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.783784

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ LESS-ALPHP NEQ GREATER-ALPHP)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<SEQUENTIAL-SEARCH-OOT
 <LAMBDA (OL KEY INDEX)
  (COND
   ((EQ (QUOTE OFFENELISTE-LISPATOM)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
     (COND
      ((OR (TYPEP KEY 30)
           (TYPEP KEY 22)
           (TYPEP KEY 23))
        (COND
         ((GETD (QUOTE GREATER-ALPHP))
           (COND
            ((GETD (QUOTE LESS-ALPHP))
              (SETQ INDEX 1)
              <RPLACA (QUOTE SEQUENTIAL-SEARCH-OOT)
                      (DSUBST OL (QUOTE OL)
                        (DSUBST
```



```
                                <QUOTE
                                  (RETEVAL
                                    (STKPOS
                                      (QUOTE SEQUENTIAL-SEARCH-OOT)
                                      -1)
                                    (QUOTE
                                      (APPLY SEQUENTIAL-SEARCH-OOT
                                             NIL>
                                <QUOTE
                                  (SEQUENTIAL-SEARCH-OOT OL
                                                         KEY
                                                         INDEX))
                                (COPY
                                  (GETP
                                    (QUOTE SEQUENTIAL-SEARCH-OOT)
                                    (QUOTE ALGORITHMUS>
                          (APPLY SEQUENTIAL-SEARCH-OOT NIL))
                        (T (PRINTL "***ERROR*** HILFSFUNKTION"
                                   (QUOTE LESS-ALPHP)
                                   "UNDEFINIERT")
                           (BREAK1 NIL T)
                           NIL)))
                    (T (PRINTL "***ERROR*** HILFSFUNKTION"
                               (QUOTE GREATER-ALPHP)
                               "UNDEFINIERT")
                       (BREAK1 NIL T)
                       NIL)))
                (T (PRINTL "***ERROR***" KEY "NICHT VOM TYP LISPATOM")
                   (BREAK1 NIL T)
                   NIL)))
          (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
                     (QUOTE OFFENELISTE-LISPATOM))
             (BREAK1 NIL T)
             NIL>>
LISP-PHASE BEENDET

Nach der Implementierung besitzt das Atom SEQUENTIAL-SEARCH-
OOT folgende Bindungen:

(DEFINEQ

<SEQUENTIAL-SEARCH-OOT
  <LAMBDA (OL KEY INDEX)
            .
            .
            .
     siehe Ergebnis LISP-Phase
            .
            .
            .    >>
)
  <PUTPROPS SEQUENTIAL-SEARCH-OOT KOMMENTAR
            (SEQUENTIELLES SUCHEN %" ON ORDERD TABLES %"
            %( OOT %) %. OL WIRD ALS GEORDNET
            VORAUSGESETZT. [KNU 73 , P.396])
            RELATION
```

```
          (ALPHORDER ((== . EQ)
                      (/= . NEQ)
                      (%< . LESS-ALPHP)
                      (%> . GREATER-ALPHP)
                      (%<= . LESSEQ-ALPHP)
                      (%>= . GREATEREQ-ALPHP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
                .
                .
                .
    siehe Ergebnis der Automatischen Codierung
                .
                .
                .   >
STOP
```

4.3.3. Binary-Search

Beim binären Suchen wird das Suchintervall, das von ANF und END begrenzt wird, durch den Wert des Parameter MITTE halbiert; dann wird entschieden, in welcher Hälfte sich das gesuchte Element befinden kann. Diese Hälfte wird das neue Suchintervall für den nächsten rekursiven Aufruf von Binary-Search.

Binary-Search ist folgendermaßen spezifiziert:

```
ALGORITHMUS-NAME: BIN-SEARCH;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-INTEGER  OL,
                            INTEGER KEY;
                  SYSTEM:   INTEGER ANF    := 1,
                            INTEGER MITTE := (IQUOTIENT
                                               (LISTSIZE OL)
                                               2),
                            INTEGER END    := (LISTSIZE OL);

HILFSALGORITHMEN: LISTSIZE;

ALGORITHMUS:      BIN-SEARCH (OL,KEY,ANF,MITTE,END) =

            {     ANF > END                  ---->   FALSE;
                  (OL LIES MITTE) = KEY      ---->   TRUE;
                  (OL LIES MITTE) > KEY
            ----> BIN-SEARCH (OL,
                              KEY,
                              ANF,
                              (IQUOTIENT(IPLUS ANF
                                               (SUB1 MITTE))
                                         2),
                              (SUB1 MITTE));
```



```
                  (OL LIES MITTE) < KEY
            ----> BIN-SEARCH(OL,KEY,
                             KEY,
                             (ADD1 MITTE),
                             (IQUOTIENT(IPLUS (ADD1 MITTE)
                                              END)
                                        2),
                             END)   }

KOMMENTAR:BIN-SEARCH SUCHT IN DER SORTIERTEN OFFENENLISTE OL
          DAS ELEMENT 'KEY'. ERGEBNIS IST EIN WAHRHEITSWERT.
          [KNU 73,P.407]!!
```

Beim Initialisieren der System-Parameter MITTE und END wird die Funktion LISTSIZE (siehe 5.3.2.,S.508) verwendet. Da in der Einlese-Phase durch die Funktion FKT-SYMBOL-IN innerhalb von RALGSPEC die Korrektheit solcher Wertzuweisungen überprüft wird und LISTSIZE keine INTERLISP-Funktion wie die ebenfalls verwendete Funktion IQUOTIENT ist, muß LISTSIZE als Hilfsalgorithmus deklariert werden.
LISTSIZE liefert als Wert die Anzahl der in einer von ADTCOMP implementierten Datenstruktur gespeicherten Elemente.

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: BINS
ALGORITHMUS-NAME: (BIN-SEARCH)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-INTEGER)
                             (KEY . INTEGER))
                  SYSTEM:   ((ANF INTEGER 1)
                             (MITTE INTEGER
                                    (IQUOTIENT (LISTSIZE OL)
                                               2))
                             (END INTEGER (LISTSIZE OL)))
HILFSALGORITHMEN: (LISTSIZE)
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : (((ANF > END)))
R1 : ((FALSE))
L2 : ((((OL LIES MITTE) == KEY)))
R2 : ((TRUE))
L3 : ((((OL LIES MITTE) > KEY)))
R3 : ((BIN-SEARCH ((OL) , (KEY) , (ANF) ,
                   ((IQUOTIENT (IPLUS ANF (SUB1 MITTE)) 2)) ,
                   ((SUB1 MITTE)))))
L4 : ((((OL LIES MITTE) < KEY)))
R4 : ((BIN-SEARCH ((OL) , (KEY) , ((ADD1 MITTE)) ,
                   ((IQUOTIENT (IPLUS (ADD1 MITTE) END) 2)) ,
                   (END))))
LINKESEITEN:  (L1 L2 L3 L4)
RECHTESEITEN: (R1 R2 R3 R4)
KOMMENTAR:    (BIN-SEARCH SUCHT IN DER SORTIERTEN OFFENENLISTE
               OL DAS ELEMNENT 'KEY'. ERGEBNIS IST EIN
               WAHRHEITSWERT. [KNU 73 , P.407])
ALG-EINGABE BEENDET.
RELATION      = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
```

```
                (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET

START ALG-KONSTRUKTIONS-PHASE
SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (ANF > END))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES MITTE) == KEY))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES MITTE) > KEY))
ERGEBNIS     :      L3 LOGOP (NONE)
ERGEBNIS     :      L3 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((OL LIES MITTE) < KEY))
ERGEBNIS     :      L4 LOGOP (NONE)
ERGEBNIS     :      L4 VGLLIST (VGL1)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . ANF)
ERGEBNIS     :      L1 VGL1 (RECHTS . END)
ERGEBNIS     :      L1 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (GREATERP ANF END))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES MITTE))
ERGEBNIS     :      L2 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L2 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
```



```
ERGEBNIS     :      L2 VGL1 (FORM . (EQ (OL LIES MITTE) KEY))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL LIES MITTE))
ERGEBNIS     :      L3 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L3 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (GREATERP (OL LIES MITTE)
                                              KEY))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (LINKS . (OL LIES MITTE))
ERGEBNIS     :      L4 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L4 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (FORM . (LESSP (OL LIES MITTE)
                                           KEY))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((GREATERP ANF END))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((EQ (OL LIES MITTE) KEY))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((GREATERP (OL LIES MITTE) KEY))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 FINAL ((LESSP (OL LIES MITTE) KEY))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (NOFORM)

REGELANWENDUNG:  S4-1 AUF R2
```

```
REGELANWENDUNG:  S4-2 AUF R2
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FORM (T)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3

REGELANWENDUNG:  S4-52 AUF R3

REGELANWENDUNG:  S4-53 AUF R3

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R3 ARG3 (EXPR . (ANF))
ERGEBNIS     :      R3 ARG4 (EXPR . ((IQUOTIENT
                                (IPLUS ANF
                                       (SUB1 MITTE))
                                2)))
ERGEBNIS     :      R3 ARG5 (EXPR . ((SUB1 MITTE)))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5)

REGELANWENDUNG:  S4-1 AUF R4
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R4

REGELANWENDUNG:  S4-52 AUF R4

REGELANWENDUNG:  S4-53 AUF R4

REGELANWENDUNG:  S4-54 AUF R4

REGELANWENDUNG:  S4-55 AUF R4

REGELANWENDUNG:  S4-9 AUF R4
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (EXPR . (OL))
ERGEBNIS     :      R4 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R4 ARG3 (EXPR . ((ADD1 MITTE)))
ERGEBNIS     :      R4 ARG4 (EXPR . ((IQUOTIENT
                                (IPLUS (ADD1 MITTE)
                                       END)
                                2)))
ERGEBNIS     :      R4 ARG5 (EXPR . (END))
ERGEBNIS     :      R4 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5)
SCHRITT-4 ENDE
```



```
SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG4

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (LISP-AUFRUF IQUOTIENT)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (FORM . (SETQ MITTE
                                (IQUOTIENT
                                   (IPLUS ANF
                                          (SUB1
                                             MITTE))
                                2)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG5

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG5 (LISP-AUFRUF SUB1)
ERGEBNIS     :      R3 ARG5 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG5 (FORM . (SETQ END (SUB1 STD1-5)))
ERGEBNIS     :      R3 KOPIERT-LISP (ARG5 (MITTE STD1-5))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (EXPR . (SETQ OL OL))
```

```
REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R4 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG3 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R4 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG3 (FORM . (SETQ ANF (ADD1 MITTE)))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG4

REGELANWENDUNG:  S5-2 AUF R4 , ARGUMENT ARG4
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG4 (LISP-AUFRUF IQUOTIENT)
ERGEBNIS     :       R4 ARG4 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R4 , ARGUMENT ARG4
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG4 (FORM . (SETQ
                                      (IQUOTIENT
                                         (IPLUS (ADD1
                                                   MITTE)
                                                END)
                                         2)))
ERGEBNIS     :       R4 KOPIERT-LISP (ARG4 (2 2)(MITTE MITTE)
                                               (END END))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG5 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG5 (FORM .)
SCHRITT-5 ENDE
```



```
SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL ('NIL)

REGELANWENDUNG:  S6-1 AUF R2

REGELANWENDUNG:  S6-21 AUF R2
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 FINAL (T)

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 FINAL ((DUPLIZIERE MITTE (STD1-5))
                               (SETQ MITTE
                                     (IQUOTIENT
                                        (IPLUS ANF
                                               (SUB1 MITTE))
                                        2))
                               (SETQ END (SUB1 STD1-5))
                               (BIN-SEARCH OL KEY
                                           ANF MITTE END))

REGELANWENDUNG:  S6-1 AUF R4
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 FINAL ((SETQ ANF (ADD1 MITTE))
                               (SETQ MITTE
                                     (IQUOTIENT
                                        (IPLUS (ADD1 MITTE)
                                               END)
                                        2))
                               (BIN-SEARCH OL KEY
                                           ANF MITTE END))

START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((GREATERP ANF END)
        NIL)
      ((EQ (OL LIES MITTE)
           KEY)
        T)
      ((GREATERP (OL LIES MITTE)
                 KEY)
        (DUPLIZIERE MITTE (STD1-5))
        (SETQ MITTE (IQUOTIENT (IPLUS ANF (SUB1 MITTE))
                               2))
        (SETQ END (SUB1 STD1-5))
        (BIN-SEARCH OL KEY ANF MITTE END))
      ((LESSP (OL LIES MITTE)
```

```
            KEY)
        (SETQ ANF (ADD1 MITTE))
        (SETQ MITTE (IQUOTIENT (IPLUS (ADD1 MITTE)
                                      END)
                               2))
        (BIN-SEARCH OL KEY ANF MITTE END))
       (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-2 S2-9 S3-1 S4-2 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.707692

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(GREATERP EQ LESSP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<BIN-SEARCH
 <LAMBDA (OL KEY ANF MITTE END)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((OR (TYPEP KEY 22)
          (TYPEP KEY 23))
      (COND
       ((GETD (QUOTE DUPLIZIERE))
        (COND
         ((GETD (QUOTE LISTSIZE))
          (SETQ ANF 1)
          (SETQ MITTE (IQUOTIENT (LISTSIZE OL)
                                 2))
          (SETQ END (LISTSIZE OL))
          <RPLACA (QUOTE BIN-SEARCH)
                  (DSUBST OL (QUOTE OL)
                    (DSUBST
                      <QUOTE
                        (RETEVAL
                          (STKPOS (QUOTE BIN-SEARCH)
                                  -1)
                          (QUOTE (APPLY BIN-SEARCH NIL>
                      (QUOTE (BIN-SEARCH OL KEY
                                         ANF MITTE END))
                      (COPY (GETP (QUOTE BIN-SEARCH)
                                  (QUOTE ALGORITHMUS>
          (APPLY BIN-SEARCH NIL))
         (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
```



```
                    (QUOTE LISTSIZE)
                    "UNDEFINIERT")
                (BREAK1 NIL T)
                NIL)))
          (T (PRINTL "***ERROR*** HILFSFUNKTION"
                     (QUOTE DUPLIZIERE)
                     "UNDEFINIERT")
             (BREAK1 NIL T)
             NIL)))
       (T (PRINTL "***ERROR***" KEY "NICHT VOM TYP INTEGER")
          (BREAK1 NIL T)
          NIL)))
    (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
               (QUOTE OFFENELISTE-INTEGER))
       (BREAK1 NIL T)
       NIL>>
LISP-PHASE BEENDET

Und abschließend der implementierte Algorithmus:

(DEFINEQ

<BIN-SEARCH
  <LAMBDA (OL KEY ANF MITTE END)
            .
            .
            .
     siehe Ergebnis LISP-Phase
            .
            .
            .    >>
)
  <PUTPROPS BIN-SEARCH KOMMENTAR
            (BIN-SEARCH SUCHT IN DER SORTIERTEN OFFENENLISTE
             OL DAS ELEMENT %'KEY%'. ERGEBNIS IST EIN
             WAHRHEITSWERT. [KNU 73 , P.407])
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                .
                .
                .
     siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```

4.3.4. Fibonaccian-Search

Die Besonderheit beim FIBINACCIAN-SEARCH ist die jeweilige Einschränkung des Suchintervalls. Dabei werden Quotienten aus sukzessiven FIBONACCI-Zahlen gebildet, die dann den aktuellen Suchindex darstellen. Die Initiierung geschieht mit der Funktion FIBO-ZAHL, die in Abhängigkeit vom Wert ihres ersten Argumentes die jeweils erste, zweite oder dritte FIBONACCI-Zahl berechnet, die kleinergleich dem zweiten Argument ist. Für ein Argumentpaar (4 X) ergibt sich ein Wert aus dem Intervall (F X], wobei F die maximale FIBONACCI-Zahl kleinergleich X ist. Dies wird in der Spezifikation für den Fall benötigt, wenn in der ersten Abfrage (OL LIES I) < KEY gilt, d.h., daß das zu suchende Element, wenn es existiert, "rechts" von dem Initiierungswert des Suchindexes liegt.
Für die genaue Erklärung der Indexberechnungen sei auf [KNU 73] verwiesen.

FIBO-SEARCH arbeitet auf geordneten OFFENENLISTEN-INTEGERN.

```
ALGORITHMUS-NAME: FIBO-SEARCH;

PARAMETERTYPEN: BENUTZER: OFFENELISTE-INTEGER OL,
                          INTEGER  KEY;
                SYSTEM:   INTEGER I := (FIBO-ZAHL 1
                                        (LISTSIZE OL)),
                          INTEGER P := (FIBO-ZAHL 2
                                        (LISTSIZE OL)),
                          INTEGER Q := (FIBO-ZAHL 3
                                        (LISTSIZE OL)),
                          INTEGER M := (FIBO-ZAHL 4
                                        (LISTSIZE OL)),
                          BOOL FLAG := FALSE;

HILFSALGORITHMEN: FIBO-ZAHL, LISTSIZE;

ALGORITHMUS:    FIBO-SEARCH (OL,KEY,I,P,Q,M,FLAG) =
          {     (OL LIES I) > KEY     UND     Q /= 0
          ---> FIBO-SEARCH(OL , KEY ,(IDIFFERENCE I Q),
                           Q ,(IDIFFERENCE P Q), M , TRUE);
               (OL LIES I) < KEY     UND
               P /= 1     UND     FLAG = TRUE
          ---> FIBO-SEARCH(OL , KEY ,(IPLUS I Q),
                           (IDIFFERENCE P Q),
                           (ITIMES 2 (IDIFFERENCE Q P)),
                           M , TRUE);
               (OL LIES I) < KEY     UND
               P /= 1     UND     FLAG = FALSE
          ---> FIBO-SEARCH(OL , KEY ,
                           (IDIFFERENCE I
                                       (IDIFFERENCE M Q)),
                           (IDIFFERENCE P Q),
                           (IDIFFERENCE (ITIMES 2 Q) P),
                           M,TRUE);
                (OL LIES I) = KEY
```



```
          ---> TRUE ;
               Q = 0     ODER     P = 1
          ---> FALSE      }

KOMMENTAR: FIBO-SEARCH SUCHT IN DER OFFENENLISTE OL NACH DEM
           SCHLUESSEL ' KEY ' . DABEI WIRD DAS SUCHINTERVALL
           SUKZESSIVE GEMAESS DEM FIBONACCI-VERFAHREN
           EINGESCHRAENKT.
           ERGEBNIS IST EIN WAHRHEITSWERT[KNU 73,P.415]!!
```

Zur Initialisierung der System-Parameter I, P, Q und M werden die Funktionen FIBO-ZAHL und LISTSIZE (siehe 5.3.2.,S. 506 und 508) benötigt, die als Hilfsalgorithmen zu spezifizieren sind.

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: FIBOS
ALGORITHMUS-NAME: (FIBO-SEARCH)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-INTEGER)
                             (KEY . INTEGER))
                  SYSTEM:   ((I INTEGER (FIBO-ZAHL 1
                                         (LISTSIZE OL)))
                             (P INTEGER (FIBO-ZAHL 2
                                         (LISTSIZE OL)))
                             (Q INTEGER (FIBO-ZAHL 3
                                         (LISTSIZE OL)))
                             (M INTEGER (FIBOZAHL 4
                                         (LISTSIZE OL)))
                             (FLAG BOOL FALSE))
HILFSALGORITHMEN: (FIBO-ZAHL LISTSIZE)
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES I) > KEY) UND (Q /= 0)))
R1 : ((FIBO-SEARCH ((OL) , (KEY) , ((IDIFFERENCE I Q)) , (Q) ,
                    ((IDIFFERENCE P Q)) , (M) , (TRUE))))
L2 : ((((OL LIES I) < KEY) UND (P /= 1) UND (FLAG == TRUE)))
R2 : ((FIBO-SEARCH ((OL) , (KEY) , ((IPLUS I Q)) ,
                    ((IDIFFERENCE P Q)) ,
                    ((ITIMES 2 (IDIFFERENCE Q P))) ,
                    (M) , (TRUE))))
L3 : ((((OL LIES I) < KEY) UND (P /= 1) UND (FLAG == FALSE)))


R3 : ((FIBO-SEARCH ((OL) , (KEY) ,
                    ((IDIFFERENCE I (IDIFFERENCE M Q))) ,
                    ((IDIFFERENCE P Q)) ,
                    ((IDIFFERENCE (ITIMES 2 Q) P)) ,
                    (M) , (TRUE))))
L4 : ((((OL LIES I) == KEY)))
R4 : ((TRUE))
L5 : (((Q == 0) ODER (P == 1)))
R5 : ((FALSE))
LINKESEITEN:  (L1 L2 L3 L4 L5)
RECHTESEITEN: (R1 R2 R3 R4 R5)
KOMMENTAR:        (FIBO-SEARCH SUCHT IN DER OFFENENLISTE OL
```

```
                NACH DEM SCHLUESSEL ' KEY ' . DABEI WIRD
                DAS SUCHINTERVALL SUKZESSIVE GEMAESS DEM
                FIBONACCI-VERFAHREN EINGESCHRAENKT.
                ERGEBNIS IST EIN WAHRHEITSWERT.
                [KNU 73 , P.415])
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET

START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES I) > KEY))
ERGEBNIS     :      L1 VGL2 (EXPR . (Q /= 0))
ERGEBNIS     :      L1 LOGOP (AND)
ERGEBNIS     :      L1 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES I) < KEY))
ERGEBNIS     :      L2 VGL2 (EXPR . (P /= 1))
ERGEBNIS     :      L2 VGL3 (EXPR . (FLAG == TRUE))
ERGEBNIS     :      L2 LOGOP (AND)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES I) < KEY))
ERGEBNIS     :      L3 VGL2 (EXPR . (P /= 1))
ERGEBNIS     :      L3 VGL3 (EXPR . (FLAG == FALSE))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((OL LIES I) == KEY))
ERGEBNIS     :      L4 LOGOP (NONE)
ERGEBNIS     :      L4 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L5

REGELANWENDUNG:  S1-2 AUF L5
---REGEL     : S1-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS     :      L5 VGL1 (EXPR . (Q == 0))
ERGEBNIS     :      L5 VGL2 (EXPR . (P == 1))
ERGEBNIS     :      L5 LOGOP (OR)
ERGEBNIS     :      L5 VGLLIST (VGL1 VGL2)
SCHRITT-1 ENDE
```



```
SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL LIES I))
ERGEBNIS     :      L1 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L1 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (GREATERP (OL LIES I) KEY))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (LINKS . Q)
ERGEBNIS     :      L1 VGL2 (RECHTS . 0)
ERGEBNIS     :      L1 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (FORM . (NEQ Q 0))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES I))
ERGEBNIS     :      L2 VGL1 (RECHTS . KEY)
ERGEBNIS     :      L2 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (LESSP (OL LIES I) KEY))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (LINKS . P)
ERGEBNIS     :      L2 VGL2 (RECHTS . 1)
ERGEBNIS     :      L2 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (FORM . (NEQ P 1))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL3

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL3
---REGEL     : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL3 (LINKS . FLAG)
ERGEBNIS     :      L2 VGL3 (RECHTS . T)
ERGEBNIS     :      L2 VGL3 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL3
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
```

```
ERGEBNIS    :      L2 VGL3 (FORM . (EQ FLAG T))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (LINKS . (OL LIES I))
ERGEBNIS    :      L3 VGL1 (RECHTS . KEY)
ERGEBNIS    :      L3 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (FORM . (LESSP (OL LIES I) KEY))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL2 (LINKS . P)
ERGEBNIS    :      L3 VGL2 (RECHTS . 1)
ERGEBNIS    :      L3 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL2 (FORM . (NEQ P 1))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL3

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL3
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL3 (LINKS . FLAG)
ERGEBNIS    :      L3 VGL3 (RECHTS .)
ERGEBNIS    :      L3 VGL3 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL3 (FORM . (EQ FLAG))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL1 (LINKS . (OL LIES I))
ERGEBNIS    :      L4 VGL1 (RECHTS . KEY)
ERGEBNIS    :      L4 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL1 (FORM . (EQ (OL LIES I) KEY))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL1 (LINKS . Q)
ERGEBNIS    :      L5 VGL1 (RECHTS . O)
ERGEBNIS    :      L5 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
```



```
ERGEBNIS    :      L5 VGL1 (FORM . (EQ Q O))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L5 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL2 (LINKS . P)
ERGEBNIS    :      L5 VGL2 (RECHTS . 1)
ERGEBNIS    :      L5 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL2 (FORM . (EQ P 1))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 FINAL ((AND (GREATERP (OL LIES I) KEY)
                                  (NEQ Q O)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 FINAL ((AND (LESSP (OL LIES I) KEY)
                                  (NEQ P 1)
                                  (EQ FLAG T)))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 FINAL ((AND (LESSP (OL LIES I) KEY)
                                  (NEQ P 1)
                                  (EQ FLAG)))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 FINAL ((EQ (OL LIES I) KEY))

REGELANWENDUNG:  S3-1 AUF L5
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 FINAL ((OR (EQ Q O)(EQ P 1)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R1

REGELANWENDUNG:  S4-52 AUF R1

REGELANWENDUNG:  S4-53 AUF R1
```

```
REGELANWENDUNG:  S4-54 AUF R1

REGELANWENDUNG:  S4-55 AUF R1

REGELANWENDUNG:  S4-9 AUF R1
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . (OL))
ERGEBNIS     :      R1 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R1 ARG3 (EXPR . ((IDIFFERENCE I Q)))
ERGEBNIS     :      R1 ARG4 (EXPR . (Q))
ERGEBNIS     :      R1 ARG5 (EXPR . ((IDIFFERENCE P Q)))
ERGEBNIS     :      R1 ARG6 (EXPR . (M))
ERGEBNIS     :      R1 ARG7 (EXPR . (TRUE))
ERGEBNIS     :      R1 ARGLIST (ARG1 ARG2 ARG3 ARG4
                                ARG5 ARG6 ARG7)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-52 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (OL))
ERGEBNIS     :      R2 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R2 ARG3 (EXPR . ((IPLUS I Q)))
ERGEBNIS     :      R2 ARG4 (EXPR . ((IDIFFERENCE P Q)))
ERGEBNIS     :      R2 ARG5 (EXPR . ((ITIMES 2
                                      (IDIFFERENCE Q
                                                   P))))
ERGEBNIS     :      R2 ARG6 (EXPR . (M))
ERGEBNIS     :      R2 ARG7 (EXPR . (TRUE))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2 ARG3 ARG4
                                ARG5 ARG6 ARG7)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3

REGELANWENDUNG:  S4-52 AUF R3

REGELANWENDUNG:  S4-53 AUF R3

REGELANWENDUNG:  S4-54 AUF R3
```



```
REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . (KEY))
ERGEBNIS     :      R3 ARG3 (EXPR . ((IDIFFERENCE I
                                      (IDIFFERENCE M
                                                   Q))))
ERGEBNIS     :      R3 ARG4 (EXPR . ((IDIFFERENCE P Q)))
ERGEBNIS     :      R3 ARG5 (EXPR . ((IDIFFERENCE (ITIMES 2 Q)
                                                  P)))
ERGEBNIS     :      R3 ARG6 (EXPR . (M))
ERGEBNIS     :      R3 ARG7 (EXPR . (TRUE))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3 ARG4
                                ARG5 ARG6 ARG7)

REGELANWENDUNG:  S4-1 AUF R4

REGELANWENDUNG:  S4-2 AUF R4
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FORM (T)

REGELANWENDUNG:  S4-1 AUF R5

REGELANWENDUNG:  S4-2 AUF R5
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 FORM (NOFORM)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (LISP-AUFRUF IDIFFERENCE)
```

```
REGELANWENDUNG:  S5-5 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (FORM . (SETQ I (IDIFFERENCE I Q)))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG4 (EXPR . (SETQ P Q))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG5

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG5
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG5 (LISP-AUFRUF IDIFFERENCE)
ERGEBNIS     :      R1 ARG5 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R1 , ARGUMENT ARG5
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG5 (FORM . (SETQ Q
                                     (IDIFFERENCE STD1-5
                                                  Q)))
ERGEBNIS     :      R1 KOPIERT-LISP (ARG5 (P STD1-5)(Q Q))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG6 (EXPR . (SETQ M M))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG7
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG7 (EXPR . (SETQ FLAG T))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG7

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3
```



```
REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (LISP-AUFRUF IPLUS)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (FORM . (SETQ I (IPLUS I Q)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG4

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG4
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG4 (LISP-AUFRUF IDIFFERENCE)
ERGEBNIS     :      R2 ARG4 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG4
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG4 (FORM . (SETQ P (IDIFFERENCE P Q)))
ERGEBNIS     :      R2 KOPIERT-LISP (ARG4 (P P)(Q Q))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG5

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG5
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG5 (LISP-AUFRUF ITIMES)
ERGEBNIS     :      R2 ARG5 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG5
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG5 (FORM . (SETQ Q
                                     (ITIMES 2
                                             (IDIFFERENCE Q
                                                          STD3-5))))
ERGEBNIS     :      R2 KOPIERT-LISP (ARG5 (2 2)(Q Q)(P STD3-5))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG6 (EXPR . (SETQ M M))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG7
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG7 (EXPR . (SETQ FLAG T))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG7

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM .)
```

```
REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (EXPR . (SETQ KEY KEY))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (LISP-AUFRUF IDIFFERENCE)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM . (SETQ I
                                    (IDIFFERENCE I
                                                 (IDIFFERENCE M
                                                              Q))))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG4

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (LISP-AUFRUF IDIFFERENCE)
ERGEBNIS     :      R3 ARG4 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (FORM . (SETQ P (IDIFFERENCE P Q)))
ERGEBNIS     :      R3 KOPIERT-LISP (ARG4 (P P)(Q Q))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG5

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG5 (LISP-AUFRUF IDIFFERENCE)
ERGEBNIS     :      R3 ARG5 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG5 (FORM . (SETQ Q
                                    (IDIFFERENCE
                                                 (ITIMES 2 Q)
                                                 STD4-5)))
ERGEBNIS     :      R3 KOPIERT-LISP (ARG5 (2 2)(Q Q)(P STD4-5))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG6 (EXPR . (SETQ M M))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG6 (FORM .)
```



```
REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG7
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG7 (EXPR . (SETQ FLAG T))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG7
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL ((DUPLIZIERE P (STD1-5))
                              (SETQ I (IDIFFERENCE I Q))
                              (SETQ P Q)
                              (SETQ Q (IDIFFERENCE STD1-5 Q))
                              (SETQ FLAG T)
                              (FIBO-SEARCH OL KEY
                                           I P Q M FLAG))

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE P (STD3-5))
                              (SETQ I (IPLUS I Q))
                              (SETQ P (IDIFFERENCE P Q))
                              (SETQ Q
                                    (ITIMES 2
                                            (IDIFFERENCE Q
                                                         STD3-5)))
                              (SETQ FLAG T)
                              (FIBO-SEARCH OL KEY
                                           I P Q M FLAG))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((DUPLIZIERE P (STD4-5))
                              (SETQ I
                                    (IDIFFERENCE I
                                                 (IDIFFERENCE M
                                                              Q)))
                              (SETQ P (IDIFFERENCE P Q))
                              (SETQ Q (IDIFFERENCE (ITIMES 2 Q)
                                                   STD4-5))
                              (SETQ FLAG T)
                              (FIBO-SEARCH OL KEY
                                           I P Q M FLAG))

REGELANWENDUNG:  S6-1 AUF R4

REGELANWENDUNG:  S6-21 AUF R4
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FINAL (T)

REGELANWENDUNG:  S6-1 AUF R5
```

```
REGELANWENDUNG:  S6-21 AUF R5
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :       R5 FINAL ('NIL)

START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((AND (GREATERP (OL LIES I)
                      KEY)
             (NEQ Q 0))
        (DUPLIZIERE P (STD1-5))
        (SETQ I (IDIFFERENCE I Q))
        (SETQ P Q)
        (SETQ Q (IDIFFERENCE STD1-5 Q))
        (SETQ FLAG T)
        (FIBO-SEARCH OL KEY I P Q M FLAG))
      ((AND (LESSP (OL LIES I)
                   KEY)
             (NEQ P 1)
             (EQ FLAG T))
        (DUPLIZIERE P (STD3-5))
        (SETQ I (IPLUS I Q))
        (SETQ P (IDIFFERENCE P Q))
        (SETQ Q (ITIMES 2 (IDIFFERENCE Q STD3-5)))
        (SETQ FLAG T)
        (FIBO-SEARCH OL KEY I P Q M FLAG))
      ((AND (LESSP (OL LIES I)
                   KEY)
             (NEQ P 1)
             (EQ FLAG))
        (DUPLIZIERE P (STD4-5))
        (SETQ I (IDIFFERENCE I (IDIFFERENCE M Q)))
        (SETQ P (IDIFFERENCE P Q))
        (SETQ Q (IDIFFERENCE (ITIMES 2 Q)
                             STD4-5))
        (SETQ FLAG T)
        (FIBO-SEARCH OL KEY I P Q M FLAG))
      ((EQ (OL LIES I)
           KEY)
        T)
      ((OR (EQ Q 0)
           (EQ P 1))
        NIL)
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S1-2 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9
 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.680328

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(GREATERP NEQ LESSP EQ DUPLIZIERE)
```



```
SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET

START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<FIBO-SEARCH
 <LAMBDA (OL KEY I P Q M FLAG)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
     (COND
      ((OR (TYPEP KEY 22)
           (TYPEP KEY 23))
        (COND
         ((GETD (QUOTE DUPLIZIERE))
           (COND
            ((GETD (QUOTE LISTSIZE))
             (COND
               ((GETD (QUOTE FIBO-ZAHL))
                 (SETQ I (FIBO-ZAHL 1 (LISTSIZE OL)))
                 (SETQ P (FIBO-ZAHL 2 (LISTSIZE OL)))
                 (SETQ Q (FIBO-ZAHL 3 (LISTSIZE OL)))
                 (SETQ M (FIBO-ZAHL 4 (LISTSIZE OL)))
                 (SETQ FLAG NIL)
                 <RPLACA (QUOTE FIBO-SEARCH)
                         (DSUBST OL (QUOTE OL)
                           (DSUBST
                             <QUOTE
                               (RETEVAL
                                 (STKPOS
                                   (QUOTE FIBO-SEARCH)
                                   -1)
                                 (QUOTE (APPLY FIBO-SEARCH
                                                NIL>
                             (QUOTE (FIBO-SEARCH OL KEY
                                                 I P Q M FLAG))
                             (COPY (GETP (QUOTE FIBO-SEARCH)
                                         (QUOTE ALGORITHMUS>
                   (APPLY FIBO-SEARCH NIL))
                 (T (PRITNL "***ERROR*** HILFSALGORITHMUS"
                            (QUOTE FIBO-ZAHL)
                            "UNDEFINIERT")
                    (BREAK1 NIL T)
                    NIL)))
               (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                          (QUOTE LISTSIZE)
                          "UNDEFINIERT")
                  (BREAK1 NIL T)
                  NIL)))
           (T (PRINTL "***ERROR*** HILFSFUNKTION"
                      (QUOTE DUPLIZIERE)
```

```
                    "UNDEFINIERT")
              (BREAK1 NIL T)
              NIL)))
      (T (PRINTL "***ERROR***" KEY "NICHT VOM TYP INTEGER")
         (BREAK1 NIL T)
         NIL)))
   (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
              (QUOTE OFFENELISTE-INTEGER))
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET
```

Folgende Bindungen bestehen nach dem Implementierungslauf für das Atom FIBO-SEARCH:

```
(DEFINEQ

<FIBO-SEARCH
  <LAMBDA (OL KEY I P Q M FLAG)
          .
          .
          .
     siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS FIBO-SEARCH KOMMENTAR
            (FIBO-SEARCH SUCHT IN DER OFFENENLISTE OL NACH
             DEM SCHLUESSEL %' KEY %' %. DABEI WIRD DAS
             SUCHINTERVALL SUKZESSIVE GEMAESS DEM
             FIBONACCI-VERFAHREN EINGESCHRAENKT.
             ERGEBNIS IST EIN WAHRHEITSWERT.
             [KNU 73 , P.415])
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                .
                .
                .
     siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```


4.4. Sortieralgorithmen

Während bei den Suchalgorithmen die Eingabe-Datenstrukturen, in der nach einem Schlüssel gesucht werden soll, erhalten bleiben, wird diese bei den Sortieralgorithmen immer verändert. Sollen nach Anwendung eines Sortieralgorithmus die Eingabestruktur weiterhin in der ursprünglichen Form zur Verfühgung stehen, muß diese vorher durch Aufruf der Funktion DUPLIZIERE (siehe 5.3.2.,S. 505) kopiert werden. Ein normales Kopieren, z.B. mittels der INTERLISP-Funktion COPY, ist nicht ausreichend, da hierbei nur der WERT der Eingabestruktur, nicht aber, wie bei DUPLIZIERE, auch die Bindungen an der Property-List und der Funktionszelle kopiert werden.

Die Sortieralgorithmen arbeiten entweder auf einer Datentyp-Inkarnation, wobei dann direkt auf dieser Inkarnation gearbeitet wird (bei Bubble-Sort und Quick-Sort), oder auf mehreren Inkarnationen (Merge-Sort, Straight-Insertion-Sort und Straight-Selection-Sort). Im letzten Fall wird die Eingabe-Inkarnation (bzw. bei Merge-Sort die Eingabe-Inkarnationen) zerstört und eine Ausgabe-Inkarnation desselben Typs mit dem in der Spezifikation angegebenen Namen erzeugt. Soll auch hier die sortierte Inkarnation unter dem gleichen Namen wie vorher zugreifbar sein, kann der Algorithmus-Aufruf z.B. in ein SETQ-Statement eingebettet werden:

```
(SETQ X (STRAIGHT-INSERTION-SORT X))
```

Auch die Sortieralgorithmen arbeiten auf Offenen Listen mit Ausnahme von Merge-Sort, der auf Kellern operiert.
Die Implementierungen im einzelnen finden sich für

Merge-Sort	in 4.4.1.,
Straight-Insertion-Sort	in 4.4.2.,
Straight-Selection-Sort	in 4.4.3. und für
Quick-Sort	in 4.4.4.

Alle Sortieralgorithmen sind [KNU 73] entnommen.

4.4.1. Merge-Sort

Dieser Sortieralgorithmus arbeitet auf zwei Integer-Kellern A und B, die als Parameter beim Aufruf übergeben werden. Ergebnis von Merge-Sort ist eine neu initialisierte KELLER-INTEGER-Inkarnation, die die Elemente von A und B sortiert enthält. Voraussetzung ist allerdings, daß die Eingabe-Keller A und B schon absteigend sortiert sind, d.h. daß das jeweils größte Element von A und B das TOP-Element ist.
Merge-Sort vergleicht die beiden TOP-Elemente von A und B, pusht das größere auf den Ergebnis-Keller C und popt es aus dem entsprechenden Eingabe-Keller.
Nach Ausführung von Merge-Sort sind die Keller A und leer; sollen sie weiterhin unverändert zur Verfügung stehen, muß man sie vorher mittels DUPLIZIERE kopieren (s.o.).

Hier die Spezifikation für Merge-Sort:

```
ALGORITHMUS-NAME: MERGE-SORT;

PARAMETERTYPEN:    BENUTZER: KELLER-INTEGER  A,
                             KELLER-INTEGER  B;
                   SYSTEM:   KELLER-INTEGER  C :=
                                    (INIT C KELLER-INTEGER);

RELATION:          UNDEF-KL-INTEGER;

ALGORITHMUS:       MERGE-SORT (A,B,C) =
             {      (A TOP) > (B TOP)
              ---> MERGE-SORT ((A POP),B,(C PUSH (A TOP)));
                    (A TOP) < (B TOP)
              ---> MERGE-SORT (A,(B POP),(C PUSH (B TOP)));
                    (A TOP) = (B TOP)   UND  (A TOP) /= UNDEF
                                        UND  (B TOP) /= UNDEF
              ---> MERGE-SORT ((A POP),
                               (B POP),
                               (C PUSH (A TOP)));
                    (A TOP) = UNDEF   UND   (B TOP) = UNDEF
              ---> C    }

KOMMENTAR: MERGE-SORT LIEST DIE ABSTEIGEND SORTIERTEN, MIT
           GANZEN ZAHLEN GEFUELLTEN KELLER A UND B. WERT VON
           MERGE-SORT IST DER AUS DEN ELEMENTEN VON A UND B
           AUFSTEIGEND SORTIERTE KELLER C.
           A UND B WERDEN GELEERT [KNU 73,P.160]!!
```


Der System-Parameter C wird als KELLER-INTEGER initialisiert; er nimmt die Elemente der Eingabe-Keller A und B geordnet, ohne Doppelspeicherung, auf. Die Wertzuweisung an C wird durch die Funktion FKT-SYMBOL-IN innerhalb von RALGSPEC überprüft.
Als Relation wird UNDEF-KL-INTEGER (siehe 5.3.2.,S. 509) vereinbart. Dieser Relationssatz arbeitet auf dem Datentyp Integer, wobei die ganzen Zahlen mit dem Element UNDEF vereinigt werden. UNDEF ist per Definition kleiner als jede ganze Zahl. Auf diese Weise lassen sich die Regeln von Merge-Sort einfacher formulieren, da der Fall, daß ein Argument UNDEF ist, nicht explizit berücksichtigt werden muß.

Es folgt das Implementierungsprotokoll:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: MER
ALGORITHMUS-NAME: (MERGE-SORT)
PARAMETERTYPEN:    BENUTZER: ((A . KELLER-INTEGER)
                              (B . KELLER-INTEGER))
                   SYSTEM:   ((C KELLER-INTEGER
                                (INIT C KELLER-INTEGER)))
RELATION:          (UNDEF-KL-INTEGER)
RELATIONSSATZ:      ((== . GLEICHP)(/= . UNGLEICHP)
                     (< . KLEINERP)(> . GROESSERP)
                     (<= . KLEINERGLEICHP)
                     (>= . GROESSERGLEICHP))
ALGORITHMUS:       PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((A TOP) > (B TOP))))
R1 : ((MERGE-SORT (((A POP)) , (B) , ((C PUSH (A TOP))))))
L2 : ((((A TOP) < (B TOP))))
R2 : ((MERGE-SORT ((A) , ((B POP)) , ((C PUSH (B TOP))))))
L3 : ((((A TOP) == (B TOP)) UND ((A TOP) /= UNDEF)
                            UND ((B TOP) /= UNDEF)))
R3 : ((MERGE-SORT (((A POP)) ,
                   ((B POP)) ,
                   ((C PUSH (A TOP))))))
L4 : ((((A TOP) == UNDEF) UND ((B TOP) == UNDEF)))
R4 : ((C))
LINKESEITEN:   (L1 L2 L3 L4)
RECHTESEITEN:  (R1 R2 R3 R4)
KOMMENTAR:         (MERGE-SORT LIEST DIE ABSTEIGEND SORTIERTEN
                    , MIT GANZEN ZAHLEN GEFUELLTEN KELLER A
                    UND B. WERT VON MERGE-SORT IST DER AUS DEN
                    ELEMENTEN VON A UND B AUFSTEIGEND SORTIERTE
                    KELLER C. A UND B WERDEN GELEERT [KNU 73 ,
                    P.160])
ALG-EINGABE BEENDET.
EINLESE-PHASE BEENDET
```

```
START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((A TOP) > (B TOP)))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((A TOP) < (B TOP)))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((A TOP) == (B TOP)))
ERGEBNIS     :      L3 VGL2 (EXPR . ((A TOP) /= UNDEF))
ERGEBNIS     :      L3 VGL3 (EXPR . ((B TOP) /= UNDEF))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((A TOP) == UNDEF))
ERGEBNIS     :      L4 VGL2 (EXPR . ((B TOP) == UNDEF))
ERGEBNIS     :      L4 LOGOP (AND)
ERGEBNIS     :      L4 VGLLIST (VGL1 VGL2)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (A TOP))
ERGEBNIS     :      L1 VGL1 (RECHTS . (B TOP))
ERGEBNIS     :      L1 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (GROESSERP (A TOP)(B TOP)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (A TOP))
ERGEBNIS     :      L2 VGL1 (RECHTS . (B TOP))
ERGEBNIS     :      L2 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (KLEINERP (A TOP)(B TOP)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
```



```
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (A TOP))
ERGEBNIS     :      L3 VGL1 (RECHTS . (B TOP))
ERGEBNIS     :      L3 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (GLEICHP (A TOP)(B TOP)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (LINKS . (A TOP))
ERGEBNIS     :      L3 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (FORM . (UNGLEICHP (A TOP)
                                               (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL3
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL3 (LINKS . (B TOP))
ERGEBNIS     :      L3 VGL3 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL3 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL3
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL3 (FORM . (UNGLEICHP (B TOP)
                                               (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (LINKS . (A TOP))
ERGEBNIS     :      L4 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L4 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (FORM . (GLEICHP (A TOP)
                                             (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (LINKS . (B TOP))
ERGEBNIS     :      L4 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L4 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (FORM . (GLEICHP (B TOP)
                                             (QUOTE UNDEF)))
SCHRITT-2 ENDE
```

```
SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((GROESSERP (A TOP)(B TOP)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((KLEINERP (A TOP)(B TOP)))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((AND (GLEICHP (A TOP)(B TOP))
                                   (UNGLEICHP (A TOP)
                                              (QUOTE UNDEF))
                                   (UNGLEICHP (B TOP)
                                              (QUOTE UNDEF))))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 FINAL ((AND (GLEICHP (A TOP)
                                            (QUOTE UNDEF))
                                   (GLEICHP (B TOP)
                                            (QUOTE UNDEF))))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R1
---REGEL     : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 KOPIERT3 (A)

REGELANWENDUNG:  S4-9 AUF R1
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . ((A POP)))
ERGEBNIS     :      R1 ARG2 (EXPR . (B))
ERGEBNIS     :      R1 ARG3 (EXPR . ((C PUSH (A TOP))))
ERGEBNIS     :      R1 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2
---REGEL     : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 KOPIERT3 (B)

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (A))
ERGEBNIS     :      R2 ARG2 (EXPR . ((B POP)))
```



```
ERGEBNIS     :      R2 ARG3 (EXPR . ((C PUSH (B TOP))))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3
---REGEL     : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 KOPIERT3 (A)

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . ((A POP)))
ERGEBNIS     :      R3 ARG2 (EXPR . ((B POP)))
ERGEBNIS     :      R3 ARG3 (EXPR . ((C PUSH (A TOP))))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R4

REGELANWENDUNG:  S4-2 AUF R4
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FORM (C)
ERGEBNIS     :      R4 EVALUIERT (PARAMPASS)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (EXPR . (SETQ A (A POP)))

REGELANWENDUNG:  S5-8 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG1 (FORM . (A POP))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (EXPR . (SETQ B B))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG3

REGELANWENDUNG:  S5-31 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (EXPR . (SETQ C (C PUSH (X3 TOP))))
```

```
REGELANWENDUNG:  S5-8 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 ARG3 (FORM . (C PUSH (X3 TOP)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ A A))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (EXPR . (SETQ B (B POP)))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM . (B POP))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (EXPR . (SETQ C (C PUSH (X3 TOP))))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-8 ERFOLGREICH ANGEWENDET AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (FORM . (C PUSH (X3 TOP)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ A (A POP)))

REGELANWENDUNG:  S5-8 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM . (A POP))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-31 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (EXPR . (SETQ B (B POP)))
```



```
REGELANWENDUNG:  S5-8 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM . (B POP))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-31 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (EXPR . (SETQ C (C PUSH (X3 TOP))))

REGELANWENDUNG:  S5-8 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-8 ERFOLGREICH ANGEWENDET AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM . (C PUSH (X3 TOP)))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL ((DUPLIZIERE A X3)
                              (A POP)
                              (C PUSH (X3 TOP))
                              (MERGE-SORT A B C))

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE B X3)
                              (B POP)
                              (C PUSH (X3 TOP))
                              (MERGE-SORT A B C))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((DUPLIZIERE A X3)
                              (A POP)
                              (B POP)
                              (C PUSH (X3 TOP))
                              (MERGE-SORT A B C))

REGELANWENDUNG:  S6-1 AUF R4

REGELANWENDUNG:  S6-21 AUF R4
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FINAL (C)
```

```
START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((GROESSERP (A TOP)
                  (B TOP))
        (DUPLIZIERE A X3)
        (A POP)
        (C PUSH (X3 TOP))
        (MERGE-SORT A B C))
      ((KLEINERP (A TOP)
                 (B TOP))
        (DUPLIZIERE B X3)
        (B POP)
        (C PUSH (X3 TOP))
        (MERGE-SORT A B C))
      ((AND (GLEICHP (A TOP)
                     (B TOP))
            (UNGLEICHP (A TOP)
                       (QUOTE UNDEF))
            (UNGLEICHP (B TOP)
                       (QUOTE UNDEF)))
        (DUPLIZIERE A X3)
        (A POP)
        (B POP)
        (C PUSH (X3 TOP))
        (MERGE-SORT A B C))
      ((AND (GLEICHP (A TOP)
                     (QUOTE UNDEF))
            (GLEICHP (B TOP)
                     (QUOTE UNDEF)))
        C)
      (T (FEHLER 10>

 VERWENDET WURDEN DIE REGELN
 (S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9
  S5-1 S5-31 S5-8 S5-9 S6-1 S6-21)

 REGELANWENDUNGSFAKTOR:  0.698630

 ERKANNT WURDEN DIE HILFSFUNKTIONEN
 (GROESSERP KLEINERP GLEICHP UNGLEICHP DUPLIZIERE)

 SCHRITT-6 ENDE

 ALG-KONSTRUKTIONS-PHASE BEENDET
```



```
START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<MERGE-SORT
 <LAMBDA (A B C)
  (COND
   ((EQ (QUOTE KELLER-INTEGER)
        (GETP A (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((EQ (QUOTE KELLER-INTEGER)
          (GETP B (QUOTE PASSIVSTRUKTUR)))
      (COND
       ((GETD (QUOTE DUPLIZIERE))
        (COND
         ((GETD (QUOTE UNGLEICHP))
          (COND
           ((GETD (QUOTE GLEICHP))
            (COND
             ((GETD (QUOTE KLEINERP))
              (COND
               ((GETD (QUOTE GROESSERP))
                (INIT C KELLER-INTEGER)
                <RPLACA (QUOTE MERGE-SORT)
                  (DSUBST B (QUOTE B)
                    (DSUBST A (QUOTE A)
                      (DSUBST
                        <QUOTE
                          (RETEVAL
                            (STKPOS
                              (QUOTE MERGE-SORT)
                              -1)
                            (QUOTE (APPLY MERGE-SORT
                                          NIL>
                        (QUOTE (MERGE-SORT A B C))
                        (COPY (GETP (QUOTE MERGE-SORT)
                                    (QUOTE ALGORITHMUS>
                (APPLY MERGE-SORT NIL))
               (T (PRINTL "***ERROR*** HILFSFUNKTION"
                          (QUOTE GROESSERP)
                          "UNDEFINIERT")
                  (BREAK1 NIL T)
                  NIL)))
              (T (PRINTL "***ERROR*** HILFSFUNKTION"
                         (QUOTE KLEINERP)
                         "UNDEFINIERT")
                 (BREAK1 NIL T)
                 NIL)))
            (T (PRINTL "***ERROR*** HILFSFUNKTION"
                       (QUOTE GLEICHP)
                       "UNDEFINIERT")
               (BREAK1 NIL T)
               NIL)))
          (T (PRINTL "***ERROR*** HILFSFUNKTION"
                     (QUOTE UNGLEICHP)
                     "UNDEFINIERT")
```

```
              (BREAK1 NIL T)
              NIL)))
        (T (PRINTL "***ERROR*** HILFSFUNKTION"
                   (QUOTE DUPLIZIERE)
                   "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
     (T (PRINTL "***ERROR***" B "NICHT VOM TYP"
        (QUOTE KELLER-INTEGER))
        (BREAK1 NIL T)
        NIL)))
   (T (PRINTL "***ERROR***" A "NICHT VOM TYP"
      (QUOTE KELLER-INTEGER))
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET

(DEFINEQ

<MERGE-SORT
  <LAMBDA (A B C)
          .
          .
          .
     siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS MERGE-SORT KOMMENTAR
          (MERGE-SORT LIEST DIE ABSTEIGEND SORTIERTEN ,
           MIT GANZEN ZAHLEN GEFUELLTEN KELLER A UND B.
           WERT VON MERGE-SORT IST DER AUS DEN ELEMENTEN
           VON A UND B AUFSTEIGEND SORTIERTE KELLER C. A
           UND B WERDEN GELEERT [KNU 73 , P.160])
          RELATION
          (UNDEF-KL-INTEGER ((== . GLEICHP)
                             (/= . UNGLEICHP)
                             (%< . KLEINERP)
                             (%> . GROESSERP)
                             (%<= . KLEINERGLEICHP)
                             (%>= . GROESSERGLEICHP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
                  .
                  .
                  .
     siehe Ergebnis der Automatischen Codierung
                  .
                  .
                  .    >
STOP
```


4.4.2. Straight-Insertion-Sort

Straight-Insertion-Sort liest das jeweils erste Element der OFFENENLISTE OL1, sortiert es an der richtigen Stelle in die Ergebnisliste OL2, eine neu erzeugte OFFENELISTE-Inkarnation, ein, und entfernt es aus OL1.

Es werden wieder OFFENELISTEN verwendet, da beim Einsortieren in OL2 ein Zugriff auf eine beliebige Stelle der Liste möglich sein muß.
Straight-Insertion-Sort zerstört die Eingabe-Liste OL1. Soll diese jedoch erhalten bleiben, so muß mit von DUPLIZIERE erzeugten Kopien gearbeitet werden (s.o.).

Die Spezifikation für Straight-Insertion-Sort sieht folgendermaßen aus:

```
ALGORITHMUS-NAME: STRAIGHT-INSERTION-SORT;

PARAMETERTYPEN:    BENUTZER: OFFENELISTE-INTEGER  OL1;
                   SYSTEM:   OFFENELISTE-INTEGER  OL2 :=
                             (INIT OL2 OFFENELISTE-INTEGER),
                             INTEGER  INDEX := 1;

ALGORITHMUS:       STRAIGHT-INSERTION-SORT (OL1,OL2,INDEX) =
              {     (OL2 LIES INDEX) = UNDEF   UND
                    (OL1 LIES 1) /= UNDEF
               ---> STRAIGHT-INSERTION-SORT
                             ((OL1 ENTFERN 1),
                              (OL2 HINZU INDEX (OL1 LIES 1)),
                              1);
                    (OL2 LIES INDEX) /= UNDEF   UND
                    (OL1 LIES 1) /= UNDEF       UND
                    (OL2 LIES INDEX) > (OL1 LIES 1)
               ---> STRAIGHT-INSERTION-SORT
                             ((OL1 ENTFERN 1),
                              (OL2 HINZU INDEX (OL1 LIES 1)),
                              1);
                    (OL2 LIES INDEX) /= UNDEF   UND
                    (OL1 LIES 1) /= UNDEF       UND
                    (OL2 LIES INDEX) < (OL1 LIES 1)
               ---> STRAIGHT-INSERTION-SORT
                             (OL1,OL2,(ADD1 INDEX));
                    (OL2 LIES INDEX) /= UNDEF   UND
                    (OL1 LIES 1) /= UNDEF       UND
                    (OL2 LIES INDEX) = (OL1 LIES 1)
               ---> STRAIGHT-INSERTION-SORT
                             ((OL1 ENTFERN 1),OL2,1);
                    (OL1 LIES 1) = UNDEF
               ---> OL2     }

KOMMENTAR: STRAIGHT-INSERTION-SORT LIEST DIE MIT GANZEN ZAH-
           LEN GEFUELLTE OFFENELISTE OL1 VON VORNE UND
           SCHREIBT DIE ZAHLEN RICHTIG EINSORTIERT IN DIE
           OFFENELISTE OL2, DIE WERT VON STRAIGHT-INSERTION-
```

SORT IST. OL1 WIRD ZERSTOERT. [KNU 73,P.81]!!

```
Es folgt das Implementierungsprotokoll:
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: SIS
ALGORITHMUS-NAME: (STRAIGHT-INSERTION-SORT)
PARAMETERTYPEN:   BENUTZER: ((OL1 . OFFENELISTE-INTEGER))
                  SYSTEM:   ((OL2 OFFENELISTE-INTEGER
                                  (INIT OL2
                                        OFFENELISTE-INTEGER))
                            (INDEX INTEGER 1))
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL2 LIES INDEX) == UNDEF) UND
       ((OL1 LIES 1) /= UNDEF)))
R1 : ((STRAIGHT-INSERTION-SORT (((OL1 ENTFERN 1)) ,
                                ((OL2 HINZU INDEX
                                      (OL1 LIES 1))) ,
                                (1))))
L2 : ((((OL2 LIES INDEX) /= UNDEF) UND
       ((OL1 LIES 1) /= UNDEF) UND
       ((OL2 LIES INDEX) > (OL1 LIES 1))))
R2 : ((STRAIGHT-INSERTION-SORT (((OL1 ENTFERN 1)) ,
                                ((OL2 HINZU INDEX
                                      (OL1 LIES 1))) ,
                                (1))))
L3 : ((((OL2 LIES INDEX) /= UNDEF) UND
       ((OL1 LIES 1) /= UNDEF) UND
       ((OL2 LIES INDEX) < (OL1 LIES 1))))
R3 : ((STRAIGHT-INSERTION-SORT ((OL1) ,
                                (OL2) ,
                                ((ADD1 INDEX)))))
L4 : ((((OL2 LIES INDEX) /= UNDEF) UND
       ((OL1 LIES 1) /= UNDEF) UND
       ((OL2 LIES INDEX) == (OL1 LIES 1))))
R4 : ((STRAIGHT-INSERTION-SORT (((OL1 ENTFERN 1)) ,
                                (OL2) ,
                                (1))))
L5 : ((((OL1 LIES 1) == UNDEF)))
R5 : ((OL2))
LINKESEITEN:   (L1 L2 L3 L4 L5)
RECHTESEITEN:  (R1 R2 R3 R4 R5)
KOMMENTAR:        (STRAIGHT-INSERTION-SORT LIEST DIE MIT
                   GANZEN ZAHLEN GEFUELLTE OFFENELISTE OL1 VON
                   VORNE UND SCHREIBT DIE ZAHLEN RICHTIG
                   EINSORTIERT IN DIE OFFENELISTE OL2 , DIE
                   WERT VON STRAIGHT-INSERTION-SORT IST. OL1
                   WIRD ZERSTOERT. [KNU 73 , P.81])
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET
```



```
START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL2 LIES INDEX)
                                     == UNDEF))
ERGEBNIS     :      L1 LOGOP (AND)
ERGEBNIS     :      L1 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL2 LIES INDEX)
                                     /= UNDEF))
ERGEBNIS     :      L2 VGL2 (EXPR . ((OL1 LIES 1) /= UNDEF))
ERGEBNIS     :      L2 VGL3 (EXPR . ((OL2 LIES INDEX)
                                     > (OL1 LIES 1)))
ERGEBNIS     :      L2 LOGOP (AND)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL2 LIES INDEX)
                                     /= UNDEF))
ERGEBNIS     :      L3 VGL2 (EXPR . ((OL1 LIES 1) /= UNDEF))
ERGEBNIS     :      L3 VGL3 (EXPR . ((OL2 LIES INDEX)
                                     < (OL1 LIES 1)))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((OL2 LIES INDEX)
                                     /= UNDEF))
ERGEBNIS     :      L4 VGL2 (EXPR . ((OL1 LIES 1) /= UNDEF))
ERGEBNIS     :      L4 VGL3 (EXPR . ((OL2 LIES INDEX)
                                     == (OL1 LIES 1)))
ERGEBNIS     :      L4 LOGOP (AND)
ERGEBNIS     :      L4 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L5
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS     :      L5 VGL1 (EXPR . ((OL1 LIES 1) == UNDEF))
ERGEBNIS     :      L5 LOGOP (NONE)
ERGEBNIS     :      L5 VGLLIST (VGL1)
SCHRITT-1 ENDE
```

```
SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L1 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ (OL2 LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L1 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL2 (FORM . (NEQ (OL1 LIES 1)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L2 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L2 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (NEQ (OL2 LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L2 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L2 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL2 (FORM . (NEQ (OL1 LIES 1)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL3
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL3 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L2 VGL3 (RECHTS . (OL1 LIES 1))
ERGEBNIS     :      L2 VGL3 (REL . >)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL3
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL3 (FORM . (GREATERP (OL2 LIES INDEX)
                                              (OL1 LIES 1)))
```



```
REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L3 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (NEQ (OL2 LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L3 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L3 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL2 (FORM . (NEQ (OL1 LIES 1)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL3
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL3 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L3 VGL3 (RECHTS . (OL1 LIES 1))
ERGEBNIS     :      L3 VGL3 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL3
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL3 (FORM . (LESSP (OL2 LIES INDEX)
                                           (OL1 LIES 1)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (LINKS . (OL2 LIES INDEX))
ERGEBNIS     :      L4 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L4 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (FORM . (NEQ (OL2 LIES INDEX)
                                         (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L4 VGL2 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L4 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL2
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL2 (FORM . (NEQ (OL1 LIES 1)
                                         (QUOTE UNDEF)))
```

```
REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL3
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL3 (LINKS . (OL2 LIES INDEX))
ERGEBNIS    :      L4 VGL3 (RECHTS . (OL1 LIES 1))
ERGEBNIS    :      L4 VGL3 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL3 (FORM . (EQ (OL2 LIES INDEX)
                                       (OL1 LIES 1)))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL1 (LINKS . (OL1 LIES 1))
ERGEBNIS    :      L5 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS    :      L5 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL1 (FORM . (EQ (OL1 LIES 1)
                                       (QUOTE UNDEF)))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 FINAL ((AND (EQ (OL2 LIES INDEX)
                                      (QUOTE UNDEF))
                                  (NEQ (OL1 LIES 1)
                                       (QUOTE UNDEF))))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 FINAL ((AND (NEQ (OL2 LIES INDEX)
                                       (QUOTE UNDEF))
                                  (NEQ (OL1 LIES 1)
                                       (QUOTE UNDEF))
                                  (GREATERP (OL2 LIES INDEX)
                                            (OL1 LIES 1))))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 FINAL ((AND (NEQ (OL2 LIES INDEX)
                                       (QUOTE UNDEF))
                                  (NEQ (OL1 LIES 1)
                                       (QUOTE UNDEF))
                                  (LESSP (OL2 LIES INDEX)
                                         (OL1 LIES 1))))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 FINAL ((AND (NEQ (OL2 LIES INDEX)
                                       (QUOTE UNDEF))
                                  (NEQ (OL1 LIES 1)
```



```
                                       (QUOTE UNDEF))
                                  (EQ (OL2 LIES INDEX)
                                      (OL1 LIES 1))))

REGELANWENDUNG:  S3-1 AUF L5
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 FINAL ((EQ (OL1 LIES 1)(QUOTE UNDEF)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R1
---REGEL    : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 KOPIERT2 (OL1)

REGELANWENDUNG:  S4-9 AUF R1
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG1 (EXPR . ((OL1 ENTFERN 1)))
ERGEBNIS    :      R1 ARG2 (EXPR . ((OL2 HINZU INDEX
                                        (OL1 LIES 1))))
ERGEBNIS    :      R1 ARG3 (EXPR . (1))
ERGEBNIS    :      R1 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2
---REGEL    : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 KOPIERT2 (OL1)

REGELANWENDUNG:  S4-9 AUF R2
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG1 (EXPR . ((OL1 ENTFERN 1)))
ERGEBNIS    :      R2 ARG2 (EXPR . ((OL2 HINZU INDEX
                                        (OL1 LIES 1))))
ERGEBNIS    :      R2 ARG3 (EXPR . (1))
ERGEBNIS    :      R2 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG1 (EXPR . (OL1))
ERGEBNIS    :      R3 ARG2 (EXPR . (OL2))
```

```
ERGEBNIS     :       R3 ARG3 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :       R3 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R4
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R4

REGELANWENDUNG:  S4-55 AUF R4

REGELANWENDUNG:  S4-9 AUF R4
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 ARG1 (EXPR . ((OL1 ENTFERN 1)))
ERGEBNIS     :       R4 ARG2 (EXPR . (OL2))
ERGEBNIS     :       R4 ARG3 (EXPR . (1))
ERGEBNIS     :       R4 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R5

REGELANWENDUNG:  S4-2 AUF R5
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :       R5 FORM (OL2)
ERGEBNIS     :       R5 EVALUIERT (PARAMPASS)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG1 (EXPR . (SETQ OL1 (OL1 ENTFERN 1)))

REGELANWENDUNG:  S5-8 AUF R1 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG1 (FORM . (OL1 ENTFERN 1))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG2

REGELANWENDUNG:  S5-31 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG2 (EXPR . (SETQ OL2
                                      (OL2 HINZU
                                       INDEX
                                       (X2 LIES 1))))

REGELANWENDUNG:  S5-8 AUF R1 , ARGUMENT ARG2
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG2 (FORM . (OL2 HINZU INDEX
                                       (X2 LIES 1)))
```



```
REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 ARG3 (EXPR . (SETQ INDEX 1))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (EXPR . (SETQ OL1 (OL1 ENTFERN 1)))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (FORM . (OL1 ENTFERN 1))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (EXPR . (SETQ OL2
                                      (OL2 HINZU
                                       INDEX
                                       (X2 LIES 1))))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (FORM . (OL2 HINZU INDEX
                                       (X2 LIES 1)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG3 (EXPR . (SETQ INDEX 1))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (SETQ OL1 OL1))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (EXPR . (SETQ OL2 OL2))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (FORM .)
```

```
REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R4 , ARGUMENT ARG1
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (EXPR . (SETQ OL1 (OL1 ENTFERN 1)))

REGELANWENDUNG:  S5-8 AUF R4 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (FORM . (OL1 ENTFERN 1))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG2 (EXPR . (SETQ OL2 OL2))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG3 (EXPR . (SETQ INDEX 1))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG3
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL ((DUPLIZIERE OL1 X2)
                              (OL1 ENTFERN 1)
                              (OL2 HINZU INDEX (X2 LIES 1))
                              (SETQ INDEX 1)
                              (STRAIGHT-INSERTION-SORT OL1
                                                       OL2
                                                       INDEX))

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE OL1 X2)
                              (OL1 ENTFERN 1)
                              (OL2 HINZU INDEX (X2 LIES 1))
```



```
                              (SETQ INDEX 1)
                              (STRAIGHT-INSERTION-SORT OL1
                                                       OL2
                                                       INDEX))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ INDEX (ADD1 INDEX))
                              (STRAIGHT-INSERTION-SORT OL1
                                                       OL2
                                                       INDEX))

REGELANWENDUNG:  S6-1 AUF R4
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FINAL ((OL1 ENTFERN 1)
                              (SETQ INDEX 1)
                              (STRAIGHT-INSERTION-SORT OL1
                                                       OL2
                                                       INDEX))

REGELANWENDUNG:  S6-1 AUF R5

REGELANWENDUNG:  S6-21 AUF R5
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 FINAL (OL2)


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((AND (EQ (OL2 LIES INDEX)
                (QUOTE UNDEF))
            (NEQ (OL1 LIES 1)
                 (QUOTE UNDEF)))
        (DUPLIZIERE OL1 X2)
        (OL1 ENTFERN 1)
        (OL2 HINZU INDEX (X2 LIES 1))
        (SETQ INDEX 1)
        (STRAIGHT-INSERTION-SORT OL1 OL2 INDEX))
      ((AND (NEQ (OL2 LIES INDEX)
                 (QUOTE UNDEF))
            (NEQ (OL1 LIES 1)
                 (QUOTE UNDEF))
            (GREATERP (OL2 LIES INDEX)
                      (OL1 LIES 1)))
        (DUPLIZIERE OL1 X2)
        (OL1 ENTFERN 1)
        (OL2 HINZU INDEX (X2 LIES 1))
        (SETQ INDEX 1)
        (STRAIGHT-INSERTION-SORT OL1 OL2 INDEX))
      ((AND (NEQ (OL2 LIES INDEX)
                 (QUOTE UNDEF))
            (NEQ (OL1 LIES 1)
                 (QUOTE UNDEF))
```

```
          (LESSP (OL2 LIES INDEX)
                 (OL1 LIES 1)))
        (SETQ INDEX (ADD1 INDEX))
        (STRAIGHT-INSERTION-SORT OL1 OL2 INDEX))
       ((AND (NEQ (OL2 LIES INDEX)
                  (QUOTE UNDEF))
             (NEQ (OL1 LIES 1)
                  (QUOTE UNDEF))
             (EQ (OL2 LIES INDEX)
                 (OL1 LIES 1)))
        (OL1 ENTFERN 1)
        (SETQ INDEX 1)
        (STRAIGHT-INSERTION-SORT OL1 OL2 INDEX))
       ((EQ (OL1 LIES 1)
            (QUOTE UNDEF))
        OL2)
       (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9
 S5-1 S5-2 S5-31 S5-5 S5-8 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.780220

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ NEQ GREATERP LESSP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<STRAIGHT-INSERTION-SORT
 <LAMBDA (OL1 OL2 INDEX)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL1 (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((GETD (QUOTE DUPLIZIERE))
      (INIT OL2 OFFENELISTE-INTEGER)
      (SETQ INDEX 1)
      <RPLACA (QUOTE STRAIGHT-INSERTION-SORT)
        (DSUBST OL1 (QUOTE OL1)
          (DSUBST
            <QUOTE
              (RETEVAL
                (STKPOS
                  (QUOTE STRAIGHT-INSERTION-SORT)
                  -1)
                (QUOTE (APPLY STRAIGHT-INSERTION-SORT NIL>
            (QUOTE (STRAIGHT-INSERTION-SORT OL1 OL2 INDEX))
            (COPY (GETP (QUOTE STRAIGHT-INSERTION-SORT)
```



```
                          (QUOTE ALGORITHMUS>
        (APPLY STRAIGHT-INSERTION-SORT NIL))
       (T (PRINTL "***ERROR*** HILFSFUNKTION"
                  (QUOTE DUPLIZIERE)
                  "UNDEFINIERT")
          (BREAK1 NIL T)
          NIL)))
    (T (PRINTL "***ERROR***" OL1 "NICHT VOM TYP"
               (QUOTE OFFENELISTE-INTEGER))
       (BREAK1 NIL T)
       NIL>>
LISP-PHASE BEENDET


(DEFINEQ

<STRAIGHT-INSERTION-SORT
  <LAMBDA (OL1 OL2 INDEX)
            .
            .
            .
      siehe Ergebnis der LISP-Phase
            .
            .
            .    >>
)
  <PUTPROPS STRAIGHT-INSERTION-SORT KOMMENTAR
                                    (STRAIGHT-INSERTION-SORT
                                     LIEST DIE MIT GANZEN
                                     ZAHLEN GEFUELLTE
                                     OFFENELISTE OL1 VON
                                     VORNE UND SCHREIBT DIE
                                     ZAHLEN RICHTIG
                                     EINSORTIERT IN DIE
                                     OFFENELISTE OL2 , DIE
                                     WERT VON
                                     STRAIGHT-INSERTION-SORT
                                     IST. OL1 WIRD ZERSTOERT.
                                     [KNU 73 , p.81])
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                .
                .
                .
      siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
```

STOP

4.4.3. Straight-Selection-Sort

Straight-Selection-Sort wählt mittels des Hilfsalgorithmus MAXLIST das jeweils größte Element der Eingabe-Liste OL1 aus und schreibt es an die erste Position der neu erzeugten Ausgabe-Liste OL2. Dieses Element wird anschließend aus OL1 entfernt.
Straight-Selection-Sort muß auf beliebige Elemente der Eingabe-Liste zugreifen, weshalb auch hier der Datentyp Offene Liste gewählt wurde.
Nach Ausführung des Algorithmus ist die Eingabe-Liste zerstört. Soll sie erhalten bleiben, muß auf einer von DUPLIZIERE erzeugten Kopie gearbeitet werden (s.o.).
Die Hilfsalgorithmen MAXLIST und INDMAX sind in 4.5.2 bzw. 4.5.3. beschrieben.

Zunächst die Spezifikation für Straight-Selection-Sort:

```
ALGORITHMUS-NAME: STRAIGHT-SELECTION-SORT;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-INTEGER  OL1;
                  SYSTEM:   OFFENELISTE-INTEGER  OL2 :=
                            (INIT OL2 OFFENELISTE-INTEGER);

HILFSALGORITHMEN: MAXLIST,
                  INDMAX;

ALGORITHMUS:      STRAIGHT-SELECTION-SORT (OL1,OL2) =
            {       (OL1 LIES 1) = UNDEF
             ---->  OL2;
                    (OL1 LIES 1) /= UNDEF
             ---->  STRAIGHT-SELECTION-SORT
                            ((OL1 ENTFERN INDMAX ('OL1)) ,
                             (OL2 HINZU 1 MAXLIST ('OL1))) }

KOMMENTAR: STRAIGHT-SELECTION-SORT LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE OFFENELISTE OL1. WERT IST DIE SORTIERTE OFFENELISTE OL2. OL1 WIRD ZERSTOERT.
           [KNU 73,P.140]!!
```


Zu beachten ist, daß die Inkarnations-Bezeichner OL1 und OL2 in den Parameterlisten der Hilfsalgorithmusaufrufe von MAXLIST und INDMAX in der zweiten Produktionsregel gekwotet werden, da die Bezeichner selbst und nicht ihre Werte übergeben werden sollen.

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: SSS
ALGORITHMUS-NAME: (STRAIGHT-SELECTION-SORT)
PARAMETERTYPEN:   BENUTZER: ((OL1 . OFFENELISTE-INTEGER))
                  SYSTEM:   ((OL2 OFFENELISTE-INTEGER
                                  (INIT OL2
                                        OFFENELISTE-INTEGER)))
HILFSALGORITHMEN: (MAXLIST INDMAX)
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL1 LIES 1) == UNDEF)))
R1 : ((OL2))
L2 : ((((OL1 LIES 1) /= UNDEF)))
R2 : ((STRAIGHT-SELECTION-SORT (((OL1 ENTFERN INDMAX
                                        (('OL1)))) ,
                                ((OL2 HINZU 1 MAXLIST
                                        (('OL1)))))))
LINKESEITEN:  (L1 L2)
RECHTESEITEN: (R1 R2)
KOMMENTAR:        (STRAIGHT-SELECTION-SORT LIEST DIE MIT
                   GANZEN ZAHLEN GEFUELLTE OFFENELISTE OL1.
                   WERT IST DIE SORTIERTE OFFENELISTE OL2.
                   OL1 WIRD ZERSTOERT.[KNU 73 , P.140])
ALG-EINGABE BEENDET.
RELATION      = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL1 LIES 1) == UNDEF))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL1 LIES 1) /= UNDEF))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)
SCHRITT-1 ENDE
```

```
SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L1 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ (OL1 LIES 1)
                                (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL1 LIES 1))
ERGEBNIS     :      L2 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L2 VGL1 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (NEQ (OL1 LIES 1)
                                     (QUOTE UNDEF)))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ (OL1 LIES 1)(QUOTE UNDEF)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((NEQ (OL1 LIES 1)(QUOTE UNDEF)))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (OL2)
ERGEBNIS     :      R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2
---REGEL     : S4-51 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 KOPIERT2 ('OL1)

REGELANWENDUNG:  S4-9 AUF R2
```



```
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . ((OL1 ENTFERN INDMAX
                                        (('OL1)))))
ERGEBNIS     :      R2 ARG2 (EXPR . ((OL2 HINZU 1 MAXLIST
                                        (('OL1)))))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-32 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-33 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-33 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ OL1
                                      (OL1 ENTFERN
                                           (INDMAX 'OL1))))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM . (OL1 ENTFERN
                                     (INDMAX 'OL1)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-32 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-33 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-33 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (EXPR . (SETQ OL2
                                      (OL2 HINZU
                                      1
                                      (MAXLIST 'X2))))


REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM . (OL2 HINZU
                                     1
                                     (MAXLIST 'X2)))
SCHRITT-5 ENDE
```

```
SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
--REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :       R1 FINAL (OL2)

REGELANWENDUNG:  S6-1 AUF R2
--REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :       R2 FINAL ((DUPLIZIERE OL1 X2)
                              (OL1 ENTFERN (INDMAX 'OL1))
                              (OL2 HINZU 1 (MAXLIST 'X2))
                              (STRAIGHT-SELECTION-SORT OL1
                                                       OL2))


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ (OL1 LIES 1)
           (QUOTE UNDEF))
       OL2)
      ((NEQ (OL1 LIES 1)
            (QUOTE UNDEF))
       (DUPLIZIERE OL1 X2)
       (OL1 ENTFERN (INDMAX (QUOTE OL1)))
       (OL2 HINZU 1 (MAXLIST (QUOTE X2)))
       (STRAIGHT-SELECTION-SORT OL1 OL2))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9
 S5-33 S5-8 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.642857

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ NEQ DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<STRAIGHT-SELECTION-SORT
 <LAMBDA (OL1 OL2)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL1 (QUOTE PASSIVSTRUKTUR)))
```



```
    (COND
     ((GETD (QUOTE DUPLIZIERE))
      (COND
       ((GETD (QUOTE INDMAX))
        (COND
         ((GETD (QUOTE MAXLIST))
          (INIT OL2 OFFENELISTE-INTEGER)
          <RPLACA (QUOTE STRAIGHT-SELECTION-SORT)
           (DSUBST OL1 (QUOTE OL1)
            (DSUBST
             <QUOTE
              (RETEVAL
               (STKPOS
                (QUOTE STRAIGHT-SELECTION-SORT)
                -1)
               (QUOTE (APPLY STRAIGHT-SELECTION-SORT
                             NIL>
              (QUOTE (STRAIGHT-SELECTION-SORT OL1 OL2))
              (COPY
               (GETP (QUOTE STRAIGHT-SELECTION-SORT)
                     (QUOTE ALGORITHMUS>
           (APPLY STRAIGHT-SELECTION-SORT NIL))
          (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                     (QUOTE MAXLIST)
                     "UNDEFINIERT")
             (BREAK1 NIL T)
             NIL)))
        (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                   (QUOTE INDMAX)
                   "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
      (T (PRINTL "***ERROR*** HILFSFUNKTION"
                 (QUOTE DUPLIZIERE)
                 "UNDEFINIERT")
         (BREAK1 NIL T)
         NIL)))
   (T (PRINTL "***ERROR***" OL1 "NICHT VOM TYP"
              (QUOTE OFFENELISTE-INTEGER))
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET


(DEFINEQ

<STRAIGHT-SELECTION-SORT
  <LAMBDA (OL1 OL2)
          .
          .
          .
    siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
```

```
<PUTPROPS STRAIGHT-SELECTION-SORT KOMMENTAR
        (STRAIGHT-SELECTION-SORT LIEST DIE MIT GANZEN
                                 ZAHLEN GEFUELLTE
                                 OFFENELISTE OL1. WERT
                                 IST DIE SORTIERTE
                                 OFFENELISTE OL2. OL1
                                 WIRD ZERSTOERT.
                                 [KNU 73 , P.140])
        RELATION
        (STANDARD ((== . EQ)
                   (/= . NEQ)
                   (%< . LESSP)
                   (%> . GREATERP)
                   (%<= . LESSEQP)
                   (%>= . GREATEREQP)))
        ALGORITHMUS
        (NLAMBDA NIL
          (COND
                .
                .
                .
    siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```

4.4.4. Quick-Sort

QUICK-SORT nimmt aus mehreren Gründen eine Sonderstellung ein: einmal verdient dieser Algorithmus seinen Namen zu Recht, denn er ist einer der schnellsten Sortieralgorithmen. Weiterhin mußten zu seiner Spezifikation und Implementierung das bisher ausreichende Konzept für Spezifikationssprache und Zielstruktur erweitert werden.

Die Funktionsweise von QUICK-SORT sei hier nur kurz angedeutet (weitere Daten und Berechnungen dazu: [KNU 73]).

Sortiert werden soll eine Struktur OFFENELISTE-INTEGER. Dazu werden zwei Zeiger auf das erste und letzte Element der Liste initialisiert. Ebenso zeigen zwei "Bereichsindizes" ANF(ang) und END(e) auf diese Elemente.

```
  4    1    9    3    6    8    7
  ↑                             ↑
IND1                          IND2
ANF                           END
```

Eine boolsche Variable FL(ag) zeigt eine "Laufrichtung" an,

durch die die Indizes IND1 und IND2 erhöht oder verringert werden: FL=FALSE erlaubt Dekrementierung von IND2, FL=TRUE erlaubt Inkrementierung von IND1. Initialisiert ist FL mit FALSE.

Im Ablauf des Algorithmus werden jetzt die mit IND1 und IND2 erreichbaren Elemente der OFFENENLISTE-INTEGER verglichen. Gilt (OFL LIES IND1) < (OFL LIES IND2), so wird IND2 um 1 verringert. Gilt die ">"-Relation (gleiche Elemente sollen nicht auftreten), so werden die entsprechenden Elemente vertauscht und IND1 um 1 erhöht. Weiterhin wird FL auf TRUE gesetzt und ab sofort im "<"-Fall der Index IND1 manipuliert. Muß an einer anderen Stelle wieder vertauscht werden, wird FL wieder zurückgesetzt und ab sofort im "<"-Fall IND2 manipuliert.

Gilt schließlich IND1=IND2, so befindet sich das über IND1 (bzw. IND2) erreichbare Element an seiner endgültigen Position. QUICK-SORT wird jetzt nur noch auf die Teil-Listen rechts und links neben diesem Element angewendet.

```
  4    1    9    3    6    8    7
- - - -
  3    1    4    9    6    8    7
  ↑         ↑                   ↑
 ANF       IND1                END
           IND2
  └────┘   └──────────────────┘
 QUICK-SORT          QUICK-SORT
```

d.h., es müssen zwei QUICK-SORT-Aufrufe generiert werden, die einmal den Bereich

```
ANF' = ANF
END' = IND1 - 1
```

sowie

```
ANF'' = IND1 + 1
END'' = END
```

sortieren.

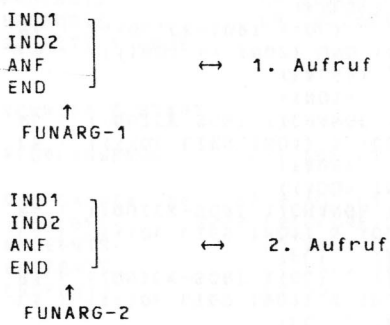
Der Algorithmus terminiert, wenn die zu sortierenden Bereiche aus einem Element oder aus zwei Elementen in richtiger Reihenfolge bestehen.

Das Problem der Spezifikation einer solchen Arbeitsweise wurde durch Einführung des Konstruktes PRA (Parallel-Rekursiver Aufruf) in die Spezifikationssprache gelöst. Die Argumente von PRA sind rekursive Aufrufe des Algorithmus, die unabhängig voneinander (parallel!) auszuführen sind.

Die Gewährleistung der Unabhängigkeit in einer vom APE-System erzeugten Implementierung konnte aber nicht durch die bisherige Zielstruktur für Algorithmen erreicht werden. Zwei Probleme ergaben sich dabei:

- **Die Ausführung des (im obigen Beispiel) ersten Aufrufs von QUICK-SORT könnte mit einem RETEVAL-Sprung terminieren. Die damit einhergehende Stackbereinigung hätte zur Folge, daß der zweite Aufruf vernichtet wird.**
- **Terminiert der erste Aufruf, so findet der zweite nicht mehr die Bedingungen vor, die der erste hatte; insbesondere die Werte von IND1, IND2, ANF und END können im Laufe der Bearbeitung des ersten Aufrufs verändert worden sein, und da es sich um globale Variable handelt, wirken sich diese Veränderungen auf den zweiten Aufruf aus.**

Um diese zu vermeiden, müssen für alle in einem PRA-Konstrukt auftretenden parallel-rekursiven Aufrufe die zum Abarbeitungszeitpunkt des PRA-Konstruktes gültigen Bindungen globaler Variabler in für jeden Aufruf private Bereiche gerettet werden. Dies geschieht durch Einbettung in ein LISP-FUNARG-Konstrukt (siehe [GK 77]) für jeden Aufruf:



Dadurch wird erreicht, daß die Referenz z.B. zu IND1 nicht auf dem Top-Level ausgewertet wird, sondern in dem jeweils zugeordneten FUNARG-Bereich. So bleiben insbesondere Manipulationen an Indizes lokal.

Durch Umdefinition des Standardsprungziels läßt sich die erste oben angesprochene Schwierigkeit beheben. Enthält mindestens eine rechte Seite einer Spezifikation ein PRA-Konstrukt, so werden alle RETEVAL-Sprunge unter das letzte PROG ausgeführt. Der Grund für dieses Ziel liegt darin, daß die Realisierung von parallel-rekursiven Aufrufen über ein LISP-PROG erreicht wird. Mit dieser Modifikation bleibt der wichtige Effekt einer Stackbereinigung teilweise erhalten, und die oben angeschnittene Problematik wird dadurch gelöst. (Zur weiteren Erläuterung siehe auch die Funktionsbeschreibung von PAR-REK-AUFBAU; diese Funktion erzeugt den Code für PRA-Konstrukte).

Quick-Sort ist wie folgt spezifiziert:


```
ALGORITHMUS-NAME: QUICK-SORT;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-INTEGER OL ;
                  SYSTEM:   INTEGER ANF  := 1,
                            INTEGER END  := (LISTSIZE OL),
                            INTEGER IND1 := 1,
                            INTEGER IND2 := (LISTSIZE OL),
                            BOOL    FL   := FALSE;

HILFSALGORITHMEN: CHANGE, LISTSIZE;

ALGORITHMUS:      QUICK-SORT (OL,IND1,IND2,FL,ANF,END) =

      {  IND1 = IND2  UND  ANF = IND1  UND  END = IND2
       ---> OL;

        (OL LIES IND1) < (OL LIES IND2)  UND  FL = FALSE
       ---> QUICK-SORT (OL,IND1,(SUB1 IND2),FL,ANF,END);

        (OL LIES IND1) < (OL LIES IND2)  UND  FL = TRUE
       ---> QUICK-SORT(OL,(ADD1 IND1), IND2,FL,ANF,END);

        (OL LIES IND1) > (OL LIES IND2)  UND  FL = FALSE
       ---> QUICK-SORT(CHANGE('OL,IND1,IND2),(ADD1 IND1),
                       IND2,TRUE,ANF,END);

        (OL LIES IND1) > (OL LIES IND2)  UND  FL = TRUE
       ---> QUICK-SORT(CHANGE('OL,IND1,IND2),IND1,
                       (SUB1 IND2),FALSE,ANF,END);

        IND1 = IND2  UND  ANF /= IND1  UND  END /= IND2
       ---> PRA(QUICK-SORT(OL,ANF,(SUB1 IND1),
                           FALSE,ANF,(SUB1 IND1)),
                QUICK-SORT(OL,(ADD1 IND2),END,
                           FALSE,(ADD1 IND2),END));

        IND1 = IND2  UND    (ADD1 ANF) = END
       ---> OL ;

        IND1 = IND2 UND  ANF = IND1 UND  (ADD1 ANF) /= END
       ---> QUICK-SORT(OL,(ADD1 IND1),END,
                       FALSE,(ADD1 IND1),END);

        IND1 = IND2 UND  END = IND2 UND  (SUB1 END) /= ANF
       ---> QUICK-SORT(OL,ANF,(SUB1 IND2),
                       FALSE,ANF,(SUB1 IND2))    }

KOMMENTAR: !!
```

In den Hilfsalgorithmusaufrufen von CHANGE in den Produktionsregeln vier und fünf müssen die Inkarnations-Bezeichner OL gekwotet werden, da der Bezeichner selbst, nicht sein Wert übergeben werden soll.

Es folgt das Protokoll der Implementierung:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: QUI
ALGORITHMUS-NAME: (QUICK-SORT)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-INTEGER))
                  SYSTEM:   ((ANF INTEGER 1)
                             (END INTEGER (LISTSIZE OL))
                             (IND1 INTEGER 1)
                             (IND2 INTEGER (LISTSIZE OL))
                             (FL BOOL FALSE))
HILFSALGORITHMEN: (CHANGE LISTSIZE)
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : (((IND1 == IND2) UND (ANF == IND1) UND (END == IND2)))
R1 : ((OL))
L2 : ((((OL LIES IND1) < (OL LIES IND2)) UND (FL == FALSE)))
R2 : ((QUICK-SORT ((OL) , (IND1) , ((SUB1 IND2)) ,
                  (FL) , (ANF) , (END))))
L3 : ((((OL LIES IND1) < (OL LIES IND2)) UND (FL == TRUE)))
R3 : ((QUICK-SORT ((OL) , ((ADD1 IND1)) , (IND2) ,
                  (FL) , (ANF) , (END))))
L4 : ((((OL LIES IND1) > (OL LIES IND2)) UND (FL == FALSE)))
R4 : ((QUICK-SORT ((CHANGE (('OL) , (IND1) , (IND2))) ,
                  ((ADD1 IND1)) , (IND2) ,
                  (TRUE) , (ANF) , (END))))
L5 : ((((OL LIES IND1) > (OL LIES IND2)) UND (FL == TRUE)))
R5 : ((QUICK-SORT ((CHANGE (('OL) , (IND1) , (IND2))) ,
                  (IND1) , ((SUB1 IND2)) ,
                  (FALSE) , (ANF) , (END))))
L6 : (((IND1 == IND2) UND (ANF /= IND1) UND (END /= IND2)))
R6 : (((QUICK-SORT ((OL) , (ANF) , ((SUB1 IND1)) ,
                    (FALSE) , (ANF) , ((SUB1 IND1)))) ,
       (QUICK-SORT ((OL) , ((ADD1 IND2)) , (END) ,
                    (FALSE) , ((ADD1 IND2)) , (END)))))
L7 : (((IND1 == IND2) UND ((ADD1 ANF) == END)))
R7 : ((OL))
L8 : (((IND1 == IND2) UND (ANF == IND1) UND
      ((ADD1 ANF) /= END)))
R8 : ((QUICK-SORT ((OL) , ((ADD1 IND1)) , (END) ,
                  (FALSE) , ((ADD1 IND1)) , (END))))
L9 : (((IND1 == IND2) UND (END == IND2) UND
       ((SUB1 END) /= ANF)))
R9 : ((QUICK-SORT ((OL) , (ANF) , ((SUB1 IND2)) ,
                  (FALSE) , (ANF) , ((SUB1 IND2)))))
LINKESEITEN:  (L1 L2 L3 L4 L5 L6 L7 L8 L9)
RECHTESEITEN: (R1 R2 R3 R4 R5 R6 R7 R8 R9)
KOMMENTAR: ()
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
```



```
EINLESE-PHASE BEENDET

START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (IND1 == IND2))
ERGEBNIS     :      L1 VGL2 (EXPR . (ANF == IND1))
ERGEBNIS     :      L1 VGL3 (EXPR . (END == IND2))
ERGEBNIS     :      L1 LOGOP (AND)
ERGEBNIS     :      L1 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES IND1)
                                     < (OL LIES IND2)))
ERGEBNIS     :      L2 VGL2 (EXPR . (FL == FALSE))
ERGEBNIS     :      L2 LOGOP (AND)
ERGEBNIS     :      L2 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES IND1)
                                     < (OL LIES IND2)))
ERGEBNIS     :      L3 VGL2 (EXPR . (FL == TRUE))
ERGEBNIS     :      L3 LOGOP (AND)
ERGEBNIS     :      L3 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . ((OL LIES IND1)
                                     > (OL LIES IND2)))
ERGEBNIS     :      L4 VGL2 (EXPR . (FL == FALSE))
ERGEBNIS     :      L4 LOGOP (AND)
ERGEBNIS     :      L4 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L5
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS     :      L5 VGL1 (EXPR . ((OL LIES IND1)
                                     > (OL LIES IND2)))
ERGEBNIS     :      L5 VGL2 (EXPR . (FL == TRUE))
ERGEBNIS     :      L5 LOGOP (AND)
ERGEBNIS     :      L5 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L6
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS     :      L6 VGL1 (EXPR . (IND1 == IND2))
ERGEBNIS     :      L6 VGL2 (EXPR . (ANF /= IND1))
ERGEBNIS     :      L6 VGL3 (EXPR . (END /= IND2))
ERGEBNIS     :      L6 LOGOP (AND)
ERGEBNIS     :      L6 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L7
```

```
---REGEL    : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :      L7 VGL1 (EXPR . (IND1 == IND2))
ERGEBNIS    :      L7 VGL2 (EXPR . ((ADD1 ANF) == END))
ERGEBNIS    :      L7 LOGOP (AND)
ERGEBNIS    :      L7 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L8
---REGEL    : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 VGL1 (EXPR . (IND1 == IND2))
ERGEBNIS    :      L8 VGL2 (EXPR . (ANF == IND1))
ERGEBNIS    :      L8 VGL3 (EXPR . ((ADD1 ANF) /= END))
ERGEBNIS    :      L8 LOGOP (AND)
ERGEBNIS    :      L8 VGLLIST (VGL1 VGL2 VGL3)

REGELANWENDUNG:  S1-1 AUF L9
---REGEL    : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL1 (EXPR . (IND1 == IND2))
ERGEBNIS    :      L9 VGL2 (EXPR . (END == IND2))
ERGEBNIS    :      L9 VGL3 (EXPR . ((SUB1 END) /= ANF))
ERGEBNIS    :      L9 LOGOP (AND)
ERGEBNIS    :      L9 VGLLIST (VGL1 VGL2 VGL3)
SCHRITT-1 ENDE


SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL1 (LINKS . IND1)
ERGEBNIS    :      L1 VGL1 (RECHTS . IND2)
ERGEBNIS    :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL1 (FORM . (EQ IND1 IND2))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL2 (LINKS . ANF)
ERGEBNIS    :      L1 VGL2 (RECHTS . IND1)
ERGEBNIS    :      L1 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL2 (FORM . (EQ ANF IND1))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL3

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL3
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL3 (LINKS . END)
ERGEBNIS    :      L1 VGL3 (RECHTS . IND2)
```



```
ERGEBNIS    :      L1 VGL3 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL3 (FORM . (EQ END IND2))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL1 (LINKS . (OL LIES IND1))
ERGEBNIS    :      L2 VGL1 (RECHTS . (OL LIES IND2))
ERGEBNIS    :      L2 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL1 (FORM . (LESSP (OL LIES IND1)
                                          (OL LIES IND2)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL2 (LINKS . FL)
ERGEBNIS    :      L2 VGL2 (RECHTS .)
ERGEBNIS    :      L2 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL2 (FORM . (EQ FL))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (LINKS . (OL LIES IND1))
ERGEBNIS    :      L3 VGL1 (RECHTS . (OL LIES IND2))
ERGEBNIS    :      L3 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (FORM . (LESSP (OL LIES IND1)
                                          (OL LIES IND2)))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL2 (LINKS . FL)
ERGEBNIS    :      L3 VGL2 (RECHTS . T)
ERGEBNIS    :      L3 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL2 (FORM . (EQ FL T))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL1 (LINKS . (OL LIES IND1))
ERGEBNIS    :      L4 VGL1 (RECHTS . (OL LIES IND2))
```

```
ERGEBNIS    :       L4 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :       L4 VGL1 (FORM . (GREATERP (OL LIES IND1)
                                              (OL LIES IND2)))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L4 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :       L4 VGL2 (LINKS . FL)
ERGEBNIS    :       L4 VGL2 (RECHTS .)
ERGEBNIS    :       L4 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :       L4 VGL2 (FORM . (EQ FL))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :       L5 VGL1 (LINKS . (OL LIES IND1))
ERGEBNIS    :       L5 VGL1 (RECHTS . (OL LIES IND2))
ERGEBNIS    :       L5 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :       L5 VGL1 (FORM . (GREATERP (OL LIES IND1)
                                              (OL LIES IND2)))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L5 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :       L5 VGL2 (LINKS . FL)
ERGEBNIS    :       L5 VGL2 (RECHTS . T)
ERGEBNIS    :       L5 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :       L5 VGL2 (FORM . (EQ FL T))

REGELANWENDUNG:  S2-1 AUF L6 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L6 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL1 (LINKS . IND1)
ERGEBNIS    :       L6 VGL1 (RECHTS . IND2)
ERGEBNIS    :       L6 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L6 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL1 (FORM . (EQ IND1 IND2))

REGELANWENDUNG:  S2-1 AUF L6 , VERGLEICH  VGL2
```



```
REGELANWENDUNG:  S2-2 AUF L6 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL2 (LINKS . ANF)
ERGEBNIS    :       L6 VGL2 (RECHTS . IND1)
ERGEBNIS    :       L6 VGL2 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L6 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL2 (FORM . (NEQ ANF IND1))

REGELANWENDUNG:  S2-1 AUF L6 , VERGLEICH  VGL3

REGELANWENDUNG:  S2-2 AUF L6 , VERGLEICH  VGL3
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL3 (LINKS . END)
ERGEBNIS    :       L6 VGL3 (RECHTS . IND2)
ERGEBNIS    :       L6 VGL3 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L6 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :       L6 VGL3 (FORM . (NEQ END IND2))

REGELANWENDUNG:  S2-1 AUF L7 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L7 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :       L7 VGL1 (LINKS . IND1)
ERGEBNIS    :       L7 VGL1 (RECHTS . IND2)
ERGEBNIS    :       L7 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L7 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :       L7 VGL1 (FORM . (EQ IND1 IND2))

REGELANWENDUNG:  S2-1 AUF L7 , VERGLEICH  VGL2
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :       L7 VGL2 (LINKS . (ADD1 ANF))
ERGEBNIS    :       L7 VGL2 (RECHTS . END)
ERGEBNIS    :       L7 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L7 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :       L7 VGL2 (FORM . (EQ (ADD1 ANF) END))

REGELANWENDUNG:  S2-1 AUF L8 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L8 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :       L8 VGL1 (LINKS . IND1)
ERGEBNIS    :       L8 VGL1 (RECHTS . IND2)
ERGEBNIS    :       L8 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L8 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :       L8 VGL1 (FORM . (EQ IND1 IND2))
```

```
REGELANWENDUNG:  S2-1 AUF L8 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L8 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 VGL2 (LINKS . ANF)
ERGEBNIS    :      L8 VGL2 (RECHTS . IND1)
ERGEBNIS    :      L8 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L8 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 VGL2 (FORM . (EQ ANF IND1))

REGELANWENDUNG:  S2-1 AUF L8 , VERGLEICH  VGL3
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 VGL3 (LINKS . (ADD1 ANF))
ERGEBNIS    :      L8 VGL3 (RECHTS . END)
ERGEBNIS    :      L8 VGL3 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L8 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 VGL3 (FORM . (NEQ (ADD1 ANF) END))

REGELANWENDUNG:  S2-1 AUF L9 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L9 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL1 (LINKS . IND1)
ERGEBNIS    :      L9 VGL1 (RECHTS . IND2)
ERGEBNIS    :      L9 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L9 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL1 (FORM . (EQ IND1 IND2))

REGELANWENDUNG:  S2-1 AUF L9 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L9 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL2 (LINKS . END)
ERGEBNIS    :      L9 VGL2 (RECHTS . IND2)
ERGEBNIS    :      L9 VGL2 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L9 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL2 (FORM . (EQ END IND2))

REGELANWENDUNG:  S2-1 AUF L9 , VERGLEICH  VGL3
---REGEL    : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL3 (LINKS . (SUB1 END))
ERGEBNIS    :      L9 VGL3 (RECHTS . ANF)
ERGEBNIS    :      L9 VGL3 (REL . /=)

REGELANWENDUNG:  S2-9 AUF L9 , VERGLEICH  VGL3
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 VGL3 (FORM . (NEQ (SUB1 END) ANF))
SCHRITT-2 ENDE
```



```
SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 FINAL ((AND (EQ IND1 IND2)
                              (EQ ANF IND1)
                              (EQ END IND2)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 FINAL ((AND (LESSP (OL LIES IND1)
                                         (OL LIES IND2))
                                  (EQ FL)))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 FINAL ((AND (LESSP (OL LIES IND1)
                                         (OL LIES IND2))
                                  (EQ FL T)))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 FINAL ((AND (GREATERP (OL LIES IND1)
                                            (OL LIES IND2))
                                  (EQ FL)))

REGELANWENDUNG:  S3-1 AUF L5
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 FINAL ((AND (GREATERP (OL LIES IND1)
                                            (OL LIES IND2))
                                  (EQ FL T)))

REGELANWENDUNG:  S3-1 AUF L6
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :      L6 FINAL ((AND (EQ IND1 IND2)
                                  (NEQ ANF IND1)
                                  (NEQ END IND2)))

REGELANWENDUNG:  S3-1 AUF L7
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L7
ERGEBNIS    :      L7 FINAL ((AND (EQ IND1 IND2)
                                  (EQ (ADD1 ANF)
                                      END)))

REGELANWENDUNG:  S3-1 AUF L8
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L8
ERGEBNIS    :      L8 FINAL ((AND (EQ IND1 IND2)
                                  (EQ ANF IND1)
                                  (NEQ (ADD1 ANF)
                                       END)))

REGELANWENDUNG:  S3-1 AUF L9
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L9
ERGEBNIS    :      L9 FINAL ((AND (EQ IND1 IND2)
                                  (EQ END IND2)
                                  (NEQ (SUB1 END)
```

```
                                        ANF)))
SCHRITT-3 ENDE

SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (OL)
ERGEBNIS     :      R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-52 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (OL))
ERGEBNIS     :      R2 ARG2 (EXPR . (IND1))
ERGEBNIS     :      R2 ARG3 (EXPR . ((SUB1 IND2)))
ERGEBNIS     :      R2 ARG4 (EXPR . (FL))
ERGEBNIS     :      R2 ARG5 (EXPR . (ANF))
ERGEBNIS     :      R2 ARG6 (EXPR . (END))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3

REGELANWENDUNG:  S4-52 AUF R3

REGELANWENDUNG:  S4-53 AUF R3

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . ((ADD1 IND1)))
ERGEBNIS     :      R3 ARG3 (EXPR . (IND2))
```



```
ERGEBNIS     :      R3 ARG4 (EXPR . (FL))
ERGEBNIS     :      R3 ARG5 (EXPR . (ANF))
ERGEBNIS     :      R3 ARG6 (EXPR . (END))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R4
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R4

REGELANWENDUNG:  S4-52 AUF R4

REGELANWENDUNG:  S4-53 AUF R4

REGELANWENDUNG:  S4-54 AUF R4

REGELANWENDUNG:  S4-55 AUF R4

REGELANWENDUNG:  S4-9 AUF R4
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (EXPR . (CHANGE (('OL) ,
                                             (IND1) ,
                                             (IND2))))
ERGEBNIS     :      R4 ARG2 (EXPR . ((ADD1 IND1)))
ERGEBNIS     :      R4 ARG3 (EXPR . (IND2))
ERGEBNIS     :      R4 ARG4 (EXPR . (TRUE))
ERGEBNIS     :      R4 ARG5 (EXPR . (ANF))
ERGEBNIS     :      R4 ARG6 (EXPR . (END))
ERGEBNIS     :      R4 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R5
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R5

REGELANWENDUNG:  S4-52 AUF R5

REGELANWENDUNG:  S4-53 AUF R5

REGELANWENDUNG:  S4-54 AUF R5

REGELANWENDUNG:  S4-55 AUF R5

REGELANWENDUNG:  S4-9 AUF R5
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG1 (EXPR . (CHANGE (('OL) ,
                                             (IND1) ,
                                             (IND2))))
ERGEBNIS     :      R5 ARG2 (EXPR . (IND1))
ERGEBNIS     :      R5 ARG3 (EXPR . ((SUB1 IND2)))
ERGEBNIS     :      R5 ARG4 (EXPR . (FALSE))
ERGEBNIS     :      R5 ARG5 (EXPR . (ANF))
ERGEBNIS     :      R5 ARG6 (EXPR . (END))
ERGEBNIS     :      R5 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)
```

```
REGELANWENDUNG:  S4-1 AUF R6

REGELANWENDUNG:  S4-2 AUF R6

REGELANWENDUNG:  S4-3 AUF R6
---REGEL     : S4-3 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :      R6 EVALUIERT (PARALLEL-REKURSIV)
ERGEBNIS     :      R6 NEUE-NAMEN (R6-1 R6-2)
ERGEBNIS     :      R6-1 EVALUIERT (FUN-ARG-REKURSION)
ERGEBNIS     :      R6-2 EVALUIERT (FUN-ARG-REKURSION)

REGELANWENDUNG:  S4-1 AUF R6-1
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R6-1

REGELANWENDUNG:  S4-52 AUF R6-1

REGELANWENDUNG:  S4-53 AUF R6-1

REGELANWENDUNG:  S4-54 AUF R6-1

REGELANWENDUNG:  S4-55 AUF R6-1

REGELANWENDUNG:  S4-9 AUF R6-1
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG1 (EXPR . (OL))
ERGEBNIS     :      R6-1 ARG2 (EXPR . (ANF))
ERGEBNIS     :      R6-1 ARG3 (EXPR . ((SUB1 IND1)))
ERGEBNIS     :      R6-1 ARG4 (EXPR . (FALSE))
ERGEBNIS     :      R6-1 ARG5 (EXPR . (ANF))
ERGEBNIS     :      R6-1 ARG6 (EXPR . ((SUB1 IND1)))
ERGEBNIS     :      R6-1 ARGLIST (ARG1 ARG2 ARG3
                                  ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R6-2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R6-2

REGELANWENDUNG:  S4-52 AUF R6-2

REGELANWENDUNG:  S4-53 AUF R6-2

REGELANWENDUNG:  S4-54 AUF R6-2

REGELANWENDUNG:  S4-55 AUF R6-2

REGELANWENDUNG:  S4-9 AUF R6-2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG1 (EXPR . (OL))
ERGEBNIS     :      R6-2 ARG2 (EXPR . ((ADD1 IND2)))
ERGEBNIS     :      R6-2 ARG3 (EXPR . (END))
```



```
ERGEBNIS     :      R6-2 ARG4 (EXPR . (FALSE))
ERGEBNIS     :      R6-2 ARG5 (EXPR . ((ADD1 IND2)))
ERGEBNIS     :      R6-2 ARG6 (EXPR . (END))
ERGEBNIS     :      R6-2 ARGLIST (ARG1 ARG2 ARG3
                                  ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R7

REGELANWENDUNG:  S4-2 AUF R7
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R7
ERGEBNIS     :      R7 FORM (OL)
ERGEBNIS     :      R7 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R8
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R8

REGELANWENDUNG:  S4-52 AUF R8

REGELANWENDUNG:  S4-53 AUF R8

REGELANWENDUNG:  S4-54 AUF R8

REGELANWENDUNG:  S4-55 AUF R8

REGELANWENDUNG:  S4-9 AUF R8
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG1 (EXPR . (OL))
ERGEBNIS     :      R8 ARG2 (EXPR . ((ADD1 IND1)))
ERGEBNIS     :      R8 ARG3 (EXPR . (END))
ERGEBNIS     :      R8 ARG4 (EXPR . (FALSE))
ERGEBNIS     :      R8 ARG5 (EXPR . ((ADD1 IND1)))
ERGEBNIS     :      R8 ARG6 (EXPR . (END))
ERGEBNIS     :      R8 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)

REGELANWENDUNG:  S4-1 AUF R9
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R9

REGELANWENDUNG:  S4-52 AUF R9

REGELANWENDUNG:  S4-53 AUF R9

REGELANWENDUNG:  S4-54 AUF R9

REGELANWENDUNG:  S4-55 AUF R9

REGELANWENDUNG:  S4-9 AUF R9
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG1 (EXPR . (OL))
ERGEBNIS     :      R9 ARG2 (EXPR . (ANF))
ERGEBNIS     :      R9 ARG3 (EXPR . ((SUB1 IND2)))
```

```
ERGEBNIS     :       R9 ARG4 (EXPR . (FALSE))
ERGEBNIS     :       R9 ARG5 (EXPR . (ANF))
ERGEBNIS     :       R9 ARG6 (EXPR . ((SUB1 IND2)))
ERGEBNIS     :       R9 ARGLIST (ARG1 ARG2 ARG3 ARG4 ARG5 ARG6)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (EXPR . (SETQ IND1 IND1))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG3 (LISP-AUFRUF SUB1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG3 (FORM . (SETQ IND2 (SUB1 IND2)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG4 (EXPR . (SETQ FL FL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG4
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG4 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG5 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG5 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG6 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG6
```



```
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG2 (FORM . (SETQ IND1 (ADD1 IND1)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG3 (EXPR . (SETQ IND2 IND2))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG4 (EXPR . (SETQ FL FL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG4 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG5 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG5 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG6 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R4 , ARGUMENT ARG1
```

```
REGELANWENDUNG:  S5-31 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-32 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-33 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-4 AUF R4 , ARGUMENT ARG1
---REGEL     : S5-4 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG1 (EXPR . (SETQ OL
                                      (CHANGE 'OL
                                              IND1
                                              IND2)))

REGELANWENDUNG:  S5-8 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG1

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R4 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG2 (FORM . (SETQ IND1 (ADD1 IND1)))

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG3 (EXPR . (SETQ IND2 IND2))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG4 (EXPR . (SETQ FL T))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG5 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG5 (FORM .)

REGELANWENDUNG:  S5-1 AUF R4 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 ARG6 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R4 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
```



```
ERGEBNIS     :      R4 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-32 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-33 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-4 AUF R5 , ARGUMENT ARG1
---REGEL     : S5-4 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG1 (EXPR . (SETQ OL
                                      (CHANGE 'OL
                                              IND1
                                              IND2)))

REGELANWENDUNG:  S5-8 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-9 AUF R5 , ARGUMENT ARG1

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG2 (EXPR . (SETQ IND1 IND1))

REGELANWENDUNG:  S5-9 AUF R5 , ARGUMENT ARG2
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R5 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG3 (LISP-AUFRUF SUB1)

REGELANWENDUNG:  S5-5 AUF R5 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG3 (FORM . (SETQ IND2 (SUB1 IND2)))

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG4 (EXPR . (SETQ FL))

REGELANWENDUNG:  S5-9 AUF R5 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG5 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R5 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG5 (FORM .)

REGELANWENDUNG:  S5-1 AUF R5 , ARGUMENT ARG6
```

```
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG6 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R5 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 ARG6 (FORM .)

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG1
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG1 (FORM .)

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG2
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG2 (FORM . (IND1 ANF))

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG3
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG3 (FORM . (IND2 (SUB1 IND1)))

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG4
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG4 (FORM . FL)

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG5
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG5 (FORM .)

REGELANWENDUNG:  S5-D1 AUF R6-1 , ARGUMENT ARG6
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-1
ERGEBNIS     :      R6-1 ARG6 (FORM . (END (SUB1 IND1)))

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG1
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG1 (FORM .)

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG2
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG2 (FORM . (IND1 (ADD1 IND2)))

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG3
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG3 (FORM . (IND2 END))

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG4
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG4 (FORM . FL)

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG5
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG5 (FORM . (ANF (ADD1 IND2)))

REGELANWENDUNG:  S5-D1 AUF R6-2 , ARGUMENT ARG6
---REGEL     : S5-D1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6-2
ERGEBNIS     :      R6-2 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG1
```



```
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R8 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R8 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R8 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG2 (FORM . (SETQ IND1 (ADD1 IND1)))

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG3 (EXPR . (SETQ IND2 END))

REGELANWENDUNG:  S5-9 AUF R8 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG4 (EXPR . (SETQ FL))

REGELANWENDUNG:  S5-9 AUF R8 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG5

REGELANWENDUNG:  S5-2 AUF R8 , ARGUMENT ARG5
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG5 (LISP-AUFRUF ADD1)
ERGEBNIS     :      R8 ARG5 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R8 , ARGUMENT ARG5
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG5 (FORM . (SETQ ANF (ADD1 STD1-5)))
ERGEBNIS     :      R8 KOPIERT-LISP (ARG5 (IND1 STD1-5))

REGELANWENDUNG:  S5-1 AUF R8 , ARGUMENT ARG6
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG6 (EXPR . (SETQ END END))

REGELANWENDUNG:  S5-9 AUF R8 , ARGUMENT ARG6
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS     :      R8 ARG6 (FORM .)

REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R9 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG1 (FORM .)
```

```
REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG2 (EXPR . (SETQ IND1 ANF))

REGELANWENDUNG:  S5-9 AUF R9 , ARGUMENT ARG2

REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R9 , ARGUMENT ARG3
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG3 (LISP-AUFRUF SUB1)

REGELANWENDUNG:  S5-5 AUF R9 , ARGUMENT ARG3
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG3 (FORM . (SETQ IND2 (SUB1 IND2)))

REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG4 (EXPR . (SETQ FL))

REGELANWENDUNG:  S5-9 AUF R9 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG5
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG5 (EXPR . (SETQ ANF ANF))

REGELANWENDUNG:  S5-9 AUF R9 , ARGUMENT ARG5
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG5 (FORM .)

REGELANWENDUNG:  S5-1 AUF R9 , ARGUMENT ARG6

REGELANWENDUNG:  S5-2 AUF R9 , ARGUMENT ARG6
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG6 (LISP-AUFRUF SUB1)
ERGEBNIS     :      R9 ARG6 (KOPIERT-LISP)

REGELANWENDUNG:  S5-5 AUF R9 , ARGUMENT ARG6
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS     :      R9 ARG6 (FORM . (SETQ END (SUB1 STD1-6)))
ERGEBNIS     :      R9 KOPIERT-LISP (ARG6 (IND2 STD1-6))
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FINAL (OL)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((SETQ IND2 (SUB1 IND2))
```



```
                                (QUICK-SORT OL IND1 IND2
                                            FL ANF END))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ IND1 (ADD1 IND1))
                              (QUICK-SORT OL IND1 IND2
                                          FL ANF END))

REGELANWENDUNG:  S6-1 AUF R4
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :      R4 FINAL ((SETQ OL (CHANGE 'OL IND1 IND2))
                              (SETQ IND1 (ADD1 IND1))
                              (SETQ FL T)
                              (QUICK-SORT OL IND1 IND2
                                          FL ANF END))

REGELANWENDUNG:  S6-1 AUF R5
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :      R5 FINAL ((SETQ OL (CHANGE 'OL IND1 IND2))
                              (SETQ IND2 (SUB1 IND2))
                              (SETQ FL)
                              (QUICK-SORT OL IND1 IND2
                                          FL ANF END))

REGELANWENDUNG:  S6-1 AUF R6

REGELANWENDUNG:  S6-21 AUF R6

REGELANWENDUNG:  S6-22 AUF R6
---REGEL     : S6-22 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :      R6 FINAL
((<SETQ A-FORM
      (SUBST (GENSYM)
             (QUOTE F2)
             (SUBST (GENSYM)
                    (QUOTE F1)
                    (QUOTE ((SETQ F1
                                  (PROG
                                   ((IND1 ANF)
                                    (IND2 (SUB1 IND1))
                                    FL
                                    (END (SUB1 IND1)))
                                   (RETURN
                                     (FUNCTION APPLY*
                                         (ANF END
                                          IND1 IND2 FL)))))
                            (SETQ F2
                                  (PROG
                                   ((IND1 (ADD1 IND2))
                                    (IND2 END)
                                    FL
                                    (ANF (ADD1 IND2)))
                                   (RETURN
                                     (FUNCTION APPLY*
                                         (ANF END
```

```
                                    IND1 IND2 FL)))))
                          (RETEVAL
                            (STKNTH 1 (STKPOS (QUOTE PROG)))
                            (QUOTE (PROG NIL
                                         (APPLY* F1 QUICK-SORT)
                                         (APPLY* F2 QUICK-SORT>
 (MAPC A-FORM (FUNCTION EVAL)))

REGELANWENDUNG:  S6-1 AUF R7

REGELANWENDUNG:  S6-21 AUF R7
---REGEL    : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R7
ERGEBNIS    :       R7 FINAL (OL)

REGELANWENDUNG:  S6-1 AUF R8
REGELANWENDUNG:  S6-1 AUF R8
---REGEL    : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R8
ERGEBNIS    :       R8 FINAL ((DUPLIZIERE IND1 (STD1-5))
                              (SETQ IND1 (ADD1 IND1))
                              (SETQ IND2 END)
                              (SETQ FL)
                              (SETQ ANF (ADD1 STD1-5))
                              (QUICK-SORT OL IND1 IND2
                                          FL ANF END))

REGELANWENDUNG:  S6-1 AUF R9
---REGEL    : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R9
ERGEBNIS    :       R9 FINAL ((DUPLIZIERE IND2 (STD1-6))
                              (SETQ IND1 ANF)
                              (SETQ IND2 (SUB1 IND2))
                              (SETQ FL)
                              (SETQ END (SUB1 STD1-6))
                              (QUICK-SORT OL IND1 IND2
                                          FL ANF END))


START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((AND (EQ IND1 IND2)
            (EQ ANF IND1)
            (EQ END IND2))
       OL)
      ((AND (LESSP (OL LIES IND1)
                   (OL LIES IND2))
            (EQ FL))
       (SETQ IND2 (SUB1 IND2))
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      ((AND (LESSP (OL LIES IND1)
                   (OL LIES IND2))
            (EQ FL T))
       (SETQ IND1 (ADD1 IND1))
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      ((AND (GREATERP (OL LIES IND1)
```



```
                      (OL LIES IND2))
            (EQ FL))
       (SETQ OL (CHANGE (QUOTE OL)
                        IND1 IND2))
       (SETQ IND1 (ADD1 IND1))
       (SETQ FL T)
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      ((AND (GREATERP (OL LIES IND1)
                      (OL LIES IND2))
            (EQ FL T))
       (SETQ OL (CHANGE (QUOTE OL)
                        IND1 IND2))
       (SETQ IND2 (SUB1 IND2))
       (SETQ FL)
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      ((AND (EQ IND1 IND2)
            (NEQ ANF IND1)
            (NEQ END IND2))
       <SETQ A-FORM
            (SUBST (GENSYM)
                   (QUOTE F2)
                   (SUBST (GENSYM)
                          (QUOTE F1)
                          (QUOTE
                            (<SETQ F1
                                  (PROG
                                   ((IND1 ANF)
                                    (IND2 (SUB1 IND1))
                                    FL
                                    (END (SUB1 IND1)))
                                   (RETURN
                                     (FUNCTION APPLY*
                                       (ANF END IND1 IND2 FL>
                             <SETQ F2
                                  (PROG
                                   ((IND1 (ADD1 IND2))
                                    (IND2 END)
                                    FL
                                    (ANF (ADD1 IND2)))
                                   (RETURN
                                     (FUNCTION APPLY*
                                       (ANF END IND1 IND2 FL>
                            (RETEVAL
                              (STKNTH 1
                                      (STKPOS (QUOTE PROG)))
                              (QUOTE
                                (PROG NIL
                                      (APPLY* F1 QUICK-SORT)
                                      (APPLY* F2 QUICK-SORT>
       (MAPC A-FORM (FUNCTION EVAL)))
      ((AND (EQ IND1 IND2)
            (EQ (ADD1 ANF)
                END))
       OL)
      ((AND (EQ IND1 IND2)
            (EQ ANF IND1)
```

```
          (NEQ (ADD1 ANF)
               END))
       (DUPLIZIERE IND1 (STD1-5))
       (SETQ IND1 (ADD1 IND1))
       (SETQ IND2 END)
       (SETQ FL)
       (SETQ ANF (ADD1 STD1-5))
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      ((AND (EQ IND1 IND2)
            (EQ END IND2)
            (NEQ (SUB1 END)
                 ANF))
       (DUPLIZIERE IND2 (STD1-6))
       (SETQ IND1 ANF)
       (SETQ IND2 (SUB1 IND2))
       (SETQ FL)
       (SETQ END (SUB1 STD1-6))
       (QUICK-SORT OL IND1 IND2 FL ANF END))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-3 S4-9
 S5-1 S5-2 S5-4 S5-5 S5-9 S5-D1 S6-1 S6-21 S6-22)

REGELANWENDUNGSFAKTOR:  0.645914

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ LESSP GREATERP NEQ DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<QUICK-SORT
 <LAMBDA (OL IND1 IND2 FL ANF END)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((GETD (QUOTE DUPLIZIERE))
      (COND
       ((GETD (QUOTE LISTSIZE))
        (COND
         <(GETD (QUOTE CHANGE))
           (SETQ ANF 1)
           (SETQ END (LISTSIZE OL))
           (SETQ IND1 1)
           (SETQ IND2 (LISTSIZE OL))
           (SETQ FL NIL)
           <RPLACA (QUOTE QUICK-SORT)
             (DSUBST OL (QUOTE OL)
```



```
                 (DSUBST
                   <QUOTE
                     (RETEVAL (STKNTH 1
                                      (STKPOS (QUOTE PROG)))
                              (QUOTE (APPLY QUICK-SORT NIL>
                     (QUOTE (QUICK-SORT OL IND1 IND2
                                        FL ANF END))
                     (COPY (GETP (QUOTE QUICK-SORT)
                                 (QUOTE ALGORITHMUS>
                   (PROG NIL
                         (APPLY QUICK-SORT NIL)
                         (RETURN (EVAL OL>
                   (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                              (QUOTE CHANGE)
                              "UNDEFINIERT")
                      (BREAK1 NIL T)
                      NIL)))
                 (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                            (QUOTE LISTSIZE)
                            "UNDEFINIERT")
                    (BREAK1 NIL T)
                    NIL)))
               (T (PRINTL "***ERROR*** HILFSFUNKTION"
                          (QUOTE DUPLIZIERE)
                          "UNDEFINIERT")
                  (BREAK1 NIL T)
                  NIL)))
             (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
                        (QUOTE OFFENELISTE-INTEGER))
                (BREAK1 NIL T)
                NIL>>
LISP-PHASE BEENDET


(DEFINEQ

<QUICK-SORT
  <LAMBDA (OL IND1 IND2 FL ANF END)
          .
          .
          .
    siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS QUICK-SORT KOMMENTAR ()
          RELATION
          (STANDARD ((== . EQ)
                     (/= . NEQ)
                     (%< . LESSP)
                     (%> . GREATERP)
                     (%<= . LESSEQP)
                     (%>= . GREATEREQP)))
          ALGORITHMUS
          (NLAMBDA NIL
```

```
            (COND
                .
                .
                .
    siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```


4.5. Andere Algorithmen

In diesem Abschnitt werden noch einige Algorithmen besprochen, die als Hilfsalgorithmen für andere implementierte Algorithmen dienen, aber natürlich auch einzeln aufgerufen werden können. Es handelt sich dabei um folgende Algorithmen:

CHANGE wird in 4.5.1. besprochen,

MAXLIST in 4.5.2. und

INDMAX in 4.5.3.

Alle Algorithmen sind auf dem von ADTCOMP implementierten Datentyp OFFENELISTE definiert. CHANGE vertauscht zwei Elemente einer Offenen Liste, MAXLIST ermittelt das Maximum und INDMAX liefert den Index des maximalen Elementes einer Offenen Liste.

4.5.1. Der Algorithmus CHANGE

Zunächst die Spezifikation von CHANGE:

```
ALGORITHMUS-NAME: CHANGE;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE  OL,
                            INTEGER  INDEX1,
                            INTEGER  INDEX2;
                  SYSTEM:   INTEGER  COUNT := 1;

VORAUSSETZUNGEN:  (OL LIES INDEX1) /= 'UNDEF,
                  (OL LIES INDEX2) /= 'UNDEF;

ALGORITHMUS:      CHANGE (OL,INDEX1,INDEX2,COUNT) =
            {      COUNT = 1   UND   INDEX1 <= INDEX2
            ---> CHANGE((OL HINZU INDEX1 (OL LIES INDEX2)),
                        INDEX1,INDEX2,2);
                   COUNT = 2
            ---> CHANGE((OL HINZU (ADD1 INDEX2)
                 (OL LIES (ADD1 INDEX1))),INDEX1,INDEX2,3);
                   COUNT = 3
            ---> CHANGE((OL ENTFERN (ADD1 INDEX1)),
                        INDEX1,INDEX2,4);
                   COUNT = 4
             ---> (OL ENTFERN (ADD1 INDEX2));
                   INDEX1 = INDEX2
             ---> OL;
                   INDEX1 > INDEX2
             ---> CHANGE (OL,INDEX2,INDEX1,1)     }

KOMMENTAR: CHANGE VERTAUSCHT DIE ELEMENTE MIT DEN INDIZES
```

```
INDEX1 UND INDEX2 VON OFFENELISTE OL MITEINANDER.
WERT VON CHANGE IST DIE SO VERAENDERTE LISTE.
HILFSALGORITHMUS FÜR BUBBLE-SORT UND QUICK-SORT!!
```

CHANGE vertauscht die Elemente mit den Indizes INDEX1 und INDEX2 der Offenen Liste OL miteinander. Dabei ist es einerlei, in welcher Reihenfolge die Indizes angegeben werden (Produktionsregel 6) oder ob sie gleiche Werte besitzen (Produktionsregel 5).

Mit dem System-Parameter COUNT wird das eigentliche Vertauschen in vier Schritten gesteuert: zunächst werden die Elemente an die jeweils andere Stelle in OL geschrieben (Regel 1 und 2); dannach werden die alten Elemente an dieser Stelle gelöscht (Regel 3 und 4). Das etwas umständliche Vertauschen von Elementen ist bedingt durch die eine Offene Liste definierenden Operatoren. Man kann CHANGE auch als einen abgeleiteten Operator des Typs OFFENELISTE betrachten.
Falls eines der unter den Indizes angegebenen Elemente nicht existiert, der Leseoperator also UNDEF liefert, terminiert CHANGE (siehe Voraussetzungen)

CHANGE dient u.a. als Hilfsalgorithmus für Bubble-Sort und Quick-Sort.

Es folgt das Protokoll der Implementierung:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: CHA
ALGORITHMUS-NAME: (CHANGE)
PARAMETERTYPEN:    BENUTZER: ((OL . OFFENELISTE)
                              (INDEX1 . INTEGER)
                              (INDEX2 . INTEGER))
                   SYSTEM:   ((COUNT INTEGER 1))
VORAUSSETZUNGEN:   ((NEQ (OL LIES INDEX1) 'UNDEF)
                    (NEQ (OL LIES INDEX2) 'UNDEF))
R1 : ((CHANGE (((OL HINZU INDEX1 (OL LIES INDEX2))) ,
              (INDEX1) , (INDEX2) , (2))))
L2 : (((COUNT == 2)))
R2 : ((CHANGE (((OL HINZU (ADD1 INDEX2)
                          (OL LIES (ADD1 INDEX1)))) ,
              (INDEX1) , (INDEX2) , (3))))
L3 : (((COUNT == 3)))
R3 : ((CHANGE (((OL ENTFERN (ADD1 INDEX1))) ,
              (INDEX1) , (INDEX2) , (4))))
L4 : (((COUNT == 4)))
R4 : (((OL ENTFERN (ADD1 INDEX2))))
L5 : (((INDEX1 == INDEX2)))
R5 : ((OL))
L6 : (((INDEX1 > INDEX2)))
R6 : ((CHANGE ((OL) , (INDEX2) , (INDEX1) , (1))))
LINKESEITEN:   (L1 L2 L3 L4 L5 L6)
RECHTESEITEN:  (R1 R2 R3 R4 R5 R6)
KOMMENTAR:         (CHANGE VERTAUSCHT DIE ELEMENTE MIT DEN
```



```
                   INDIZES INDEX1 UND INDEX2 VON OFFENELISTE
                   OL MITEINANDER. WERT VON CHANGE IST DIE SO
                   VERAENDERTE LISTE. HILFSALGORITHMUS FUER
                   BUBBLE-SORT UND QUICKSORT)
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET

START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . (COUNT == 1))
ERGEBNIS     :      L1 VGL2 (EXPR . (INDEX1 <= INDEX2))
ERGEBNIS     :      L1 LOGOP (AND)
ERGEBNIS     :      L1 VGLLIST (VGL1 VGL2)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . (COUNT == 2))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . (COUNT == 3))
ERGEBNIS     :      L3 LOGOP (NONE)
ERGEBNIS     :      L3 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L4
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS     :      L4 VGL1 (EXPR . (COUNT == 4))
ERGEBNIS     :      L4 LOGOP (NONE)
ERGEBNIS     :      L4 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L5
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS     :      L5 VGL1 (EXPR . (INDEX1 == INDEX2))
ERGEBNIS     :      L5 LOGOP (NONE)
ERGEBNIS     :      L5 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L6
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS     :      L6 VGL1 (EXPR . (INDEX1 > INDEX2))
ERGEBNIS     :      L6 LOGOP (NONE)
ERGEBNIS     :      L6 VGLLIST (VGL1)
SCHRITT-1 ENDE
```

```
SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL1 (LINKS . COUNT)
ERGEBNIS    :      L1 VGL1 (RECHTS . 1)
ERGEBNIS    :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL1 (FORM . (EQ COUNT 1))

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL2

REGELANWENDUNG:  S2-2 AUF L1 , VERGLEICH  VGL2
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL2 (LINKS . INDEX1)
ERGEBNIS    :      L1 VGL2 (RECHTS . INDEX2)
ERGEBNIS    :      L1 VGL2 (REL . <=)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL2
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 VGL2 (FORM . (LESSEQP INDEX1 INDEX2))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L2 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL1 (LINKS . COUNT)
ERGEBNIS    :      L2 VGL1 (RECHTS . 2)
ERGEBNIS    :      L2 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 VGL1 (FORM . (EQ COUNT 2))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (LINKS . COUNT)
ERGEBNIS    :      L3 VGL1 (RECHTS . 3)
ERGEBNIS    :      L3 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 VGL1 (FORM . (EQ COUNT 3))

REGELANWENDUNG:  S2-1 AUF L4 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL1 (LINKS . COUNT)
ERGEBNIS    :      L4 VGL1 (RECHTS . 4)
```



```
ERGEBNIS    :      L4 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L4 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 VGL1 (FORM . (EQ COUNT 4))

REGELANWENDUNG:  S2-1 AUF L5 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL1 (LINKS . INDEX1)
ERGEBNIS    :      L5 VGL1 (RECHTS . INDEX2)
ERGEBNIS    :      L5 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L5 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 VGL1 (FORM . (EQ INDEX1 INDEX2))

REGELANWENDUNG:  S2-1 AUF L6 , VERGLEICH  VGL1

REGELANWENDUNG:  S2-2 AUF L6 , VERGLEICH  VGL1
---REGEL    : S2-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :      L6 VGL1 (LINKS . INDEX1)
ERGEBNIS    :      L6 VGL1 (RECHTS . INDEX2)
ERGEBNIS    :      L6 VGL1 (REL . >)

REGELANWENDUNG:  S2-9 AUF L6 , VERGLEICH  VGL1
---REGEL    : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :      L6 VGL1 (FORM . (GREATERP INDEX1 INDEX2))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS    :      L1 FINAL ((AND (EQ COUNT 1)
                                  (LESSEQP INDEX1 INDEX2)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS    :      L2 FINAL ((EQ COUNT 2))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS    :      L3 FINAL ((EQ COUNT 3))

REGELANWENDUNG:  S3-1 AUF L4
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L4
ERGEBNIS    :      L4 FINAL ((EQ COUNT 4))

REGELANWENDUNG:  S3-1 AUF L5
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L5
ERGEBNIS    :      L5 FINAL ((EQ INDEX1 INDEX2))

REGELANWENDUNG:  S3-1 AUF L6
```

```
---REGEL    : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L6
ERGEBNIS    :      L6 FINAL ((GREATERP INDEX1 INDEX2))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R1

REGELANWENDUNG:  S4-52 AUF R1

REGELANWENDUNG:  S4-53 AUF R1

REGELANWENDUNG:  S4-54 AUF R1

REGELANWENDUNG:  S4-55 AUF R1

REGELANWENDUNG:  S4-9 AUF R1
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG1 (EXPR . ((OL HINZU
                                    INDEX1
                                    (OL LIES INDEX2))))
ERGEBNIS    :      R1 ARG2 (EXPR . (INDEX1))
ERGEBNIS    :      R1 ARG3 (EXPR . (INDEX2))
ERGEBNIS    :      R1 ARG4 (EXPR . (2))
ERGEBNIS    :      R1 ARGLIST (ARG1 ARG2 ARG3 ARG4)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-52 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2

REGELANWENDUNG:  S4-9 AUF R2
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG1 (EXPR . ((OL HINZU
                                    (ADD1 INDEX2)
                                    (OL LIES
                                        (ADD1 INDEX1)))))
ERGEBNIS    :      R2 ARG2 (EXPR . (INDEX1))
ERGEBNIS    :      R2 ARG3 (EXPR . (INDEX2))
ERGEBNIS    :      R2 ARG4 (EXPR . (3))
ERGEBNIS    :      R2 ARGLIST (ARG1 ARG2 ARG3 ARG4)
```



```
REGELANWENDUNG:  S4-1 AUF R3
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3

REGELANWENDUNG:  S4-53 AUF R3

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG1 (EXPR . ((OL ENTFERN
                                        (ADD1 INDEX1))))
ERGEBNIS    :      R3 ARG2 (EXPR . (INDEX1))
ERGEBNIS    :      R3 ARG3 (EXPR . (INDEX2))
ERGEBNIS    :      R3 ARG4 (EXPR . (4))
ERGEBNIS    :      R3 ARGLIST (ARG1 ARG2 ARG3 ARG4)

REGELANWENDUNG:  S4-1 AUF R4

REGELANWENDUNG:  S4-2 AUF R4

REGELANWENDUNG:  S4-3 AUF R4

REGELANWENDUNG:  S4-4 AUF R4
---REGEL    : S4-4 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS    :      R4 EVALUIERT (INK-AUFRUF)
ERGEBNIS    :      R4 FORM ((OL ENTFERN (ADD1 INDEX2)))

REGELANWENDUNG:  S4-1 AUF R5

REGELANWENDUNG:  S4-2 AUF R5
---REGEL    : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS    :      R5 FORM (OL)
ERGEBNIS    :      R5 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R6
---REGEL    : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS    :      R6 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R6
---REGEL    : S4-54 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS    :      R6 KOPIERT3 (INDEX1)

REGELANWENDUNG:  S4-9 AUF R6
---REGEL    : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS    :      R6 ARG1 (EXPR . (OL))
ERGEBNIS    :      R6 ARG2 (EXPR . (INDEX2))
ERGEBNIS    :      R6 ARG3 (EXPR . (INDEX1))
ERGEBNIS    :      R6 ARG4 (EXPR . (1))
ERGEBNIS    :      R6 ARGLIST (ARG1 ARG2 ARG3 ARG4)
SCHRITT-4 ENDE
```

```
SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R1 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R1 , ARGUMENT ARG1
---REGEL    : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG1 (EXPR . (SETQ OL
                                    (OL HINZU
                                        INDEX1
                                        (OL LIES
                                            INDEX2))))

REGELANWENDUNG:  S5-8 AUF R1 , ARGUMENT ARG1
---REGEL    : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG1 (FORM . (OL HINZU
                                    INDEX1
                                    (OL LIES INDEX2)))

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG2
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG2 (EXPR . (SETQ INDEX1 INDEX1))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG2
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG3
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG3 (EXPR . (SETQ INDEX2 INDEX2))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG3
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R1 , ARGUMENT ARG4
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS    :      R1 ARG4 (EXPR . (SETQ COUNT 2))

REGELANWENDUNG:  S5-9 AUF R1 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG1
---REGEL    : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG1 (EXPR . (SETQ OL
                                    (OL HINZU
                                        (ADD1 INDEX2)
                                        (OL LIES
                                           (ADD1 INDEX1)))
                                ))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG1
```



```
---REGEL    : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG1 (FORM . (OL HINZU
                                    (ADD1 INDEX2)
                                    (OL LIES
                                        (ADD1 INDEX1))))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG2 (EXPR . (SETQ INDEX1 INDEX1))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG2
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG3 (EXPR . (SETQ INDEX2 INDEX2))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG3
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG4
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS    :      R2 ARG4 (EXPR . (SETQ COUNT 3))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG1

REGELANWENDUNG:  S5-31 AUF R3 , ARGUMENT ARG1
---REGEL    : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG1 (EXPR . (SETQ OL
                                    (OL ENTFERN
                                        (ADD1 INDEX1))))

REGELANWENDUNG:  S5-8 AUF R3 , ARGUMENT ARG1
---REGEL    : S5-8 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG1 (FORM . (OL ENTFERN (ADD1 INDEX1)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG2 (EXPR . (SETQ INDEX1 INDEX1))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG2
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG2 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL    : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS    :      R3 ARG3 (EXPR . (SETQ INDEX2 INDEX2))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
---REGEL    : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
```

```
ERGEBNIS     :       R3 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG4 (EXPR . (SETQ COUNT 4))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R6 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R6 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R6 , ARGUMENT ARG2
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 ARG2 (EXPR . (SETQ INDEX1 INDEX2))

REGELANWENDUNG:  S5-9 AUF R6 , ARGUMENT ARG2

REGELANWENDUNG:  S5-1 AUF R6 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 ARG3 (EXPR . (SETQ INDEX2 X3))

REGELANWENDUNG:  S5-9 AUF R6 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R6 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 ARG4 (EXPR . (SETQ COUNT 1))

REGELANWENDUNG:  S5-9 AUF R6 , ARGUMENT ARG4
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL ((OL HINZU INDEX1
                                    (OL LIES INDEX2))
                               (SETQ COUNT 2)
                               (CHANGE OL INDEX1 INDEX2 COUNT))

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 FINAL ((OL HINZU
                                  (ADD1 INDEX2)
                                  (OL LIES (ADD1 INDEX1)))
                               (SETQ COUNT 3)
                               (CHANGE OL INDEX1 INDEX2 COUNT))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 FINAL ((OL ENTFERN (ADD1 INDEX1))
```



```
                               (SETQ COUNT 4)
                               (CHANGE OL INDEX1 INDEX2 COUNT))

REGELANWENDUNG:  S6-1 AUF R4

REGELANWENDUNG:  S6-21 AUF R4

REGELANWENDUNG:  S6-22 AUF R4

REGELANWENDUNG:  S6-23 AUF R4
---REGEL     : S6-23 ERFOLGREICH ANGEWANDT AUF PRUEFLING R4
ERGEBNIS     :       R4 FINAL ((OL ENTFERN (ADD1 INDEX2)))

REGELANWENDUNG:  S6-1 AUF R5

REGELANWENDUNG:  S6-21 AUF R5
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R5
ERGEBNIS     :       R5 FINAL (OL)

REGELANWENDUNG:  S6-1 AUF R6
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R6
ERGEBNIS     :       R6 FINAL ((DUPLIZIERE INDEX1 X3)
                               (SETQ INDEX1 INDEX2)
                               (SETQ INDEX2 X3)
                               (SETQ COUNT 1)
                               (CHANGE OL INDEX1 INDEX2 COUNT))

START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((AND (EQ COUNT 1)
            (LESSEQP INDEX1 INDEX2))
        (OL HINZU INDEX1 (OL LIES INDEX2))
        (SETQ COUNT 2)
        (CHANGE OL INDEX1 INDEX2 COUNT))
      ((EQ COUNT 2)
        (OL HINZU (ADD1 INDEX2)
            (OL LIES (ADD1 INDEX1)))
        (SETQ COUNT 3)
        (CHANGE OL INDEX1 INDEX2 COUNT))
      ((EQ COUNT 3)
        (OL ENTFERN (ADD1 INDEX1))
        (SETQ COUNT 4)
        (CHANGE OL INDEX1 INDEX2 COUNT))
      ((EQ COUNT 4)
        (OL ENTFERN (ADD1 INDEX2)))
      ((EQ INDEX1 INDEX2)
        OL)
      ((GREATERP INDEX1 INDEX2)
        (DUPLIZIERE INDEX1 X3)
        (SETQ INDEX1 INDEX2)
        (SETQ INDEX2 X3)
        (SETQ COUNT 1)
        (CHANGE OL INDEX1 INDEX2 COUNT))
```

```
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-4 S4-54 S4-9
 S5-1 S5-31 S5-8 S5-9 S6-1 S6-21 S6-23)

REGELANWENDUNGSFAKTOR:  0.627273

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ LESSEQP GREATERP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<CHANGE
 <LAMBDA (OL INDEX1 INDEX2 COUNT)
  (COND
   ((EQ (QUOTE OFFENELISTE)
        (GETP OL (QUOTE STRUKTUR)))
    (COND
     ((OR (TYPEP INDEX1 22)
          (TYPEP INDEX1 23))
      (COND
       ((OR (TYPEP INDEX2 22)
            (TYPEP INDEX2 23))
        (COND
         ((GETD (QUOTE DUPLIZIERE))
          (COND
           ((GETD (QUOTE LESSEQP))
            (COND
             ((NEQ <EVAL (SUBST OL (QUOTE OL)
                                   (QUOTE (OL LIES INDEX2>
                         (QUOTE UNDEF))
              (COND
               ((NEQ <EVAL (SUBST OL (QUOTE OL)
                                     (QUOTE (OL LIES INDEX1>
                           (QUOTE UNDEF))
                (SETQ COUNT 1)
                <RPLACA (QUOTE CHANGE)
                  (DSUBST OL (QUOTE OL)
                    (DSUBST
                      <QUOTE
                        (RETEVAL
                          (STKPOS (QUOTE CHANGE)
                                  -1)
                          (QUOTE (APPLY CHANGE NIL>
                      (QUOTE (CHANGE OL INDEX1 INDEX2
                                                 COUNT))
                      (COPY (GETP (QUOTE CHANGE)
                                  (QUOTE ALGORITHMUS>
```



```
                    (APPLY CHANGE NIL))
                  (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                             (QUOTE (NEQ (OL LIES INDEX1)
                                         (QUOTE UNDEF)))
                             "NICHT ERFUELLT")
                     (BREAK1 NIL T)
                     NIL)))
                (T (PRINTL "***ERROR*** VORAUSSETZUNG"
                           (QUOTE (NEQ (OL LIES INDEX2)
                                       (QUOTE UNDEF)))
                           "NICHT ERFUELLT")
                   (BREAK1 NIL T)
                   NIL)))
              (T (PRINTL "***ERROR*** HILFSFUNKTION"
                         (QUOTE LESSEQP)
                         "UNDEFINIERT")
                 (BREAK1 NIL T)
                 NIL)))
            (T (PRINTL "***ERROR*** HILFSFUNKTION"
                       (QUOTE DUPLIZIERE)
                       "UNDEFINIERT")
               (BREAK1 NIL T)
               NIL)))
          (T (PRINTL "***ERROR***" INDEX2
                     "NICHT VOM TYP INTEGER")
             (BREAK1 NIL T)
             NIL)))
        (T (PRINTL "***ERROR***" INDEX1
                   "NICHT VOM TYP INTEGER")
           (BREAK1 NIL T)
           NIL)))
     (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
                (QUOTE OFFENELISTE))
        (BREAK1 NIL T)
        NIL>>
LISP-PHASE BEENDET

Abschließend der fertig implementierte Algorithmus CHANGE:

(DEFINEQ

<CHANGE
  <LAMBDA (OL INDEX1 INDEX2 COUNT)
          .
          .
          .
    siehe Ergebnis der LISP-Phase
          .
          .
          .    >>
)
  <PUTPROPS CHANGE KOMMENTAR
          (CHANGE VERTAUSCHT DIE ELEMENTE MIT DEN INDIZES
                  INDEX1 UND INDEX2 VON OFFENELISTE OL
                  MITEINANDER. WERT VON CHANGE IST DIE SO
                  VERAENDERTE LISTE. HILFSALGORITHMUS FUER
```

```
                BUBBLE-SORT UND QUICK-SORT)
          RELATION
          (STANDARD ((== . EQ)
                     (/= . NEQ)
                     (%< . LESSP)
                     (%> . GREATERP)
                     (%<= . LESSEQP)
                     (%>= . GREATEREQP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
                .
                .
                .
    siehe Ergebnis der Automatischen Codierung
                .
                .
                .    >
STOP
```

4.5.2. Der Algorithmus MAXLIST

MAXLIST liefert als Wert das maximale Element der Offenen Liste OL. Ist OL leer, so gilt (MAXLIST 'OL) = UNDEF.
U.a. wird MAXLIST als Hilfsalgorithmus von Straight-Selection-Sort aufgerufen.

MAXLIST ist folgendermaßen spezifiziert:

```
ALGORITHMUS-NAME: MAXLIST;

PARAMETERTYPEN:    BENUTZER: OFFENELISTE-INTEGER  OL;
                   SYSTEM:   INTEGER  INDEX := 1,
                             INTEGER  MAXIMUM := (OL LIES 1);

ALGORITHMUS:       MAXLIST (OL,INDEX,MAXIMUM) =
             {     (OL LIES INDEX) = UNDEF
              ---> MAXIMUM;
                   (OL LIES INDEX) >= MAXIMUM
              ---> MAXLIST (OL,(ADD1 INDEX),(OL LIES INDEX));
                   (OL LIES INDEX) < MAXIMUM
              ---> MAXLIST (OL,(ADD1 INDEX),MAXIMUM)     }

KOMMENTAR:    MAXLIST LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
              OFFENELISTE OL UND HAT ALS WERT DAS GROESSTE
              ELEMENT VON OL. OL BLEIBT ERHALTEN.
              HILFSALGORITHMUS FUER STRAIGHT-SELECTION-SORT!!
```


Es folgt das Implementierungprotokoll:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: MAX
ALGORITHMUS-NAME: (MAXLIST)
PARAMETERTYPEN:    BENUTZER: ((OL . OFFENELISTE-INTEGER))
                   SYSTEM:   ((INDEX INTEGER 1)
                              (MAXIMUM INTEGER (OL LIES 1)))
ALGORITHMUS:       PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES INDEX) == UNDEF)))
R1 : ((MAXIMUM))
L2 : ((((OL LIES INDEX) >= MAXIMUM)))
R2 : ((MAXLIST ((OL) , ((ADD1 INDEX)) , ((OL LIES INDEX)))))
L3 : ((((OL LIES INDEX) < MAXIMUM)))
R3 : ((MAXLIST ((OL) , ((ADD1 INDEX)) , (MAXIMUM))))
LINKESEITEN:  (L1 L2 L3)
RECHTESEITEN: (R1 R2 R3)
KOMMENTAR:         (MAXLIST LIEST DIE MIT GANZEN ZAHLEN
                    GEFUELLTE OFFENELISTE OL UND HAT ALS WERT
                    DAS GROESSTE ELEMENT VON OL. OL BLEIBT
                    ERHALTEN. HILFSALGORITHMUS FUER
                    STRAIGHT-SELECTION-SORT)
ALG-EINGABE BEENDET.
RELATION       = STANDARD
RELATIONSSATZ = ((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET


START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES INDEX) == UNDEF))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES INDEX)
                                          >= MAXIMUM))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES INDEX)
                                          < MAXIMUM))
ERGEBNIS     :      L3 LOGOP (NONE)
ERGEBNIS     :      L3 VGLLIST (VGL1)
SCHRITT-1 ENDE
```

```
SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L1 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ (OL LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L2 VGL1 (RECHTS . MAXIMUM)
ERGEBNIS     :      L2 VGL1 (REL . >=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (GREATEREQP (OL LIES INDEX)
                                                 MAXIMUM))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L3 VGL1 (RECHTS . MAXIMUM)
ERGEBNIS     :      L3 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (LESSP (OL LIES INDEX)
                                           MAXIMUM))
SCHRITT-2 ENDE


SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ (OL LIES INDEX)
                                  (QUOTE UNDEF)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 FINAL ((GREATEREQP (OL LIES INDEX)
                                          MAXIMUM))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 FINAL ((LESSP (OL LIES INDEX) MAXIMUM))
SCHRITT-3 ENDE
```



```
SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :      R1 FORM (MAXIMUM)
ERGEBNIS     :      R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2
---REGEL     : S4-55 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 KOPIERT3 (INDEX)

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (OL))
ERGEBNIS     :      R2 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :      R2 ARG3 (EXPR . ((OL LIES INDEX)))
ERGEBNIS     :      R2 ARGLIST (ARG1 ARG2 ARG3)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (OL))
ERGEBNIS     :      R3 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :      R3 ARG3 (EXPR . (MAXIMUM))
ERGEBNIS     :      R3 ARGLIST (ARG1 ARG2 ARG3)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG1 (FORM .)
```

```
REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 ARG3 (EXPR . (SETQ MAXIMUM
                                    (OL LIES X3)))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (EXPR . (SETQ MAXIMUM MAXIMUM))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM .)
SCHRITT-5 ENDE
```



```
SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL (MAXIMUM)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE INDEX X3)
                              (SETQ INDEX (ADD1 INDEX))
                              (SETQ MAXIMUM (OL LIES X3))
                              (MAXLIST OL INDEX MAXIMUM))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ INDEX (ADD1 INDEX))
                              (MAXLIST OL INDEX MAXIMUM))

START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ (OL LIES INDEX)
           (QUOTE UNDEF))
       MAXIMUM)
      ((GREATEREQP (OL LIES INDEX)
                   MAXIMUM)
       (DUPLIZIERE INDEX X3)
       (SETQ INDEX (ADD1 INDEX))
       (SETQ MAXIMUM (OL LIES X3))
       (MAXLIST OL INDEX MAXIMUM))
      ((LESSP (OL LIES INDEX)
              MAXIMUM)
       (SETQ INDEX (ADD1 INDEX))
       (MAXLIST OL INDEX MAXIMUM))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-55 S4-9
 S5-1 S5-2 S5-31 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.711111

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ GREATEREQP LESSP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET
```

```
START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<MAXLIST
 <LAMBDA (OL INDEX MAXIMUM)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((GETD (QUOTE DUPLIZIERE))
      (COND
       ((GETD (QUOTE GREATEREQP))
        (SETQ INDEX 1)
        <SETQ MAXIMUM (EVAL (SUBST OL (QUOTE OL)
                                   (QUOTE (OL LIES 1>
        <RPLACA (QUOTE MAXLIST)
          (DSUBST OL (QUOTE OL)
            (DSUBST <QUOTE (RETEVAL (STKPOS (QUOTE MAXLIST)
                                            -1)
                                    (QUOTE (APPLY MAXLIST
                                                  NIL>
                    (QUOTE (MAXLIST OL INDEX MAXIMUM))
                    (COPY (GETP (QUOTE MAXLIST)
                                (QUOTE ALGORITHMUS>
        (APPLY MAXLIST NIL))
       (T (PRINTL "***ERROR*** HILFSFUNKTION"
                  (QUOTE GREATEREQP)
                  "UNDEFINIERT")
          (BREAK1 NIL T)
          NIL)))
     (T (PRINTL "***ERROR*** HILFSFUNKTION"
                (QUOTE DUPLIZIERE)
                "UNDEFINIERT")
        (BREAK1 NIL T)
        NIL)))
   (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
              (QUOTE OFFENELISTE-INTEGER))
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET

Die Implementation von MAXLIST sieht folgendermaßen aus:

(DEFINEQ

<MAXLIST
  <LAMBDA (OL INDEX MAXIMUM)
            .
            .
            .
     siehe Ergebnis der LISP-Phase
            .
            .
            .    >>
)
```



```
  <PUTPROPS MAXLIST KOMMENTAR
            (MAXLIST LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
            OFFENELISTE OL UND HAT ALS WERT DAS GROESSTE
            ELEMENT VON OL. OL BLEIBT ERHALTEN.
            HILFSALGORITHMUS FUER STRAIGHT-SELECTION-SORT)
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                    .
                    .
                    .
    siehe Ergebnis der Automatischen Codierung
                    .
                    .
                    .    >
STOP
```

4.5.3. Der Algorithmus INDMAX

INDMAX arbeitet, genau wie MAXLIST, auf dem von ADTCOMP implementierten Datentyp OFFENELISTE-INTEGER, liefert jedoch als Wert den Index des maximalen Listenelements.

Hier die Spezifikation von INDMAX:

```
ALGORITHMUS-NAME: INDMAX;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-INTEGER  OL;
                  SYSTEM:   INTEGER  INDEX := 1,
                            INTEGER  MAXIMUM := (OL LIES 1),
                            INTEGER  INDEXMAXIMUM := 0;

ALGORITHMUS:      INDMAX (OL,INDEX,MAXIMUM,INDEXMAXIMUM) =
                  {   (OL LIES INDEX) = UNDEF
                    ---> INDEXMAXIMUM;
                      (OL LIES INDEX) >= MAXIMUM
                    ---> INDMAX (OL,(ADD1 INDEX),(OL LIES INDEX),
                                                  INDEX);
                      (OL LIES INDEX) < MAXIMUM
                    ---> INDMAX (OL,(ADD1 INDEX),MAXIMUM,
                                                  INDEXMAXIMUM); }

KOMMENTAR:        INDMAX LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
                  OFFENELISTE OL UND HAT ALS WERT DEN INDEX DES
```

```
MAXIMUMS VON OL. OL BLEIBT ERHALTEN. HILFS-
ALGORITHMUS FUER STRAIGHT-SELECTION-SORT!!
```

Falls OL die leere Liste ist, liefert INDMAX als Wert 0.
U.a. dient INDMAX als Hilfsalgorithmus für Straight-Selection-Sort.

Es folgt das Protokoll der Implementierung:

```
START EINLESE-PHASE
ALG-EINGABE UEBER DATEI: IND
ALGORITHMUS-NAME: (INDMAX)
PARAMETERTYPEN:   BENUTZER: ((OL . OFFENELISTE-INTEGER))
                  SYSTEM:   ((INDEX INTEGER 1)
                             (MAXIMUM INTEGER (OL LIES 1))
                             (INDEXMAXIMUM INTEGER 0))
ALGORITHMUS:      PARSING - AUFSPALTUNG DER PRODUKTIONSREGELN:
L1 : ((((OL LIES INDEX) == UNDEF)))
R1 : ((INDEXMAXIMUM))
L2 : ((((OL LIES INDEX) >= MAXIMUM)))
R2 : ((INDMAX ((OL) , ((ADD1 INDEX)) , ((OL LIES INDEX)) ,
                                       (INDEX))))
L3 : ((((OL LIES INDEX) < MAXIMUM)))
R3 : ((INDMAX ((OL) , ((ADD1 INDEX)) , (MAXIMUM) ,
                                       (INDEXMAXIMUM))))
LINKESEITEN:  (L1 L2 L3)
RECHTESEITEN: (R1 R2 R3)
KOMMENTAR:        (INDMAX LIEST DIE MIT GANZEN ZAHLEN
                   GEFUELLTE OFFENELISTE OL UND HAT ALS WERT
                   DEN INDEX DES MAXIMUMS VON OL. OL BLEIBT
                   ERHALTEN. HILFSALGORITHMUS FUER
                   STRAIGHT-SELECTION-SORT)
ALG-EINGABE BEENDET.
RELATION      = STANDARD
RELATIONSSATZ = (((== . EQ)(/= . NEQ)(< . LESSP)(> . GREATERP)
                 (<= . LESSEQP)(>= . GREATEREQP))
EINLESE-PHASE BEENDET

START ALG-KONSTRUKTIONS-PHASE

SCHRITT-1 START

REGELANWENDUNG:  S1-1 AUF L1
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (EXPR . ((OL LIES INDEX) == UNDEF))
ERGEBNIS     :      L1 LOGOP (NONE)
ERGEBNIS     :      L1 VGLLIST (VGL1)

REGELANWENDUNG:  S1-1 AUF L2
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (EXPR . ((OL LIES INDEX)
                                     >= MAXIMUM))
ERGEBNIS     :      L2 LOGOP (NONE)
ERGEBNIS     :      L2 VGLLIST (VGL1)
```



```
REGELANWENDUNG:  S1-1 AUF L3
---REGEL     : S1-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (EXPR . ((OL LIES INDEX)
                                     < MAXIMUM))
ERGEBNIS     :      L3 LOGOP (NONE)
ERGEBNIS     :      L3 VGLLIST (VGL1)
SCHRITT-1 ENDE

SCHRITT-2 START

REGELANWENDUNG:  S2-1 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L1 VGL1 (RECHTS . (QUOTE UNDEF))
ERGEBNIS     :      L1 VGL1 (REL . ==)

REGELANWENDUNG:  S2-9 AUF L1 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 VGL1 (FORM . (EQ (OL LIES INDEX)
                                        (QUOTE UNDEF)))

REGELANWENDUNG:  S2-1 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L2 VGL1 (RECHTS . MAXIMUM)
ERGEBNIS     :      L2 VGL1 (REL . >=)

REGELANWENDUNG:  S2-9 AUF L2 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
ERGEBNIS     :      L2 VGL1 (FORM . (GREATEREQP (OL LIES INDEX)
                                                 MAXIMUM))

REGELANWENDUNG:  S2-1 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (LINKS . (OL LIES INDEX))
ERGEBNIS     :      L3 VGL1 (RECHTS . MAXIMUM)
ERGEBNIS     :      L3 VGL1 (REL . <)

REGELANWENDUNG:  S2-9 AUF L3 , VERGLEICH  VGL1
---REGEL     : S2-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :      L3 VGL1 (FORM . (LESSP (OL LIES INDEX)
                                           MAXIMUM))
SCHRITT-2 ENDE

SCHRITT-3 START

REGELANWENDUNG:  S3-1 AUF L1
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L1
ERGEBNIS     :      L1 FINAL ((EQ (OL LIES INDEX)
                                  (QUOTE UNDEF)))

REGELANWENDUNG:  S3-1 AUF L2
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L2
```

```
ERGEBNIS     :       L2 FINAL ((GREATEREQP (OL LIES INDEX)
                                           MAXIMUM))

REGELANWENDUNG:  S3-1 AUF L3
---REGEL     : S3-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING L3
ERGEBNIS     :       L3 FINAL ((LESSP (OL LIES INDEX) MAXIMUM))
SCHRITT-3 ENDE


SCHRITT-4 START

REGELANWENDUNG:  S4-1 AUF R1

REGELANWENDUNG:  S4-2 AUF R1
---REGEL     : S4-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FORM (INDEXMAXIMUM)
ERGEBNIS     :       R1 EVALUIERT (PARAMPASS)

REGELANWENDUNG:  S4-1 AUF R2
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R2

REGELANWENDUNG:  S4-53 AUF R2

REGELANWENDUNG:  S4-54 AUF R2

REGELANWENDUNG:  S4-55 AUF R2
---REGEL     : S4-55 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 KOPIERT3 (INDEX)
ERGEBNIS     :       R2 KOPIERT4 (INDEX)

REGELANWENDUNG:  S4-9 AUF R2
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (EXPR . (OL))
ERGEBNIS     :       R2 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :       R2 ARG3 (EXPR . ((OL LIES INDEX)))
ERGEBNIS     :       R2 ARG4 (EXPR . (INDEX))
ERGEBNIS     :       R2 ARGLIST (ARG1 ARG2 ARG3 ARG4)

REGELANWENDUNG:  S4-1 AUF R3
---REGEL     : S4-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 EVALUIERT (REKURSIV)

REGELANWENDUNG:  S4-51 AUF R3

REGELANWENDUNG:  S4-53 AUF R3

REGELANWENDUNG:  S4-54 AUF R3

REGELANWENDUNG:  S4-55 AUF R3

REGELANWENDUNG:  S4-9 AUF R3
---REGEL     : S4-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (OL))
```



```
ERGEBNIS     :       R3 ARG2 (EXPR . ((ADD1 INDEX)))
ERGEBNIS     :       R3 ARG3 (EXPR . (MAXIMUM))
ERGEBNIS     :       R3 ARG4 (EXPR . (INDEXMAXIMUM))
ERGEBNIS     :       R3 ARGLIST (ARG1 ARG2 ARG3 ARG4)
SCHRITT-4 ENDE


SCHRITT-5 START

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG2

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R2 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-2 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-31 AUF R2 , ARGUMENT ARG3
---REGEL     : S5-31 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG3 (EXPR . (SETQ MAXIMUM
                                           (OL LIES X3)))

REGELANWENDUNG:  S5-8 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG3

REGELANWENDUNG:  S5-1 AUF R2 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :       R2 ARG4 (EXPR . (SETQ INDEXMAXIMUM X4))

REGELANWENDUNG:  S5-9 AUF R2 , ARGUMENT ARG4

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (EXPR . (SETQ OL OL))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG1
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :       R3 ARG1 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG2
```

```
REGELANWENDUNG:  S5-2 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-2 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (LISP-AUFRUF ADD1)

REGELANWENDUNG:  S5-5 AUF R3 , ARGUMENT ARG2
---REGEL     : S5-5 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG2 (FORM . (SETQ INDEX (ADD1 INDEX)))

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (EXPR . (SETQ MAXIMUM MAXIMUM))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG3
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG3 (FORM .)

REGELANWENDUNG:  S5-1 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (EXPR . (SETQ INDEXMAXIMUM
                                    INDEXMAXIMUM))

REGELANWENDUNG:  S5-9 AUF R3 , ARGUMENT ARG4
---REGEL     : S5-9 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 ARG4 (FORM .)
SCHRITT-5 ENDE


SCHRITT-6 START

REGELANWENDUNG:  S6-1 AUF R1

REGELANWENDUNG:  S6-21 AUF R1
---REGEL     : S6-21 ERFOLGREICH ANGEWANDT AUF PRUEFLING R1
ERGEBNIS     :       R1 FINAL (INDEXMAXIMUM)

REGELANWENDUNG:  S6-1 AUF R2
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R2
ERGEBNIS     :      R2 FINAL ((DUPLIZIERE INDEX X4)
                              (DUPLIZIERE INDEX X3)
                              (SETQ INDEX (ADD1 INDEX))
                              (SETQ MAXIMUM (OL LIES X3))
                              (SETQ INDEXMAXIMUM X4)
                              (INDMAX OL INDEX MAXIMUM
                                      INDEXMAXIMUM))

REGELANWENDUNG:  S6-1 AUF R3
---REGEL     : S6-1 ERFOLGREICH ANGEWANDT AUF PRUEFLING R3
ERGEBNIS     :      R3 FINAL ((SETQ INDEX (ADD1 INDEX))
                              (INDMAX OL INDEX MAXIMUM
                                      INDEXMAXIMUM))
```



```
START KOMPOSITION
ERGEBNIS DER AUTOMATISCHEN CODIERUNG:

  <NLAMBDA NIL
    (COND
      ((EQ (OL LIES INDEX)
           (QUOTE UNDEF))
        INDEXMAXIMUM)
      ((GREATEREQP (OL LIES INDEX)
                   MAXIMUM)
        (DUPLIZIERE INDEX X4)
        (DUPLIZIERE INDEX X3)
        (SETQ INDEX (ADD1 INDEX))
        (SETQ MAXIMUM (OL LIES X3))
        (SETQ INDEXMAXIMUM X4)
        (INDMAX OL INDEX MAXIMUM INDEXMAXIMUM))
      ((LESSP (OL LIES INDEX)
              MAXIMUM)
        (SETQ INDEX (ADD1 INDEX))
        (INDMAX OL INDEX MAXIMUM INDEXMAXIMUM))
      (T (FEHLER 10>

VERWENDET WURDEN DIE REGELN
(S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-55 S4-9
 S5-1 S5-2 S5-31 S5-5 S5-9 S6-1 S6-21)

REGELANWENDUNGSFAKTOR:  0.686274

ERKANNT WURDEN DIE HILFSFUNKTIONEN
(EQ GREATEREQP LESSP DUPLIZIERE)

SCHRITT-6 ENDE

ALG-KONSTRUKTIONS-PHASE BEENDET


START LISP-PHASE

ERGEBNIS DER LISP-PHASE:

<INDMAX
 <LAMBDA (OL INDEX MAXIMUM INDEXMAXIMUM)
  (COND
   ((EQ (QUOTE OFFENELISTE-INTEGER)
        (GETP OL (QUOTE PASSIVSTRUKTUR)))
     (COND
      ((GETD (QUOTE DUPLIZIERE))
        (COND
         ((GETD (QUOTE GREATEREQP))
           (SETQ INDEX 1)
           <SETQ MAXIMUM (EVAL (SUBST OL (QUOTE OL)
                                         (QUOTE (OL LIES 1>
           (SETQ INDEXMAXIMUM 0)
           <RPLACA (QUOTE INDMAX)
             (DSUBST OL (QUOTE OL)
               (DSUBST <QUOTE (RETEVAL (STKPOS (QUOTE INDMAX)
```

```
                                        -1)
                                    (QUOTE (APPLY INDMAX
                                                  NIL>
                            (QUOTE (INDMAX OL INDEX MAXIMUM
                                                  INDEXMAXIMUM))
                            (COPY (GETP (QUOTE INDMAX)
                                        (QUOTE ALGORITHMUS>
          (APPLY INDMAX NIL))
         (T (PRINTL "***ERROR*** HILFSFUNKTION"
                    (QUOTE GREATEREQP)
                    "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
      (T (PRINTL "***ERROR*** HILFSFUNKTION"
                 (QUOTE DUPLIZIERE)
                 "UNDEFINIERT")
         (BREAK1 NIL T)
         NIL)))
   (T (PRINTL "***ERROR***" OL "NICHT VOM TYP"
              (QUOTE OFFENELISTE-INTEGER))
      (BREAK1 NIL T)
      NIL>>
LISP-PHASE BEENDET
```

Schließlich die arbeitsfähige Implementierung von INDMAX:

```
(DEFINEQ

<INDMAX
  <LAMBDA (OL INDEX MAXIMUM INDEXMAXIMUM)
            .
            .
            .
     siehe Ergebnis der LISP-Phase
            .
            .
            .     >>
)
  <PUTPROPS INDMAX KOMMENTAR
            (INDMAX LIEST DIE MIT GANZEN ZAHLEN GEFUELLTE
                    OFFENELISTE OL UND HAT ALS WERT DEN INDEX
                    DES MAXIMUMS VON OL. OL BLEIBT ERHALTEN.
                    HILFSALGORITHMUS FUER
                    STRAIGHT-SELECTION-SORT)
            RELATION
            (STANDARD ((=  . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                .
                .
```



```
                .
     siehe Ergebnis der Automatischen Codierung
                .
                .
                .     >
STOP
```

4.6. Anwendungsbeispiel: ein Personaldatenverwaltungssystem

Das ALGCOMP-System ist in der Lage, aus Eingabespezifikationen abstrakter Algorithmen lauffähige Programme zu erzeugen. Wie man dies praktisch einsetzen kann, soll im folgenden anhand eines (elementaren) Personaldatenverwaltungssystems (PVS) demonstriert werden. Dabei sind bestimmte Fragestellungen der Personaldatenmanipulation in abstrakte Algorithmen über parametrisierten Datentypen transformiert und dem ALGCOMP-System übergeben worden. Die Spezifikationen dieser Algorithmen sowie die erzeugten Implementierungen sind jeweils dokumentiert, die Implementierungsprotokolle wurden aus Platzgründen unterdrückt.

Der einem PVS zugrundeliegende Datentyp soll der parametrisierte Typ OFFENELISTE-FLEXVERBUND-LISPATOM sein, so wie er durch das APE-System erzeugbar ist.
In diesem Typ werden Personaldaten einer Anzahl von Personen verwaltet. Die Daten einer einzelnen Person wiederum werden in dem parametrisierten Typ FLEXVERBUND-LISPATOM abgelegt. Zugriff zu den Elementen der Typen ist nur über die jeweils spezifischen Operationen möglich. (Zur Erläuterung der OFFENENLISTE und des FLEXVERBUNDES: siehe [EKT 79]).

Zu Beginn seien zwei Personaldateien PD1 und PD2 existent. In ihnen seien drei bzw. vier Personendateien geführt, DAT11,DAT12 und DAT13 sowie DAT21, DAT22, DAT23 und DAT24. Jede der Personendateien DATij habe Einträge für NAME, VORNAME, WOHNORT, STRASSE, PNR (=Personalnummer), ALTER, GROESSE und gegebenenfalls GEWERKSCHAFT, falls die betreffende Person Mitglied einer Gewerkschaft ist. Die Einträge müssen, gemäß dem zugrundeliegenden parametrisierten Datentyp, vom Typ LISPATOM sein.

Folgende Bindungen liegen vor:

```
(RPAQQ PD1 ((DAT13)
            DAT11 DAT12 DAT13))
(RPAQQ PD2 ((DAT24)
            DAT21 DAT22 DAT23 DAT24))
```

[PDi : Typ OFFENEELISTE-FLEXVERBUND-LISPATOM]

```
(RPAQQ DAT11 (NIL (NAME . OLTHOFF)
                  (VORNAME . WALTER)
                  (WOHNORT . BONN-OBERKASSEL)
                  (STRASSE . KOENIGSWINTERER STR 809)
                  (GEWERKSCHAFT . IGM)
                  (ALTER . 25)
                  (GROESSE . 194)
                  (PNR . 2214)))
(RPAQQ DAT12 (NIL (NAME . BARTELS)
                  (VORNAME . ULRICH)
                  (WOHNORT . BONN-HOHOLZ)
                  (STRASSE . GIELGEN STR 27)
```



```
                  (ALTER . 27)
                  (GROESSE . 195)
                  (PNR . 6312)))
(RPAQQ DAT13 (NIL (NAME . BURCK)
                  (VORNAME . VOLKER)
                  (WOHNORT . BONN OBERKASSEL)
                  (STRASSE . KOENIGSWINTERER STR 809)
                  (ALTER . 29)
                  (GEWERKSCHAFT . OETV)
                  (GROESSE . 188)
                  (PNR . 4211)))
(RPAQQ DAT21 (NIL (NAME . KRAUSE)
                  (VORNAME . MICHAEL)
                  (WOHNORT . BONN-OBERKASSEL)
                  (STRASSE . KOENIGSWINTERER STR 809)
                  (GROESSE . 178)
                  (ALTER . 28)
                  (GEWERKSCHAFT . BSE)
                  (PNR . 4322)))
(RPAQQ DAT22 (NIL (NAME . DREHMANN)
                  (VORNAME . SABINE)
                  (WOHNORT . BONN-OBERKASSEL)
                  (STRASSE . KOENIGSWINTERER STR 809)
                  (GROESSE . 183)
                  (ALTER . 26)
                  (PNR . 1180)))
(RPAQQ DAT23 (NIL (NAME . GROENEVELD)
                  (VORNAME . ANGELIKA)
                  (WOHNORT . OLDENBURG)
                  (STRASSE . JOHANN-JUSTUS-WEG 136)
                  (ALTER . 23)
                  (GROESSE . 180)
                  (PNR . 2681)))
(RPAQQ DAT24 (NIL (NAME . WILBERTS)
                  (VORNAME . BARBARA)
                  (WOHNORT . BONN)
                  (STRASSE . POPPELSDORFER ALLEE 54)
                  (ALTER . 24)
                  (GROESSE . 189)
                  (GEWERKSCHAFT . GEW)
                  (PNR . 4818)))
```

[DATij : Typ FLEXVERBUND-LISPATOM]

(Die Verwendung der personenbezogenen Daten ist von den betreffenden Trägern gebilligt worden...)

Die erste auf PV-Dateien PDi ausführbare Aktion wäre eine Sortierung. Dafür sind im Abschnitt 4.3. Sortieralgorithmen angegeben worden, die auch hier verwendet werden können. Allerdings muß jetzt zusätzlich gesagt werden, nach welchen Kriterien sortiert werden soll. Möglich sind beispielsweise der Name einer Person oder ihre Personalnummer. Wenn sortiert wird, wird in einer PDi die Reihenfolge der DATij entsprechend der Sortierrelation verändert. Um zu wissen, ob z.B. DATi1 vor DATi2 gemäß dem Kriterium NAME oder PNR steht, muß auf den je-

weiligen FLEXVERBUND-LISPATOM zugegriffen werden, und ein Vergleich der unter den entsprechenden Indikatoren gebundenen Werte liefert dann den verlangten Wahrheitswert.

Dies zu spezifizieren ist umständlich.

Deswegen wird für die beiden angegebenen Kriterien ein eigener Relationssatz PVSREL-PNR und PVSREL-NAM bereitgestellt, durch den genau diese Zugriffe für die verschiedenen Möglichkeiten des Tests ('kleiner', 'größer' etc.) realisiert werden. In der Spezifikation wird dann der gewünschte Relationssatz angegeben, und ALGCOMP verknüpft z.B. das '<'-Zeichen mit der Prädikat-Funktion KLEINER-PNRP (Kriterium PNR) oder KLEINER-NAMP (Kriterium NAME) und substituiert diese in den erzeugten Code. ((KLEINER-PNRP A B) ist genau dann T, wenn (LESSP (A LIES 'PNR) (B LIES 'PNR)) T ist.)

Die Relationssätze (Liste der Funktionsnamen) werden in der Datei LISP.DATA.PM-ALG.nn mitgeführt, da während der Implementierung auf den Namen zugegriffen werden muß. Die Funktionen selbst stehen in der Datei LISP.DATA.ALG-EX.nn, da sie nur bei Ausführung des Algorithmus als Funktion referiert werden.

Mit dieser Vereinfachung läßt sich die Spezifikation für BUBBLE-SORT übernehmen bis auf die Angabe eines anderen Relationssatzes, eines anderen Benutzerdatentyps und eine umgekehrte Sortierrichtung:

```
ALGORITHMUS-NAME: BUBBLE-PNR;

PARAMETERTYPEN:  BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                           OFL;
                 SYSTEM:   INTEGER  INDEX := 1,
                           BOOL     BUBBLE? := FALSE;
RELATION:          PVSREL-PNR;
HILFSALGORITHMEN: CHANGE;

ALGORITHMUS:  BUBBLE-PNR (OFL,INDEX,BUBBLE?) =
          {    (OFL LIES (ADD1 INDEX)) /= UNDEF   UND
               (OFL LIES INDEX) > (OFL LIES (ADD1 INDEX))
           ---> BUBBLE-PNR (OFL,(ADD1 INDEX),BUBBLE?);
               (OFL LIES (ADD1 INDEX)) /= UNDEF    UND
               (OFL LIES INDEX) <= (OFL LIES (ADD1 INDEX))
           ---> BUBBLE-PNR
                  (CHANGE (OFL,INDEX,(ADD1 INDEX)),
                  (ADD1 INDEX),TRUE);
               (OFL LIES (ADD1 INDEX)) = UNDEF   UND
               BUBBLE? = TRUE
           ---> BUBBLE-PNR (OFL,1,FALSE);
               (OFL LIES (ADD1 INDEX)) = UNDEF  UND
               BUBBLE? = FALSE
           ---> OFL          }

KOMMENTAR:       !!
```


Die Implementierung spiegelt die Veränderungen wieder:

```
(DEFINEQ
<BUBBLE-PNR
 <LAMBDA (OFL INDEX BUBBLE?)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((GETD (QUOTE GLEICH-PNRP))
      (COND
       ((GETD (QUOTE KLEINERGLEICH-PNRP))
        (COND
         ((GETD (QUOTE GROESSER-PNRP))
          (COND
           ((GETD (QUOTE UNGLEICH-PNRP))
            (COND
             ((GETD (QUOTE CHANGE))
              (SETQ INDEX 1)
              (SETQ BUBBLE? NIL)
              <RPLACA (QUOTE BUBBLE-PNR)
                 (DSUBST OFL (QUOTE OFL)
                 (DSUBST <QUOTE (RETEVAL
                  (STKPOS (QUOTE BUBBLE-PNR)
                                    -1)
                  (QUOTE (APPLY BUBBLE-PNR NIL>
                  (QUOTE (BUBBLE-PNR OFL INDEX BUBBLE?))
                  (COPY (GETP (QUOTE BUBBLE-PNR)
                              (QUOTE ALGORITHMUS>
              (APPLY BUBBLE-PNR NIL))
     (T (PRINTL "***ERROR*** HILFSALGORITHMUS" (QUOTE CHANGE)
                "UNDEFINIERT")
                (BREAK1 NIL T)
                NIL)))
  (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE UNGLEICH-PNRP)
             "UNDEFINIERT")
             (BREAK1 NIL T)
              NIL)))
 (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GROESSER-PNRP)
            "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
(T (PRINTL "***ERROR*** HILFSFUNKTION"
           (QUOTE KLEINERGLEICH-PNRP)
           "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
(T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GLEICH-PNRP)
           "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
(T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
```

```
        (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
        (BREAK1 NIL T)
        NIL>>
)

<PUTPROPS BUBBLE-PNR
        RELATION
                    (%> . GROESSER-PNRP)
                    (%< . KLEINER-PNRP)
                    (%>= . GROESSERGLEICH-PNRP)
                    (%<= . KLEINERGLEICH-PNRP)))
        ALGORITHMUS
        (NLAMBDA NIL
          (COND
            (<AND (UNGLEICH-PNRP (OFL LIES (ADD1 INDEX))
                                 (QUOTE UNDEF))
                  (GROESSER-PNRP (OFL LIES INDEX)
                                 (OFL LIES (ADD1 INDEX>
              (SETQ INDEX (ADD1 INDEX))
              (BUBBLE-PNR OFL INDEX BUBBLE?))
            (<AND (UNGLEICH-PNRP (OFL LIES (ADD1 INDEX))
                                 (QUOTE UNDEF))
                  (KLEINERGLEICH-PNRP (OFL LIES INDEX)
                                      (OFL LIES (ADD1 INDEX>
              (SETQ OFL (CHANGE OFL INDEX (ADD1 INDEX)))
              (SETQ INDEX (ADD1 INDEX))
              (SETQ BUBBLE? T)
              (BUBBLE-PNR OFL INDEX BUBBLE?))
            ((AND (GLEICH-PNRP (OFL LIES (ADD1 INDEX))
                               (QUOTE UNDEF))
                  (GLEICH-PNRP BUBBLE? T))
              (SETQ INDEX 1)
              (SETQ BUBBLE?)
              (BUBBLE-PNR OFL INDEX BUBBLE?))
            ((AND (GLEICH-PNRP (OFL LIES (ADD1 INDEX))
                               (QUOTE UNDEF))
                  (GLEICH-PNRP BUBBLE?))
             OFL)
            (T (FEHLER 10>
```

■

Mit BUBBLE-PNR lassen sich PD1 und PD2 nach dem Kriterium PNR sortieren :

```
(SETQ PD1 (BUBBLE-PNR (QUOTE PD1)))
((DAT13) DAT12 DAT13 DAT11)

(SETQ PD2 (BUBBLE-PNR (QUOTE PD2)))
((DAT24) DAT24 DAT21 DAT23 DAT22)
```

(Daß tatsächlich die richtige Reihenfolge vorliegt, sieht man an einem Vergleich der den DATij zugeordneten Personalnummern PNR.)

Die beiden so erzeugten sortierten Personaldateien PD1 und PD2 lassen sich mit MERGE-SORT verschmelzen. Die MERGE-PNR-Spezifikation ist jetzt die alte MERGE-SORT-Spezifikation mit veränderten Datenstrukturen und anderem Relationssatz:

```
ALGORITHMUS-NAME: MERGE-PNR;

PARAMETERTYPEN: BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                                    OFL-A,
                          OFFENELISTE-FLEXVERBUND-LISPATOM
                                    OFL-B;
                SYSTEM:   OFFENELISTE-FLEXVERBUND-LISPATOM
                          OFL-C := (INIT OFL-C
                          OFFENELISTE-FLEXVERBUND-LISPATOM);

ALGORITHMUS:      MERGE-PNR (OFL-A,OFL-B,OFL-C) =
            {     (OFL-A LIES 1) > (OFL-B LIES 1)
             ---> MERGE-PNR ((OFL-A ENTFERN 1),OFL-B,
                             (OFL-C HINZU 1 (OFL-A LIES 1)));
                  (OFL-A LIES 1) < (OFL-B LIES 1)
             ---> MERGE-PNR (OFL-A,(OFL-B ENTFERN 1),
                             (OFL-C HINZU 1 (OFL-B LIES 1)));
                  (OFL-A LIES 1) = (OFL-B LIES 1)
                      UND  (OFL-A LIES 1) /= UNDEF
                      UND  (OFL-B LIES 1) /= UNDEF
             ---> MERGE-PNR ((OFL-A ENTFERN 1),
                             (OFL-B ENTFERN 1),
                             (OFL-C HINZU 1 (OFL-A LIES 1)));
                  (OFL-A LIES 1) = UNDEF   UND
                  (OFL-B LIES 1) = UNDEF
             ---> OFL-C   }

KOMMENTAR:      !!
```

Die Implementierung:

```
(DEFINEQ
<MERGE-PNR
 <LAMBDA (OFL-A OFL-B OFL-C)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL-A (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
          (GETP OFL-B (QUOTE PASSIVSTRUKTUR)))
      (COND
       ((GETD (QUOTE DUPLIZIERE))
        (COND
         ((GETD (QUOTE UNGLEICH-PNRP))
          (COND
           ((GETD (QUOTE GLEICH-PNRP))
```

```
        (COND
         ((GETD (QUOTE KLEINER-PNRP))
          (COND
           ((GETD (QUOTE GROESSER-PNRP))
              (INIT OFL-C
                    OFFENELISTE-FLEXVERBUND-LISPATOM)
              <RPLACA (QUOTE MERGE-PNR)
                (DSUBST OFL-B (QUOTE OFL-B)
                  (DSUBST OFL-A (QUOTE OFL-A)
                    (DSUBST <QUOTE (RETEVAL
                       (STKPOS (QUOTE MERGE-PNR)
                               -1)
                       (QUOTE (APPLY MERGE-PNR NIL>
                       (QUOTE (MERGE-PNR
                                OFL-A OFL-B OFL-C))
                       (COPY (GETP (QUOTE MERGE-PNR)
                                   (QUOTE ALGORITHMUS>
              (APPLY MERGE-PNR NIL))
           (T (PRINTL "***ERROR*** HILFSFUNKTION"
                      (QUOTE GROESSER-PNRP)
                      "UNDEFINIERT")
                      (BREAK1 NIL T)
                      NIL)))
       (T (PRINTL "***ERROR*** HILFSFUNKTION"
                  (QUOTE KLEINER-PNRP)
                  "UNDEFINIERT")
                  (BREAK1 NIL T)
                   NIL)))
     (T (PRINTL "***ERROR*** HILFSFUNKTION"
                (QUOTE GLEICH-PNRP)
                "UNDEFINIERT")
                (BREAK1 NIL T)
                 NIL)))
   (T (PRINTL "***ERROR*** HILFSFUNKTION"
              (QUOTE UNGLEICH-PNRP)
              "UNDEFINIERT")
              (BREAK1 NIL T)
              NIL)))
(T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE DUPLIZIERE)
           "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
 (PRINTL "***ERROR***" OFL-B "NICHT VOM TYP"
         (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
         (BREAK1 NIL T)
         NIL)))
  (T (PRINTL "***ERROR***" OFL-A "NICHT VOM TYP"
             (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
     (BREAK1 NIL T)
     NIL>>

PROPS MERGE-PNR KOMMENTAR ()
      RELATION
      (PVSREL-PNR ((== . GLEICH-PNRP)
                   (/= . UNGLEICH-PNRP)
                   (%> . GROESSER-PNRP)
```



```
                   (%< . KLEINER-PNRP)
                     (%<= . KLEINERGLEICH-PNRP)))
ALGORITHMUS
(NLAMBDA NIL
  (COND
    ((GROESSER-PNRP (OFL-A LIES 1)
                    (OFL-B LIES 1))
      (DUPLIZIERE OFL-A X3)
      (OFL-A ENTFERN 1)
      (OFL-C HINZU 1 (X3 LIES 1))
      (MERGE-PNR OFL-A OFL-B OFL-C))
    ((KLEINER-PNRP (OFL-A LIES 1)
                   (OFL-B LIES 1))
      (DUPLIZIERE OFL-B X3)
      (OFL-B ENTFERN 1)
      (OFL-C HINZU 1 (X3 LIES 1))
      (MERGE-PNR OFL-A OFL-B OFL-C))
    ((AND (GLEICH-PNRP (OFL-A LIES 1)
                       (OFL-B LIES 1))
          (UNGLEICH-PNRP (OFL-A LIES 1)
                         (QUOTE UNDEF))
          (UNGLEICH-PNRP (OFL-B LIES 1)
                         (QUOTE UNDEF)))
      (DUPLIZIERE OFL-A X3)
      (OFL-A ENTFERN 1)
      (OFL-B ENTFERN 1)
      (OFL-C HINZU 1 (X3 LIES 1))
      (MERGE-PNR OFL-A OFL-B OFL-C))
    ((AND (GLEICH-PNRP (OFL-A LIES 1)
                       (QUOTE UNDEF))
          (GLEICH-PNRP (OFL-B LIES 1)
                       (QUOTE UNDEF)))
      OFL-C)
    (T (FEHLER 10>
```

Das Ergebnis der Verschmelzung der Dateien PD1 und PD2 soll PD1 zugewiesen werden:

```
(SETQ PD1 (MERGE-PNR (QUOTE PD1) (QUOTE PD2)))
(PD1 RESET)
((DAT12) DAT22 DAT11 DAT23 DAT13 DAT21 DAT24 DAT12)
```

(Daß tatsächlich die richtige Reihenfolge vorliegt, sieht man an einem Vergleich der den DATij zugeordneten Personalnummern PNR.)

Analog kann man vorgehen, wenn anstelle des Kriteriums PNR das Kriterium NAME zum Sortieren verwendet wird. Die entsprechenden Modifikationen führen zu BUBBLE-NAM und MERGE-NAM.

Angewandt auf die Ausgangssituation ergibt sich dann:

```
 (SETQ PD1 (BUBBLE-NAM (QUOTE PD1)))
((DAT13) DAT12 DAT13 DAT11)

  (SETQ PD2 (BUBBLE-NAM (QUOTE PD2)))
((DAT24) DAT22 DAT23 DAT21 DAT24)

  (SETQ PD1 (MERGE-NAM (QUOTE PD1) (QUOTE PD2)))
(PD1 RESET)
((DAT12) DAT24 DAT11 DAT21 DAT23 DAT22 DAT13 DAT12)
```

Im folgenden wird als Ordnungskriterium implizit immer PNR benutzt. Analoge Algorithmen für NAME oder andere Kriterien existieren teilweise bzw. sind direkt ableitbar.

Bis jetzt sind bereits in anderem Zusammenhang eingeführte Algorithmen durch leichte Veränderung zum Einsatz gekommen.

Die nächste Spezifikation EINFUEGEN beschreibt einen Algorithmus, mit dem eine neue Personendatei DATij in eine bestehende geordnete Personaldatei PDi nach dem Kriterium PNR eingefügt wird:

```
ALGORITHMUS-NAME: EINFUEGEN;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                             OFL,
                            FLEXVERBUND-LISPATOM FL;
                  SYSTEM:   INTEGER IND := 1;

RELATION:         PVSREL-PNR;

HILFSALGORITHMEN: FEHLER;

ALGORITHMUS:      EINFUEGEN (OFL,FL,IND) =

      {       (OFL LIES IND) < 'FL  UND
                (OFL LIES IND) /= UNDEF
        ----> EINFUEGEN(OFL,FL,(ADD1 IND));
              (OFL LIES IND) > 'FL  ODER
                (OFL LIES IND) = UNDEF
        ----> (OFL HINZU IND FL)
              (OFL LIES IND) = 'FL
        ----> FEHLER(99)          }

KOMMENTAR:   !!
```

Die Implementierung:


```
(DEFINEQ
<EINFUEGEN
 <LAMBDA (OFL FL IND)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((EQ (QUOTE FLEXVERBUND-LISPATOM)
          (GETP FL (QUOTE PASSIVSTRUKTUR)))
      (COND
       ((GETD (QUOTE GLEICH-PNRP))
        (COND
         ((GETD (QUOTE GROESSER-PNRP))
          (COND
           ((GETD (QUOTE UNGLEICH-PNRP))
            (COND
             ((GETD (QUOTE KLEINER-PNRP))
              (COND
               ((GETD (QUOTE FEHLER))
               (SETQ IND 1)
               <RPLACA (QUOTE EINFUEGEN)
                 (DSUBST FL (QUOTE FL)
                  (DSUBST OFL (QUOTE OFL)
                   (DSUBST <QUOTE (RETEVAL
                                   (STKPOS (QUOTE EINFUEGEN)
                                           -1)
                                   (QUOTE (APPLY EINFUEGEN NIL>
                                   (QUOTE (EINFUEGEN OFL FL IND))
                                   (COPY (GETP (QUOTE EINFUEGEN)
                                               (QUOTE ALGORITHMUS>
                    (APPLY EINFUEGEN NIL))
              (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                         (QUOTE FEHLER)
                         "UNDEFINIERT")
                         (BREAK1 NIL T)
                         NIL)))
            (T (PRINTL "***ERROR*** HILFSFUNKTION"
                       (QUOTE KLEINER-PNRP)
                       "UNDEFINIERT")
                       (BREAK1 NIL T)
                       NIL)))
          (T (PRINTL "***ERROR*** HILFSFUNKTION"
                     (QUOTE UNGLEICH-PNRP)
                     "UNDEFINIERT")
                     (BREAK1 NIL T)
                     NIL)))
        (T (PRINTL "***ERROR*** HILFSFUNKTION"
                   (QUOTE GROESSER-PNRP)
                   "UNDEFINIERT")
                   (BREAK1 NIL T)
                   NIL)))
      (T (PRINTL "***ERROR*** HILFSFUNKTION"
                 (QUOTE GLEICH-PNRP)
                 "UNDEFINIERT")
```

```
                (BREAK1 NIL T)
                NIL)))
   (T (PRINTL "***ERROR***" FL "NICHT VOM TYP"
              (QUOTE FLEXVERBUND-LISPATOM))
              (BREAK1 NIL T)
              NIL)))
  (T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
             (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
             (BREAK1 NIL T)
      NIL>>
)
 <PUTPROPS EINFUEGEN KOMMENTAR ()
          RELATION
          (PVSREL-PNR ((== . GLEICH-PNRP)
                       (/= . UNGLEICH-PNRP)
                       (%> . GROESSER-PNRP)
                       (%< . KLEINER-PNRP)
                       (%>= . GROESSERGLEICH-PNRP)
                       (%<= . KLEINERGLEICH-PNRP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
              ((AND (KLEINER-PNRP (OFL LIES IND)
                                  (QUOTE FL))
                    (UNGLEICH-PNRP (OFL LIES IND)
                                   (QUOTE UNDEF)))
                (SETQ IND (ADD1 IND))
                (EINFUEGEN OFL FL IND))
              ((OR (GROESSER-PNRP (OFL LIES IND)
                                  (QUOTE FL))
                   (GLEICH-PNRP (OFL LIES IND)
                                (QUOTE UNDEF)))
                (OFL HINZU IND FL))
              ((GLEICH-PNRP (OFL LIES IND)
                            (QUOTE FL))
                (FEHLER 99))
              (T (FEHLER 10>
```

Sei DAT31 eine Inkarnation des Typs FLEXVERBUND-LISPATOM mit folgenden Einträgen :

```
 (RPAQQ DAT31 (NIL (NAME . NIETERS)
                   (VORNAME . HANS)
                   (STRASSE . BONNER-TALWEG 123)
                   (WOHNORT . BONN)
                   (ALTER . 29)
                   (GROESSE . 183)
                   (GEWERKSCHAFT . OETV)
                   (PNR . 2516)))
```


Dann ergibt sich:

```
 (EINFUEGEN (QUOTE PD1) (QUOTE DAT31))
((DAT12) DAT22 DAT11 DAT31 DAT23 DAT13 DAT21 DAT24 DAT12)
```

d.h., DAT31 ist gemäß dem Kriterium PNR an die richtige Stelle eingefügt worden.

Eine andere Fragestellung ist die nach dem Wert eines Indikators in einer Datei DATij; ABFRAGEN durchsucht die Datei sequentiell und liefert den Wert (soweit er existiert):

```
ALGORITHMUS-NAME: ABFRAGEN;

PARAMETERTYPEN: BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                            OFL,
                          FLEXVERBUND-LISPATOM FL,
                          LISPATOM EIG;
                SYSTEM:   INTEGER IND := 1;

RELATION:          PVSREL-PNR;

HILFSALGORITHMEN: FEHLER;

ALGORITHMUS:       ABFRAGEN (OFL,FL,EIG,IND) =

             {     (OFL LIES IND) < 'FL UND
                      (OFL LIES IND) /= UNDEF
               ---> ABFRAGEN(OFL,FL,EIG,(ADD1 IND));
                   (OFL LIES IND) = 'FL
               ---> (FL LIES EIG);
                   (OFL LIES IND) = UNDEF
               ---> FEHLER(99)      }

KOMMENTAR:   !!
```

Die Implementierung:

```
(DEFINEQ
<ABFRAGEN
 <LAMBDA (OFL FL EIG IND)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((EQ (QUOTE FLEXVERBUND-LISPATOM)
          (GETP FL (QUOTE PASSIVSTRUKTUR)))
```

```
   (COND
    ((OR (TYPEP EIG 30)
         (TYPEP EIG 22)
         (TYPEP EIG 23))
     (COND
      ((GETD (QUOTE GLEICH-PNRP))
       (COND
       ((GETD (QUOTE UNGLEICH-PNRP))
        (COND
        ((GETD (QUOTE KLEINER-PNRP))
         (COND
         ((GETD (QUOTE FEHLER))
           (SETQ IND 1)
           <RPLACA (QUOTE ABFRAGEN)
            (DSUBST FL (QUOTE FL)
             (DSUBST OFL (QUOTE OFL)
              (DSUBST <QUOTE (RETEVAL
                              (STKPOS (QUOTE ABFRAGEN)
                                      -1)
                              (QUOTE (APPLY ABFRAGEN NIL>
                              (QUOTE (ABFRAGEN OFL FL EIG IND))
                              (COPY (GETP (QUOTE ABFRAGEN)
                                          (QUOTE ALGORITHMUS>
          (APPLY ABFRAGEN NIL))
    (T (PRINTL "***ERROR*** HILFSALGORITHMUS" (QUOTE FEHLER)
               "UNDEFINIERT")
               (BREAK1 NIL T)
               NIL)))
   (T (PRINTL "***ERROR*** HILFSFUNKTION"
              (QUOTE KLEINER-PNRP)
              "UNDEFINIERT")
              (BREAK1 NIL T)
              NIL)))
    (T (PRINTL "***ERROR*** HILFSFUNKTION"
               (QUOTE UNGLEICH-PNRP)
               "UNDEFINIERT")
               (BREAK1 NIL T)
               NIL)))
    (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GLEICH-PNRP)
               "UNDEFINIERT")
                (BREAK1 NIL T)
                NIL)))
    (T (PRINTL "***ERROR***" EIG "NICHT VOM TYP LISPATOM")
               (BREAK1 NIL T)
               NIL)))
  (T (PRINTL "***ERROR***" FL "NICHT VOM TYP"
             (QUOTE FLEXVERBUND-LISPATOM))
             (BREAK1 NIL T)
             NIL)))
 (T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
            (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
            (BREAK1 NIL T)
        NIL>>
)
  <PUTPROPS ABFRAGEN KOMMENTAR  ()
            RELATION
```



```
          (PVSREL-PNR ((== . GLEICH-PNRP)
                       (/= . UNGLEICH-PNRP)
                       (%> . GROESSER-PNRP)
                       (%< . KLEINER-PNRP)
                       (%>= . GROESSERGLEICH-PNRP)
                       (%<= . KLEINERGLEICH-PNRP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
              ((AND (KLEINER-PNRP (OFL LIES IND)
                                  (QUOTE FL))
                    (UNGLEICH-PNRP (OFL LIES IND)
                                   (QUOTE UNDEF)))
                (SETQ IND (ADD1 IND))
                (ABFRAGEN OFL FL EIG IND))
              ((GLEICH-PNRP (OFL LIES IND)
                            (QUOTE FL))
                (FL LIES EIG))
              ((GLEICH-PNRP (OFL LIES IND)
                            (QUOTE UNDEF))
                (FEHLER 99))
              (T (FEHLER 10>
```

Mögliche Aufrufe:

```
 (ABFRAGEN (QUOTE PD1) (QUOTE DAT12) (QUOTE NAME))
BARTELS

  (ABFRAGEN (QUOTE PD1) (QUOTE DAT24) (QUOTE ALTER))
24

  (ABFRAGEN (QUOTE PD1) (QUOTE DAT22) (QUOTE GEWERKSCHAFT))
UNDEF
```

Eine weitere Einsatzmöglichkeit für ABFRAGEN ist innerhalb des Algorithmus SAMMELN. Hinter SAMMELN steht der Gedanke, alle diejenigen Personen in einer Liste zu sammeln, die Träger einer bestimmten Eigenschaft sind. Vermerkt wird dabei die Personalnummer, wenn die betrachtete Person die Eigenschaft aufweist. Die entstehende Liste von Personalnummern wird in einer Struktur OFFENELISTE-INTEGER abgelegt und von dem Sortieralgorithmus QUICK-SORT (für OFFENELISTE-INTEGER) sortiert.

```
ALGORITHMUS-NAME: SAMMELN;

PARAMETERTYPEN: BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                          OFL,
                         LISPATOM  EIG;
                SYSTEM:  OFFENELISTE-INTEGER ERG-OL :=
                          (INIT ERG-OL OFFENELISTE-INTEGER),
                         INTEGER IND := 1;

RELATION:  UNDEF-KL-ALPHORDER;

HILFSALGORITHMEN: ABFRAGEN, QUICK-SORT;

ALGORITHMUS:       SAMMELN(OFL,EIG,ERG-OL,IND) =
        {     (OFL LIES IND ) /= UNDEF    UND
              ABFRAGEN(OFL,(OFL LIES IND),EIG) /= UNDEF
         ---> SAMMELN(OFL,EIG,
                      (ERG-OL HINZU 1 ABFRAGEN(OFL,
                                               (OFL LIES IND),
                                               PNR)),
                      (ADD1 IND));
              (OFL LIES IND ) /= UNDEF    UND
              ABFRAGEN(OFL,(OFL LIES IND),EIG) = UNDEF
         ---> SAMMELN (OFL,EIG,ERG-OL,(ADD1 IND));
              (OFL LIES IND) = UNDEF
         ---> QUICK-SORT('ERG-OL)  }

KOMMENTAR:   !!
```

Die Implementierung:

```
(DEFINEQ
<SAMMELN
 <LAMBDA (OFL EIG ERG-OL IND)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((OR (TYPEP EIG 30)
          (TYPEP EIG 22)
          (TYPEP EIG 23))
      (COND
       ((GETD (QUOTE GLEICH-ALPHP))
        (COND
        ((GETD (QUOTE UNGLEICH-ALPHP))
         (COND
         ((GETD (QUOTE QUICK-SORT))
          (COND
          ((GETD (QUOTE ABFRAGEN))
            (INIT ERG-OL OFFENELISTE-INTEGER)
            (SETQ IND 1)
```



```
            <RPLACA (QUOTE SAMMELN)
             (DSUBST OFL (QUOTE OFL)
              (DSUBST <QUOTE (RETEVAL (STKPOS (QUOTE SAMMELN)
                                              -1)
                                      (QUOTE (APPLY SAMMELN NIL>
                                       (QUOTE (SAMMELN OFL EIG ERG-OL IND))
                                       (COPY (GETP (QUOTE SAMMELN)
                                       (QUOTE ALGORITHMUS>
                                 (APPLY SAMMELN NIL))
               (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                          (QUOTE ABFRAGEN)
                          "UNDEFINIERT")
                          (BREAK1 NIL T)
                          NIL)))
              (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                         (QUOTE QUICK-SORT)
                         "UNDEFINIERT")
                         (BREAK1 NIL T)
                         NIL)))
          (T (PRINTL "***ERROR*** HILFSFUNKTION"
                     (QUOTE UNGLEICH-ALPHP)
                     "UNDEFINIERT")
                     (BREAK1 NIL T)
                     NIL)))
        (T (PRINTL "***ERROR*** HILFSFUNKTION"
                   (QUOTE GLEICH-ALPHP)
                   "UNDEFINIERT")
                   (BREAK1 NIL T)
                   NIL)))
      (T (PRINTL "***ERROR***" EIG "NICHT VOM TYP LISPATOM")
                 (BREAK1 NIL T)
                 NIL)))
    (T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
               (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
               (BREAK1 NIL T)
               NIL>>
)
  <PUTPROPS SAMMELN KOMMENTAR   ()
          RELATION
          (UNDEF-KL-ALPHORDER ((== . GLEICH-ALPHP)
                               (/= . UNGLEICH-ALPHP)
                               (%< . KLEINER-ALPHP)
                               (%> . GROESSER-ALPHP)
                               (%<= . KLEINERGLEICH-ALPHP)
                               (%>= . GROESSERGLEICH-ALPHP)))
          ALGORITHMUS
          (NLAMBDA NIL
            (COND
              ((AND (UNGLEICH-ALPHP (OFL LIES IND)
                                    (QUOTE UNDEF))
                    (UNGLEICH-ALPHP (ABFRAGEN OFL (OFL LIES IND)
                                              EIG)
                                    (QUOTE UNDEF)))
               (ERG-OL HINZU 1 (ABFRAGEN OFL (OFL LIES IND)
                                         PNR))
               (SETQ IND (ADD1 IND))
```

```
        (SAMMELN OFL EIG ERG-OL IND))
                          (QUOTE UNDEF))
            (GLEICH-ALPHP (ABFRAGEN OFL (OFL LIES IND)
                                  EIG)
                          (QUOTE UNDEF)))
        (SETQ IND (ADD1 IND))
        (SAMMELN OFL EIG ERG-OL IND))
      ((GLEICH-ALPHP (OFL LIES IND)
                     (QUOTE UNDEF))
        (QUICK-SORT (QUOTE ERG-OL)))
      (T (FEHLER 10>
```

Aufrufe für SAMMELN:

```
 (SAMMELN (QUOTE PD1) (QUOTE GEWERKSCHAFT))
((4818) 2214 2516 4211 4322 4818)

  (SAMMELN (QUOTE PD1) (QUOTE NAME))
((6312) 1180 2214 2516 2681 4211 4322 4818 6312)
```

Der als Hilfalgorithmus verwendete Algorithmus ABFRAGEN ist aber ineffezient, da er jedes Element in PDi sequentiell sucht.

Im nächsten Algorithmus wird ABFRAGEN durch GIB-WERT ersetzt. GIB-WERT erzeugt lediglich einen Zugriff zu einem DATij:

```
ALGORITHMUS-NAME: GIB-WERT;

PARAMETERTYPEN:   BENUTZER: FLEXVERBUND-LISPATOM FL,
                            LITATOM EIG;

HILFSALGORITHMEN: PRINTL;
RELATION: ALPHORDER ;


ALGORITHMUS:      GIB-WERT(FL,EIG)  =

      {     (FL LIES EIG)  /= UNDEF
        ---> (FL LIES EIG);
            (FL LIES EIG) = UNDEF
        ---> PRINTL ('UNDEF)        }


KOMMENTAR:  !!
```

Die Verwendung eines derartig spezifizierten Algorithmus hat den Vorteil, daß innerhalb anderer Algorithmen der Zugriff zu verschiedenen DATij nicht über einen formalen Parameter ausgedrückt werden muß, sondern mit Hilfe von GIB-WERT in einen Hilfsalgorithmusaufruf transformiert werden kann (siehe unten).

Andere spezielle Algorithmen dieser Art sind GIB-PNR und GIB-NAME, die jeweils den Wert des Indikatoren PNR oder NAME eines Flexverbundes berechnen. Entsprechende Spezifikationen und Implementierungen befinden sich in den Dateien ALGORITHMEN.SPEC bzw. LISP.DATA.ALGORITHMEN.IMPLEM.nn.

In SAMMELN-2 wird jetzt nicht nur nach dem Vorhandensein einer Eigenschaft gesucht, sondern der zugehörige Wert muß größergleich einem vorgegebenen sein.
Ist das der Fall, wird die entsprechende Personalnummer in eine Liste eingetragen. Diese Liste wird abschließend mit QUICK-SORT sortiert.

```
ALGORITHMUS-NAME: SAMMELN-2;

PARAMETERTYPEN: BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                          OFL,
                          LISPATOM EIG,
                          LISPATOM  WERT;
                SYSTEM:   OFFENELISTE-INTEGER ERG-OL :=
                          (INIT ERG-OL OFFENELISTE-INTEGER),
                          INTEGER IND := 1;

RELATION:  UNDEF-KL-INTEGER;

HILFSALGORITHMEN: GIB-WERT, QUICK-SORT;

ALGORITHMUS:      SAMMELN-2(OFL,EIG,WERT,ERG-OL,IND) =
        {     (OFL LIES IND ) /= UNDEF    UND
              GIB-WERT((OFL LIES IND),EIG) >= WERT
        ---> SAMMELN-2(OFL,EIG,WERT,
                    (ERG-OL HINZU 1 GIB-WERT((OFL LIES IND),
                                             'PNR)),
                    (ADD1 IND));
              (OFL LIES IND ) /= UNDEF    UND
              GIB-WERT((OFL LIES IND),EIG) < WERT
        ---> SAMMELN-2 (OFL,EIG,WERT,ERG-OL,(ADD1 IND));
              (OFL LIES IND) = UNDEF
        ---> QUICK-SORT('ERG-OL)  }

KOMMENTAR:  !!
```

Die Implementierung:

```
<SAMMELN-2
 <LAMBDA (OFL EIG WERT ERG-OL IND)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((OR (TYPEP EIG 30)
          (TYPEP EIG 22)
          (TYPEP EIG 23))
      (COND
       ((OR (TYPEP WERT 30)
            (TYPEP WERT 22)
            (TYPEP WERT 23))
        (COND
        ((GETD (QUOTE GLEICHP))
          (COND
           ((GETD (QUOTE KLEINERP))
            (COND
            ((GETD (QUOTE GROESSERGLEICHP))
             (COND
             ((GETD (QUOTE UNGLEICHP))
              (COND
              ((GETD (QUOTE QUICK-SORT))
               (COND
                ((GETD (QUOTE GIB-WERT))
                 (INIT ERG-OL OFFENELISTE-INTEGER)
                 (SETQ IND 1)
                 <RPLACA (QUOTE SAMMELN-2)
                  (DSUBST OFL (QUOTE OFL)
                   (DSUBST <QUOTE (RETEVAL
                                   (STKPOS (QUOTE SAMMELN-2)
                                           -1)
                                   (QUOTE (APPLY SAMMELN-2 NIL>
                                   (QUOTE (SAMMELN-2 OFL
                                           EIG WERT ERG-OL IND))
                                   (COPY (GETP (QUOTE SAMMELN-2)
                                               (QUOTE ALGORITHMUS>
                   (APPLY SAMMELN-2 NIL))
                  (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                             (QUOTE GIB-WERT)
                             "UNDEFINIERT")
                             (BREAK1 NIL T)
                             NIL)))
                 (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                            (QUOTE QUICK-SORT)
                            "UNDEFINIERT")
                            (BREAK1 NIL T)
                            NIL)))
                (T (PRINTL "***ERROR*** HILFSFUNKTION"
                           (QUOTE UNGLEICHP)
                           "UNDEFINIERT")
                           (BREAK1 NIL T)
                           NIL)))
               (T (PRINTL "***ERROR*** HILFSFUNKTION"
                          (QUOTE GROESSERGLEICHP)
```



```
                          "UNDEFINIERT")
                          (BREAK1 NIL T)
                          NIL)))
              (T (PRINTL "***ERROR*** HILFSFUNKTION"
                         (QUOTE KLEINERP)
                         "UNDEFINIERT")
                         (BREAK1 NIL T)
                         NIL)))
             (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GLEICHP)
                        "UNDEFINIERT")
                        (BREAK1 NIL T)
                        NIL)))
            (T (PRINTL "***ERROR***" WERT "NICHT VOM TYP LISPATOM")
               (BREAK1 NIL T)
               NIL)))
           (T (PRINTL "***ERROR***" EIG "NICHT VOM TYP LISPATOM")
              (BREAK1 NIL T)
              NIL)))
         (T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
                    (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
                    (BREAK1 NIL T)
                    NIL>>
)
 <PUTPROPS SAMMELN-2 KOMMENTAR ()
           RELATION
           (UNDEF-KL-INTEGER ((== . GLEICHP)
                              (/= . UNGLEICHP)
                              (%< . KLEINERP)
                              (%> . GROESSERP)
                              (%<= . KLEINERGLEICHP)
                              (%>= . GROESSERGLEICHP)))
           ALGORITHMUS
           (NLAMBDA NIL
             (COND
               ((AND (UNGLEICHP (OFL LIES IND)
                                (QUOTE UNDEF))
                     (GROESSERGLEICHP (GIB-WERT (OFL LIES IND)
                                                EIG)
                                      WERT))
                (ERG-OL HINZU 1 (GIB-WERT (OFL LIES IND)
                                          PNR))
                (SETQ IND (ADD1 IND))
                (SAMMELN-2 OFL EIG WERT ERG-OL IND))
               ((AND (UNGLEICHP (OFL LIES IND)
                                (QUOTE UNDEF))
                     (KLEINERP (GIB-WERT (OFL LIES IND)
                                         EIG)
                               WERT))
                (SETQ IND (ADD1 IND))
                (SAMMELN-2 OFL EIG WERT ERG-OL IND))
               ((GLEICHP (OFL LIES IND)
                         (QUOTE UNDEF))
                (QUICK-SORT (QUOTE ERG-OL)))
               (T (FEHLER 10>
```

SAMMELN-2 ergibt:

```
  (SAMMELN-2 (QUOTE PD1) (QUOTE ALTER) 25)
((6312) 1180 2214 2516 4211 4322 6312)
```

[Personalnummern der Personen, die älter als 25 sind]

```
(SAMMELN-2 (QUOTE PD1) (QUOTE PNR) 5000)
((6312) 6312)
```

[Personalnummern der Personen, deren PNR größer als 5000 ist]

Sei jetzt OI1 eine Inkarnation des Typs OFFENELISTE-INTEGER. Dann kann der Wert eines SAMMELN-2-Aufrufs folgendermaßen zugewiesen werden:

```
(SETQ OI1 (SAMMELN-2 (QUOTE PD1) (QUOTE GROESSE) (QUOTE 185)))
(OI1 RESET)
((6312) 2214 4211 4818 6312)
```

Hat man sich eine derartige Liste von PNR's erzeugt, so sollen jetzt als letzte fiktive Aufgabe des PVS' die Adressen der so bezeichneten Personen erzeugt werden. Dazu muß zunächst für jede PNR die richtige Datei DATij gesucht werden. Für dieses Problem wird eine Version des FIBO-SEARCH eingesetzt. FIBO-PNR (FIBO-NAM ist auch implementiert) sucht eine vorgegebene Personalnummer in einer Personaldatei gemäß dem Fibonacci-Suchverfahren. Dabei wird wiederum GIB-WERT benutzt.
Falls die Suche erfolgreich ist, wird der Name der zugehörigen Datei (DATij) an das Atom INKNAME gehängt, das weiter unten im Algorithmus ADRESSENLISTE referiert wird.

```
ALGORITHMUS-NAME: FIBO-PNR;

PARAMETERTYPEN:
        BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM OFL,
                  INTEGER  PNR;
        SYSTEM:   INTEGER I := (FIBO-ZAHL 1(LISTSIZE OFL)),
                  INTEGER P := (FIBO-ZAHL 2(LISTSIZE OFL)),
                  INTEGER Q := (FIBO-ZAHL 3(LISTSIZE OFL)),
                  INTEGER M := (FIBO-ZAHL 4(LISTSIZE OFL)),
                  BOOL FLAG := FALSE;

RELATION: UNDEF-KL-INTEGER;


HILFSALGORITHMEN: FIBO-ZAHL, LISTSIZE,SETQ,GIB-WERT,FEHLER;

ALGORITHMUS:    FIBO-PNR (OFL,PNR,I,P,Q,M,FLAG) =
```



```
          {   Q /= 0 UND
              GIB-WERT((OFL LIES I), 'PNR) > PNR
              ---> FIBO-PNR(OFL,PNR,(IDIFFERENCE I Q),Q,
                            (IDIFFERENCE P Q),M, TRUE);
              P /= 1 UND
              GIB-WERT((OFL LIES I), 'PNR) < PNR UND
              FLAG = TRUE
              ---> FIBO-PNR(OFL,PNR,(IPLUS I Q),
                            (IDIFFERENCE P Q),
                            (ITIMES 2(IDIFFERENCE Q P)),
                            M,TRUE);
              P /= 1 UND
              GIB-WERT ((OFL LIES I), 'PNR) < PNR UND
              FLAG = FALSE
              ---> FIBO-PNR(OFL,PNR,
                            (IDIFFERENCE I (IDIFFERENCE M Q))
                            (IDIFFERENCE P Q),
                            (IDIFFERENCE (ITIMES 2 Q) P),
                            M,TRUE);
              GIB-WERT( (OFL LIES I), 'PNR) = PNR
              ---> (SETQ INKNAME (OFL LIES I));
              Q = 0  ODER  P = 1
              ---> FEHLER(99)    }


KOMMENTAR:      !!
```

Die Implementierung :

```
(DEFINEQ
<FIBO-PNR
 <LAMBDA (OFL PNR I P Q M FLAG)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
     (COND
      ((OR (TYPEP PNR 22)
           (TYPEP PNR 23))
       (COND
        ((GETD (QUOTE DUPLIZIERE))
         (COND
          ((GETD (QUOTE GLEICHP))
           (COND
           ((GETD (QUOTE KLEINERP))
            (COND
            ((GETD (QUOTE GROESSERP))
             (COND
             ((GETD (QUOTE UNGLEICHP))
              (COND
               ((GETD (QUOTE FEHLER))
                (COND
                 ((GETD (QUOTE GIB-WERT))
                  (COND
                  ((GETD (QUOTE SETQ))
```

```
          (COND
          ((GETD (QUOTE LISTSIZE))
           (COND
           ((GETD (QUOTE FIBO-ZAHL))
            (SETQ I (FIBO-ZAHL 1 (LISTSIZE OFL)))
            (SETQ P (FIBO-ZAHL 2 (LISTSIZE OFL)))
            (SETQ Q (FIBO-ZAHL 3 (LISTSIZE OFL)))
            (SETQ M (FIBO-ZAHL 4 (LISTSIZE OFL)))
            (SETQ FLAG NIL)
            <RPLACA (QUOTE FIBO-PNR)
            (DSUBST OFL (QUOTE OFL)
              (DSUBST <QUOTE (RETEVAL
                       (STKPOS (QUOTE FIBO-PNR)
                               -1)
                       (QUOTE (APPLY FIBO-PNR NIL>
                       (QUOTE (FIBO-PNR OFL
                                PNR I P Q M FLAG))
                       (COPY (GETP (QUOTE FIBO-PNR)
                       (QUOTE ALGORITHMUS>
           (APPLY FIBO-PNR NIL))
          (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                     (QUOTE FIBO-ZAHL)
                     "UNDEFINIERT")
                     (BREAK1 NIL T)
                     NIL)))
        (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                   (QUOTE LISTSIZE)
                   "UNDEFINIERT")
                   (BREAK1 NIL T)
                   NIL)))
      (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                 (QUOTE SETQ)
                 "UNDEFINIERT")
                 (BREAK1 NIL T)
                 NIL)))
    (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
               (QUOTE GIB-WERT)
               "UNDEFINIERT")
               (BREAK1 NIL T)
                NIL)))
   (T (PRINTL "***ERROR*** HILFSALGORITHMUS" (QUOTE FEHLER)
              "UNDEFINIERT")
              (BREAK1 NIL T)
              NIL)))
  (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE UNGLEICHP)
             "UNDEFINIERT")
             (BREAK1 NIL T)
             NIL)))
 (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GROESSERP)
            "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
(T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE KLEINERP)
           "UNDEFINIERT")
           (BREAK1 NIL T)
           NIL)))
```



```
  (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE GLEICHP)
             "UNDEFINIERT")
             (BREAK1 NIL T)
             NIL)))
 (T (PRINTL "***ERROR*** HILFSFUNKTION" (QUOTE DUPLIZIERE)
            "UNDEFINIERT")
            (BREAK1 NIL T)
            NIL)))
(T (PRINTL "***ERROR***" PNR "NICHT VOM TYP INTEGER")
           (BREAK1 NIL T)
           NIL)))
(T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
           (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
          (BREAK1 NIL T)
          NIL>>
)
  <PUTPROPS FIBO-PNR KOMMENTAR ()
        RELATION
            (UNDEF-KL-INTEGER ((== . GLEICHP)
                               (/= . UNGLEICHP)
                               (%< . KLEINERP)
                               (%> . GROESSERP)
                               (%<= . KLEINERGLEICHP)
                               (%>= . GROESSERGLEICHP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                ((AND (UNGLEICHP Q 0)
                      (GROESSERP (GIB-WERT (OFL LIES I)
                                           (QUOTE PNR))
                                 PNR))
                  (DUPLIZIERE P (STD1-5))
                  (SETQ I (IDIFFERENCE I Q))
                  (SETQ P Q)
                  (SETQ Q (IDIFFERENCE STD1-5 Q))
                  (SETQ FLAG T)
                  (FIBO-PNR OFL PNR I P Q M FLAG))
                ((AND (UNGLEICHP P 1)
                      (KLEINERP (GIB-WERT (OFL LIES I)
                                          (QUOTE PNR))
                                PNR)
                      (GLEICHP FLAG T))
                  (DUPLIZIERE P (STD3-5))
                  (SETQ I (IPLUS I Q))
                  (SETQ P (IDIFFERENCE P Q))
                  (SETQ Q (ITIMES 2 (IDIFFERENCE Q STD3-5)))
                  (SETQ FLAG T)
                  (FIBO-PNR OFL PNR I P Q M FLAG))
                ((AND (UNGLEICHP P 1)
                      (KLEINERP (GIB-WERT (OFL LIES I)
                                          (QUOTE PNR))
                                PNR)
                      (GLEICHP FLAG))
                  (DUPLIZIERE P (STD4-5))
                  (SETQ I (IDIFFERENCE I (IDIFFERENCE M Q)))
                  (SETQ P (IDIFFERENCE P Q))
```

```
            (SETQ Q (IDIFFERENCE (ITIMES 2 Q)
                                 STD4-5))
            (SETQ FLAG T)
            (FIBO-PNR OFL PNR I P Q M FLAG))
          ((GLEICHP (GIB-WERT (OFL LIES I)
                              (QUOTE PNR))
                    PNR)
            (SETQ INKNAME (OFL LIES I)))
          ((OR (GLEICHP Q 0)
               (GLEICHP P 1))
           (FEHLER 99))
          (T (FEHLER 10>
```

Mit FIBO-PNR wird eine gesuchte Datei gefunden oder eine Fehlermeldung erzeugt. Im erfolgreichen Fall extrahiert GIB-ADRESSE daraus die Adresse und speichert sie in einer OFFENENLISTE-Struktur

```
ALGORITHMUS-NAME: GIB-ADRESSE;

PARAMETERTYPEN:   BENUTZER: FLEXVERBUND-LISPATOM FL;
                  SYSTEM:   OFFENELISTE ADR-OL :=
                            (INIT ADR-OL OFFENELISTE),
                            INTEGER CNT := 1;


ALGORITHMUS:      GIB-ADRESSE(FL,ADR-OL,CNT) =

 {      CNT = 1
 ---> GIB-ADRESSE(FL,(ADR-OL HINZU 1 (FL LIES 'NAME)),2);
        CNT = 2
 ---> GIB-ADRESSE(FL,(ADR-OL HINZU 1 (FL LIES 'VORNAME)),3);
        CNT = 3
 ---> GIB-ADRESSE(FL,(ADR-OL HINZU 1 (FL LIES 'STRASSE)),4);
        CNT = 4
 ---> (ADR-OL HINZU 1 (FL LIES 'WOHNORT))       }

KOMMENTAR       !!
```

Die Implementierung:

```
(DEFINEQ
<GIB-ADRESSE
 <LAMBDA (FL ADR-OI CNT)
```



```
  (COND
    ((EQ (QUOTE FLEXVERBUND-LISPATOM)
         (GETP FL (QUOTE PASSIVSTRUKTUR)))
    (INIT ADR-OI OFFENELISTE-INTEGER)
    (SETQ CNT 1)
    <RPLACA (QUOTE GIB-ADRESSE)
      (DSUBST FL (QUOTE FL)
               (DSUBST <QUOTE (RETEVAL
                               (STKPOS (QUOTE GIB-ADRESSE)
                                       -1)
                               (QUOTE (APPLY GIB-ADRESSE NIL>
                               (QUOTE (GIB-ADRESSE FL ADR-OI CNT))
                               (COPY (GETP (QUOTE GIB-ADRESSE)
                                           (QUOTE ALGORITHMUS>
        (APPLY GIB-ADRESSE NIL))
    (T (PRINTL "***ERROR***" FL "NICHT VOM TYP"
               (QUOTE FLEXVERBUND-LISPATOM))
      (BREAK1 NIL T)
      NIL>>
)
  <PUTPROPS GIB-ADRESSE KOMMENTAR ()
            RELATION
            (STANDARD ((== . EQ)
                       (/= . NEQ)
                       (%< . LESSP)
                       (%> . GREATERP)
                       (%<= . LESSEQP)
                       (%>= . GREATEREQP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                ((EQ CNT 1)
                  (ADR-OI HINZU 1 (FL LIES (QUOTE NAME)))
                  (SETQ CNT 2)
                  (GIB-ADRESSE FL ADR-OI CNT))
                ((EQ CNT 2)
                  (ADR-OI HINZU 1 (FL LIES (QUOTE VORNAME)))
                  (SETQ CNT 3)
                  (GIB-ADRESSE FL ADR-OI CNT))
                ((EQ CNT 3)
                  (ADR-OI HINZU 1 (FL LIES (QUOTE STRASSE)))
                  (SETQ CNT 4)
                  (GIB-ADRESSE FL ADR-OI CNT))
                <(EQ CNT 4)
                  (ADR-OI HINZU 1 (FL LIES (QUOTE WOHNORT>
                (T (FEHLER 10>
```

Ein Aufruf von GIB-ADRESSE führt zu folgendem Ergebnis:

```
  (GIB-ADRESSE (QUOTE DAT11))
((OLTHOFF)   BONN-OBERKASSEL  KOENIGSWINTERER STR 809   WALTER
```

OLTHOFF)

Mit Hilfe von FIBO-PNR und GIB-ADRESSE läßt sich jetzt eine mit SAMMELN-2 erzeugte Liste von Personalnummern in eine Adressenliste umwandeln. Der Algorithmus dafür lautet:

```
ALGORITHMUS-NAME: ADRESSENLISTE;

PARAMETERTYPEN:   BENUTZER: OFFENELISTE-FLEXVERBUND-LISPATOM
                              OFL,
                            OFFENELISTE-INTEGER OI;
                  SYSTEM:   OFFENELISTE ERG-OL :=
                            (INIT ERG-OL OFFENELISTE),
                            INTEGER IND1 := 1,
                            LITATOM INKNAME := 'NOBIND;

RELATION:         ALPHORDER;

HILFSALGORITHMEN: GIB-ADRESSE,FIBO-PNR;

ALGORITHMUS:   ADRESSENLISTE(OFL,OI,ERG-OL,IND1,INKNAME) =

   {      (OI LIES IND1) /= UNDEF UND
          FIBO-PNR('OFL,(OI LIES IND1)) /= UNDEF
        ---> ADRESSENLISTE(OFL,OI,(ERG-OL HINZU 1
                                   GIB-ADRESSE(INKNAME)),
                           (ADD1 IND1), INKNAME);
          (OI LIES IND1) /= UNDEF UND
          FIBO-PNR('OFL,(OI LIES IND)) = UNDEF
        ---> ADRESSENLISTE(OFL,OI,
                           (ERG-OL HINZU 1 (OI LIES IND1)),
                           (ADD1 IND1), INKNAME);
          (OI LIES IND1) = UNDEF
        --->  ERG-OL                 }

KOMMENTAR:   !!
```

Die Implementierung:

```
(DEFINEQ
<ADRESSENLISTE
 <LAMBDA (OFL OI ERG-OL IND1 INKNAME)
  (COND
   ((EQ (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM)
        (GETP OFL (QUOTE PASSIVSTRUKTUR)))
    (COND
     ((EQ (QUOTE OFFENELISTE-INTEGER)
          (GETP OI (QUOTE PASSIVSTRUKTUR)))
      (COND
       ((GETD (QUOTE FIBO-PNR))
```



```
          (COND
           ((GETD (QUOTE GIB-ADRESSE))
             (INIT ERG-OL OFFENELISTE)
             (SETQ IND1 1)
             (SETQ INKNAME (QUOTE NOBIND))
             <RPLACA (QUOTE ADRESSENLISTE)
                (DSUBST OI (QUOTE OI)
                  (DSUBST OFL (QUOTE OFL)
                    (DSUBST <QUOTE (RETEVAL
                                    (STKPOS (QUOTE ADRESSENLISTE)
                                            -1)
                                    (QUOTE (APPLY ADRESSENLISTE NIL>
                                    (QUOTE (ADRESSENLISTE OFL
                                                OI ERG-OL IND1 INKNAME))
                            (COPY (GETP (QUOTE ADRESSENLISTE)
                                  (QUOTE ALGORITHMUS>
            (APPLY ADRESSENLISTE NIL))
           (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                      (QUOTE GIB-ADRESSE)
                      "UNDEFINIERT")
                      (BREAK1 NIL T)
                      NIL)))
           (T (PRINTL "***ERROR*** HILFSALGORITHMUS"
                      (QUOTE FIBO-PNR)
                      "UNDEFINIERT")
                      (BREAK1 NIL T)
                      NIL)))
          (T (PRINTL "***ERROR***" OI "NICHT VOM TYP"
                     (QUOTE OFFENELISTE-INTEGER))
                     (BREAK1 NIL T)
                     NIL)))
      (T (PRINTL "***ERROR***" OFL "NICHT VOM TYP"
                 (QUOTE OFFENELISTE-FLEXVERBUND-LISPATOM))
                 (BREAK1 NIL T)
                  NIL>>
)
  <PUTPROPS ADRESSENLISTE KOMMENTAR ()
            RELATION
            (ALPHORDER ((== . EQ)
                        (/= . NEQ)
                        (%< . LESS-ALPHP)
                        (%> . GREATER-ALPHP)
                        (%<= . LESSEQ-ALPHP)
                        (%>= . GREATEREQ-ALPHP)))
            ALGORITHMUS
            (NLAMBDA NIL
              (COND
                ((AND (NEQ (OI LIES IND1)
                           (QUOTE UNDEF))
                      (NEQ (FIBO-PNR (QUOTE OFL)
                                     (OI LIES IND1))
                           (QUOTE UNDEF)))
                  (ERG-OL HINZU 1 (GIB-ADRESSE INKNAME))
                  (SETQ IND1 (ADD1 IND1))
                  (ADRESSENLISTE OFL OI ERG-OL IND1 INKNAME))
                ((AND (NEQ (OI LIES IND1)
```

```
                    (QUOTE UNDEF))
                (EQ (FIBO-PNR (QUOTE OFL)
                              (OI LIES IND))
                    (QUOTE UNDEF)))
            (ERG-OL HINZU 1 (OI LIES IND1))
            (SETQ IND1 (ADD1 IND1))
            (ADRESSENLISTE OFL OI ERG-OL IND1 INKNAME))
           ((EQ (OI LIES IND1)
                (QUOTE UNDEF))
            ERG-OL)
           (T (FEHLER 10>
                                                    ■
```

Mit dem oben an OI1 zugewiesenem Wert (Personalnummern der Personen mit Größe über 185) ergibt ein Aufruf von ADRESSENLISTE folgende Daten:

```
   (ADRESSENLISTE (QUOTE PD1) (QUOTE OI1)))
((((OLTHOFF)
   BONN-OBERKASSEL KOENIGSWINTERER STR 809 WALTER OLTHOFF))
 ((BARTELS)
  BONN-HOHOLZ GIELGEN STR 27 ULRICH BARTELS)
 ((WILBERTS)
  BONN POPELSDORFER ALLEE 54 BARBARA WILBERTS)
 ((BURCK)
  BONN OBERKASSEL KOENIGSWINTERER STR 809 VOLKER BURCK)
 ((OLTHOFF)
  BONN-OBERKASSEL KOENIGSWINTERER STR 809 WALTER OLTHOFF))
```

Mit Hilfe des APE-Systems lassen sich also auf relativ einfache Weise über den Weg abstrakter Spezifikationen Programmsysteme für die unterschiedlichsten Anwendungen erzeugen. Da im obigen Beispiel auch nur Datentypen Verwendung fanden, die zum gegenwärtigen Repertoire des Systems gehören, sind Probleme auf dieser Grundlage losbare Probleme. Im Falle neuer Datenstrukturen kann entweder versucht werden, diese auf die bestehenden zurückzuführen, oder, was im Hinblick auf die Leistungsfähigkeit des APE-Systems vorzuziehen wäre, den entsprechenden Typ mit einer geeigneten Spezifikation zu versehen und durch das ADTCOMP-Subsystem zu implementieren.

5. Systembeschreibung APE

Im folgenden werden die Funktionen und Regeln des APE im einzelnen beschrieben und zwar

in 5.1. Funktionen und Regeln des ADTCOMP-Subsystems und

in 5.2. Funktionen und Regeln des ALGCOMP-Subsystems.

Es folgen

in 5.3. Beschreibungen der Funktionen, die zum Arbeiten mit Datentyp- und Algorithmenimplementierungen notwendig sind.

Schließlich sind

in 5.4. noch einige Funktionen zusammengefaßt, die in allen Systemteilen Verwendung finden, und

in 5.5. einige kurze Anmerkungen zur Fehlerbehandlung im APE-System.

Bei den Funktionsbeschreibungen werden i.d.R. Angaben zum Zweck der Funktion, ihrem Aufruf, ihren Parametern, Ergebnis und Wert gemacht. Diese Informationen werden gegebenenfalls durch Erläuterungen zu besonderen Eigenschaften und Voraussetzungen sowie durch Bemerkungen ergänzt.
Bei den umfangreichen und unübersichtlicheren Funktionen dient ein Blockdiagramm dem besseren Überblick. Bei jeder Funktion schließt sich an die Beschreibung der LISP-Code, beendet durch ein ■-Zeichen, an.

Die Regeln des ADTCOMP-Substystems sind hier nicht kommentiert, da sie größtenteils aus [EKT 79] übernommen wurden. Die Änderungen, die durch die Erweiterungen zur Implementierung von parametrisierten Datentypen durchgeführt worden sind, wurden bereits in 3.1.1. dokumentiert.
Die ALGCOMP-Regeln gliedern sich in je eine Regelbasis für die sechs Phasen der Algorithmus-Konstruktions-Phase (S1-Regeln, ... ,S6-Regeln). Dieser Teil ist ausführlich beschrieben.

5.1. Systembeschreibung ADTCOMP

Die Regeln und Funktionen des ADTCOMP-Subsystems haben wir zum größten Teil aus [EKT 79] übernommen. Da diese frühe ADTCOMP-Version jedoch in LISP-F3 geschrieben war und auf einer AEG 80-60 arbeitete, mußte das System zunächst an INTERLISP und die SIEMENS 7.760 durch Modifikation zahlreicher Funktionen und Regeln angepaßt werden. Eine genaue Beschreibung der Änderungen findet sich in [BO 80].
Ferner ist ADTCOMP um einige Regeln und Funktionen erweitert worden, um das in Kapitel 3. erwähnte Parametrisierungskonzept für abstrakte Datentypen zu verwirklichen.

5.1.1. Funktionen des ADTCOMP-Systems

Die Funktionsbeschreibungen sind wie folgt angeordnet:

In 5.1.1.1. werden die übergeordneten Funktionen besprochen. Hauptfunktionen (vom Benutzer zur Implementierung eines ADT aufzurufen), sind PARAMETRISIERE, für parametrisierte Datentypen, und IMPLEMENTIERE für nicht-parametrisierte Datentypen. Diese Funktionen werden realisiert durch die Funktion ADT-IMPLEM, die wiederum realisiert wird durch die Funktionen für die verschiedenen zu durchlaufenden Phasen: RADT (für ;-Phase), ->-PHASE, =-PHASE, &-PHASE, I-PHASE, B-PHASE und L-PHASE.
In 5.1.1.2. werden die untergeordneten Funktionen besprochen. Und zwar

in 5.1.1.2.1. die Hilfsfunktionen von RADT,

in 5.1.1.2.2. die " von ->- und =-PHASE,

in 5.1.1.2.3. die " von &-, I- und B-Phase,

in 5.1.1.2.4. die " von L-PHASE,

in 5.1.1.2.5. die allgemeinen Hilfsfunktionen.

Abschnitt 5.1.1.3. beschreibt noch einige Testfunktionen.

5.1.1.1. Die übergeordneten Funktionen

<PARAMETRISIERE <LAMBDA (DATENTYP PARAMETER) ... >>

Zweck: Erzeugen einer parametrisierten, formalen Struktur mit Namen DATENTYP-PARAMETER.
Aufruf: Benutzer
Param.: DATENTYP gibt den Haupttyp des zu implementierenden Datentyps an und muß gleichzeitig mit dem Dateinamen der Datei übereinstimmen, in der die entsprechende

Spezifikation steht, da dieser Parameter an die Funktion IMPLEMENTIERE weitergegeben wird;
PARAMETER gibt den (die) Parameter an, mit dem (denen) der Haupttyp DATENTYP parametrisiert werden soll, also zum Beispiel erzeugt der Funktionsaufruf

(PARAMETRISIERE KELLER INTEGER)

eine parametrisierte, formale Struktur eines Integer-Kellers mit Haupttyp KELLER und Parameter INTEGER.
Ergebnis: Der angegebene paramatrisierte abstrakte Datentyp wird, falls in den Regeln genügend Wissen vorhanden ist, implementiert.
Wert: NIL
Bem.: PARAMETRISIERE ruft im wesentlichen die Funktion IMPLEMENTIERE auf. Außerdem wird an die property-list des Atoms D unter dem Indikator PARAMETER der aktuelle Parameter abgelegt.

```
<PARAMETRISIERE
  <LAMBDA (DATENTYP PARAMETER)
    (PUT (QUOTE D)
         (QUOTE PARAMETER)
         PARAMETER)
    (IMPLEMENTIERE DATENTYP>>                    ■
```

<IMPLEMENTIERE <LAMBDA (DATEI) ... >>

Aufruf: Vom Benutzer.
Param.: DATEI ist die Datei mit der ADT-Spezifikation.
Wenn DATEI = NIL, wird die ADT-Eingabe über den Bildschirm erwartet.
Ergebnis: Der angebebene ADT wird implementiert, sofern in den Regeln genügend Wissen kodiert ist.
Bem.: Die Funktion ADT-IMPLEM wird für alle Phasen durchgeführt.

```
<IMPLEMENTIERE
  <LAMBDA (DATEI)
    (ADT-IMPLEM (QUOTE (; -%> = & I B L))
                DATEI>>                          ■
```

<ADT-IMPLEM <LAMBDA (PHASEN DATEI ...) ... >>

"ADT-IMPLEMentierung"

Aufruf: Vom Benutzer.
Vorauss.: Wie bei RADT.
Param.: DATEI gibt die Datei an, in der die ADT-Spezifikation steht;
Bei DATEI = NIL wird Eingabe über den Bildschirm erwartet;

PHASEN ist eine Liste mit den zur Ausführung gewünschten Phasen:
";" (siehe RADT) fürs Einlesen,
"-%>" (siehe ->PHASE),
"=" (siehe =-PHASE),
"&" (siehe &-PHASE),
"I" (siehe I-PHASE),
"B" (siehe B-PHASE),
"L" (siehe L-PHASE).

Ergebnis: Aufruf der Programme für die einzelnen Phasen.

```
<ADT-IMPLEM
  <LAMBDA (PHASEN DATEI REGELNAME PRUEFLINGNAME)
    (OR PHASEN (SETQQ PHASEN (; -%> = &)))
    <MAPC PHASEN (FUNCTION (LAMBDA (PHASE)
           (COND
             ((EQ PHASE (QUOTE ;))
               (RADT DATEI))
             (T <APPLY (PACK (LIST PHASE (QUOTE -PHASE>
                (PRINTL PHASE "PHASE BEENDET">
    (TERPRI)
    (TERPRI)
    (PRINTREGLIST VERWENDET 8
                  "VERWENDET WURDEN DIE REGELN:">>          ■
```

<RADT <LAMBDA (EINGABEDATEI) ... >>
"Read ADT" (;-Phase)

Aufruf: Von ADT-IMPLEM.

Vorauss.: Datei mit Namen EINGABEDATEI muß eine ADT-Spezifikation enthalten. Bei EINGABEDATEI = NIL wird Eingabe vom Bildschirm erwartet.

Ergebnis: Einlesen der ADT-Spezifikation, Bildung der globalen Atome:
S, D und X sowie
MENGEN - Liste der im ADT auftretenden Mengen, einschließlich X, ausschließlich D, S.
OPERATOREN - Liste der im ADT auftretenden Operatoren, ausschließlich unter BENUTZUNG definierter.
BENOPS - Liste der unter BENUTZUNG definierten Operatoren.
FUNKS - Assoziationsliste der Funktionalitäten in interner Darstellung. Die Funktionalitäten sind fortlaufend mit F01, F02 bis Fnn bezeichnet. Sie selbst sind Listen, in denen der Operatorname, die Mengen und "->" als einzelne Atome enthalten sind.
AXIOME - Assoziationsliste der Axiome in interner Darstellung. Die Axiome sind fortlaufend mit A01, A02, bis Ann bezeichnet. Sie selbst sind Listen von Atomen.
URBINDUNGEN - Assoziationsliste mit den immer bestehenden Bindungen ((*ADT . "adt-traeger")(D . D)(DO . DO)(X . X)(S . S)).
IMPLEMOPS - Liste der zu implementierenden Operatoren (mit der Eigenschaft OP IMPLEM).
ADT-NAME - Name des ADTs, wie hinter dem Schlüsselwort "DATENTYP:" angegeben.
ADT-TRAEGER - Variable für den Träger des ADTs: D oder S.
ADT-NAMECOMS - Name einer Variable für MAKEFILE, z.B. GRENZKELLERCOMS. Diese Variable enthält zum leichteren Retten Angaben über alle von RADT erzeugten Variablen und deren Eigenschaften.

RADT setzt die Eigenschaften der Aufbauphase, protokolliert die Ergebnisse und druckt ferner die Mengen, Operatoren, Funktionalitäten und Axiome in Interndarstellung aus.
Alle in der Aufbauphase festzulegenden Eigenschaften werden gesetzt, so NAME, GEDSTRUKTUR, KOMPONENTENMENGE, MENGE, BENUTZUNGSDEF.
Ferner werden die unter BENUTZUNG simulierten Operatoren als LAMBDA-Ausdrücke unter ihrer Eigenschaft BENUTZUNGSDEF abgespeichert.

Bem.: Der ADT wird atomweise eingelesen. Ein ";" bewirkt Interpretation des folgenden Atoms als Schlüsselwort mittels SELECTQ. ";;" beendet das Einlesen. Die Zeichen ":" und "+" sind als Zwischenräume vereinbart und können daher auch weggelassen werden. (Veränderungen gegenüber der Dokumentation in [EKT 79] sind in [BO 80] beschrieben.)

```
<RADT
 <LAMBDA (EINGABEDATEI)
  (PROGN (SETSEPR (QUOTE (78 122)) 1)
         (SETBRK (QUOTE (94 107 126)) 1)
         (*COM* "ZEICHENDEUTUNG: : UND + ALS SEPERATORS,
                                 = - > , UND ; ALS BREAKS")
  (PRINTL "ADT-EINGABE UEBER DATEI:" (COND
            (EINGABEDATEI EINGABEDATEI)
            (T "BILDSCHIRM")))
  (PROG (ADT-SCHLUESSEL SPRUNG
                        (ALTEINGABE (INFILE EINGABEDATEI)))
        (PROGN (SETQ FUNKTAETNR 0)
               (SETQ AXNR 0)
               (SETQ MENGEN (CONS))
               (SETQ OPERATOREN)
               (SETQ IMPLEMOPS (CONS))
               (SETQ BENOPS (CONS))
               (SETQ FUNKS (CONS))
               (SETQ AXIOME (CONS)))
   NEUSCHLUESSEL
        (SETQ ADT-SCHLUESSEL (RATOM))
        (SETQQ UNGUELTIG NIL)
        (SELECTQ ADT-SCHLUESSEL
          (DATENTYP
              (REMPROPS (QUOTE (S DO X)))
              (PUTPROPS D DATENTYP T)
              (SETQ ADT-NAME
                    (RATOM-NE-SEM "ADT-NAME FEHLT"))
```

```
    <COND
     ((EQUAL (GETP (QUOTE D)
                   (QUOTE PARAMETER))
             NIL)
      NIL)
      (T (SETQ ADT-NAME
               (PACK (LIST ADT-NAME (QUOTE -)
                           PARAMETER>
    (SETBRK (QUOTE (180)) 1)
    <SET (SETQ ADT-NAMEVARS
               (PACK (LIST ADT-NAME
                           (QUOTE VARS>
    <SET (SETQ ADT-NAMEFNS
               (PACK (LIST ADT-NAME
                           (QUOTE FNS>
    (SET (PACK (LIST ADT-NAME (QUOTE COMS)))
         ADT-NAME)
    (SETQ ADT-TRAEGER
          (RATOM-NE-SEM "ADT-TRAEGER (D BZW S)
                        FEHLT BEI ADT-NAME))
    (*COM* "REGEL ;1")
    (SETZEIG* ADT-TRAEGER (QUOTE NAME)
              ADT-NAME)
    (SELECTQ ADT-TRAEGER
      (D (RATOM-NE-SEM)
         (PRINTL "BEI ADT-TRAEGER D KEINE WEITEREN
                 ANGABEN ERLAUBT. IN" ADT-NAME
                 "/ IGNORIERT")
         (SKIPSEM))
      (S (AND (EQ (RATOM-NE-SEM "GEDAECHTNISANGABE
                                FEHLT BEI" ADT-NAME
                                (QUOTE S))
              (EQ (RATOM-NE-SEM "GEDAECHTNISANGABE
                                FEHLT BEI" ADT-NAME
                                "S =")
                  (QUOTE D))
              (PROGN (*COM* "REGEL ;2")
                     (SAMMELGED))
              (EVAL SPRUNG)))
       (PRINTL "ADT-TRAEGER" ADT-TRAEGER
               "UNZULAESSIG FUER" ADT-NAME))
    (SKIPSEM))
(KOMPONENTENMENGE
    (SETQ X (RATOM-NE-SEM "KOMPONENTENMENGE NICHT
                          ZULAESSIG, NUR X")))
    (COND
     ((EQ X (QUOTE X))
       (TCONC MENGEN X)
       (SETZEIG* X (QUOTE KOMPONENTENMENGE)))
      (T (PRINTL X "ALS KOMPONENTENMENGE
                 ZULAESSIG, NUR X")))
    (RATOM-NE-SEM))
(SONSTIGE (GO NEUSCHLUESSEL))
(MENGEN
    (PROG (MENGE ART)
     NEUMENG
```



```
          (SETQ MENGE (RATOM-NE-KOMSEM
                        (QUOTE NEUMENG)
                        "MENGENANGABE FEHLT"))
          (*COM* "REGEL ;3")
          <COND
           ((NULL (MEMB MENGE (CAR MENGEN)))
             (RPLACD MENGE)
             (TCONC MENGEN MENGE)
             (SETZEIG* MENGE (QUOTE MENGE>
          (RATOM-NE-KOMSEM (QUOTE NEUMENG)
                           "MENGE" MENGE
                           ": ARTANGABE FEHLT")
          (SETQ ART (RATOM-NE-KOMSEM (QUOTE NEUMENG)
                           "MENGE" MENGE
                           ": ARTANGABE FEHLT"))
          (SETZEIG* MENGE (QUOTE MENGE)
                    ART)
          (RATOM-NE-KOMSEM (QUOTE NEUMENG)))
    (EVAL SPRUNG))
(BENUTZUNG
    (PROG (PARMS FORM OP1)
      NEUBEN
          (READC)
          (SETQ PARMS (READ))
          (RATOM-NE-KOMSEM (QUOTE NEUBEN)
            "BENUTZUNGSDEF. UNVOLLSTAENDIG IN"
            ADT-NAME)
          (*COM* "REGEL ;4")
          (SETQ FORM (READ))
          (SETQ OP1 (SQCDR PARMS))
          (RPLACD OP1)
          (SETZEIG* OP1 (QUOTE BENUTZUNGSDEF)
                    (QUOTE LAMBDA)
                    PARMS
                    (BENUTZUNGSIMPLEM FORM))
          (TCONC BENOPS OP1)
          <MAPC (WELOPS FORM)
                (FUNCTION (LAMBDA (OP)
                 (COND
                  ((NULL (MEMB OP (CAR IMPLEMOPS)))
                    (TCONC IMPLEMOPS OP)
                    (RPLACD OP)
                    (SETZEIG* OP (QUOTE OP)
                                 (QUOTE IMPLEM>
          (RATOM-NE-KOMSEM (QUOTE NEUBEN)))
    (EVAL SPRUNG))
((OPERATOREN HILFSOPERATOREN VORGEGEBEN)
    (PROG (OP FUNKTAET)
      NEUOP
          (SETQ FUNKTAET (CONS))
          (SETQ OP (RATOM-NE-KOMSEM (QUOTE NEUOP)
                     "OPERATORANGABE FEHLT IN"
                      ADT-NAME)
          (SETQ FUNKTAETNR (ADD1 FUNKTAETNR))
          (EINSORT OP (QUOTE OPERATOREN))
          (TCONC FUNKTAET (NAMBILD "F" FUNKTAETNR))
```

```
        (TCONC FUNKTAET OP)
        (TCONC FUNKS (CAR FUNKTAET))
        (SETZEIG* OP (QUOTE OP)
            (SELECTQ ADT-SCHLUESSEL
                (OPERATOREN
                    (*COM* "REGEL ;5")
                    (AND (NULL (MEMB OP
                                     (CAR IMPLEMOPS)))
                         (RPLACD OP)
                         (TCONC IMPLEMOPS OP))
                    (QUOTE IMPLEM))
                (HILFSOPERATOREN
                    (*COM* "REGEL ;6")
                    (OR (MEMB OP (CAR IMPLEMOPS))
                        (RPLACD OP))
                    (QUOTE HILFE))
                (VORGEGEBEN (*COM* "REGEL ;7")
                    (RPLACD OP)
                    (QUOTE VORGEGEBEN))
                NIL))
    NEUSORT
        (SETQ SORT
              (RATOM-NE-KOMSEM (QUOTE NEUOP)))
        (AND (EQ SORT (QUOTE %>))
             (EQ (CADR FUNKTAET)
                 (QUOTE -))
             (RPLACA (CDR FUNKTAET)
                     (QUOTE -%>))
             (GO NEUSORT))
        (TCONC FUNKTAET SORT)
        (GO NEUSORT))
    (EVAL SPRUNG))
(AXIOME
    (PROG (AXIOM)
      NEUAX
        (SETQ AXIOM (CONS))
        (SETQ AXNR (ADD1 AXNR))
        (TCONC AXIOM (NAMBILD "A" AXNR))
        (TCONC AXIOME (CAR AXIOM))
     NEUATOM
        (SETQ ATOM
              (RATOM-NE-KOMSEM (QUOTE NEUAX)))
        <AND (EQ ATOM (QUOTE =))
             (COND
              <(EQ (CAR AXIOM)
                   (CDR AXIOM))
               (*COM* "UEBERNEHME FUER"
                      (CAAR AXIOM) "LINKE SEITE
                      VORIGEN AXIOMS")
               (NCONC AXIOM (COPY LINKSEIT))
               (RPLACD AXIOM (LAST (CAR AXIOM>
              (T (SETQ LINKSEIT (COPY (CDAR AXIOM>
        (TCONC AXIOM ATOM)
        (GO NEUATOM))
    (EVAL SPRUNG))
(; (PRINTL "ADT-EINGABE BEENDET.")
```



```
            (TERPRI)
            (CLOSEF EINGABEDATEI)
            (COND
             ((EQ ALTEINGABE T)
               (INFILE))
             (T (INFILE ALTEINGABE)))
            (*COM* "ERSETZUNG ALTEINGABE"))
            (PROG NIL
                  (PRINTL (QUOTE ADT-SCHLUESSEL)
                          ADT-SCHLUESSEL "NICHT ZULAESSIG")
                  (SETQQ UNGUELTIG T)))
        (COND
         (UNGUELTIG (SETQQ UNGUELTIG NIL)
                    (GO NEUSCHLUESSEL))
         (T NIL)))
(CHRESET)
<SETQ URBINDUNGEN (LIST (CONS (QUOTE *ADT)
                              ADT-TRAEGER)
                        (CONS (QUOTE *ADT-NAME)
                              ADT-NAME)
                        (QUOTE (S . S))
                        (QUOTE (D . D))
                        (QUOTE (X . X>
(MAPC (LIST (QUOTE MENGEN)
            (QUOTE FUNKS)
            (QUOTE AXIOME)
            (QUOTE BENOPS)
            (QUOTE IMPLEMOPS))
      (FUNCTION ENTCONC))
(NCONC (SET ADT-NAMEVARS
            (APPEND (QUOTE (PROP ALL S D DO -%> = &))
                    (COPY MENGEN)))
       (COPY BENOPS))
(NCONC (EVAL ADT-NAMEVARS)
       (COPY OPERATOREN))
<NCONC (SET ADT-NAMEVARS (LIST (EVAL ADT-NAMEVARS)))
       (COPY (QUOTE (ADT-NAME ADT-NAMEVARS ADT-NAMEFNS MENGEN
                              OPERATOREN FUNKS AXIOME BENOPS
                              IMPLEMOPS URBINDUNGEN VERWENDET>
(PRINTDEF (LIST MENGEN BENOPS OPERATOREN FUNKS AXIOME
                URBINDUNGEN))
(TERPRI>>
```

<-%>-PHASE <LAMBDA NIL ... >>

Aufruf: Von ADT-IMPLEM.
Vorauss.: RADT muß durchlaufen sein.
Ergebnis: Führt die Funktionalitatenvorauswahl durch und probiert alle ->-Regeln.

```
<-%>-PHASE
  <LAMBDA NIL
    (TERPRI)
    (PRINT "START -> PHASE")
    (FUNKSKLASS)
    (SETQ VERWENDET (CONS))
    (MAPC -%>-REGELN (FUNCTION (LAMBDA (NREG)
           (PROBIERREGEL NREG (GETP (QUOTE -%>)
                                    (QUOTE PRUEFLINGNAME>>    ■
```

<=-PHASE <LAMBDA NIL ... >>

Aufruf: Von ADT-IMPLEM.
Vorauss.: Alle vorherigen Phasen müssen durchlaufen sein.
Ergebnis: Führt die Axiomen und Operatorvorauswahl durch und probiert die =-Regeln.

```
<=-PHASE
  <LAMBDA NIL
    (TERPRI)
    (PRINT "START = PHASE")
    (AXIOMENKLASS)
    (MAPC =-REGELN (FUNCTION (LAMBDA (NREG)
           (PROBIERREGEL NREG (GETP (QUOTE =)
                                    (QUOTE PRUEFLINGNAMEN>>   ■
```

<&-PHASE <LAMBDA NIL ... >>

Aufruf: Von ADT-IMPLEM.
Vorauss.: Die vorhergehenden Phasen müssen durchlaufen sein.
Ergebnis: Führt &-Regelvorauswahl und Operatorenklassifikation durch. Die noch möglichen &-Regeln werden der Reihe nach ausprobiert. Wenn bei einer erfolgreichen Regelanwendung ADTNEUEIG auf T gesetzt wurde (d.h. eine den ganzen Datentyp und nicht nur eine einen Operator oder eine Menge betreffende Eigenschaft gefolgert wurde), wird die &-Regelvorauswahl jeweils aktualisiert.

```
<&-PHASE
  <LAMBDA NIL
    (TERPRI)
    (PRINT "START  & PHASE")
    (SETQ &-NAMEN (MAPCAR &-REGELN (QUOTE CAR)))
    (REGELVORAUSWAHL &)
    (OPERATORENKLASS)
    (PROG (ADTNEUEIG)
      NEXT(PROBIERREGEL (ASSOC (SQCDR &-NAMEN)
                               &-REGELN)
                        OPERATOREN)
          (OR &-NAMEN (RETURN))
          (COND
```



```
            (ADTNEUEIG (REGELVORAUSWAHL &)
                       (SETQ ADTNEUEIG)))
          (GO NEXT>>                                          ■
```

<I-PHASE <LAMBDA NIL ... >>

Aufruf: Von ADT-IMPLEM.
Vorauss.: Die vorherigen Phasen müssen durchlaufen sein.
Ergebnis: Führt die Vorauswahl für die I-Phase (Aufruf von VORAUSSTEST) aus und testet dann für jeden zu implementierenden Operator die einzelnen I-Regeln durch Aufruf von UEBERS.

```
<I-PHASE
  <LAMBDA NIL
    (TERPRI)
    (PRINT "START I PHASE")
    (SETQ I-NAMEN (MAPCAR I-REGELN (FUNCTION CAR)))
    (REGELVORAUSWAHL I)
    (MAPC IMPLEMOPS (FUNCTION UEBERS>>                        ■
```

<B-PHASE <LAMBDA NIL ... >>

Aufruf: Von ADT-IMPLEM.
Vorauss.: Die vorherigen Phasen müssen durchlaufen sein.
Ergebnis: berechnet die Namen der B-Phasen-Regeln, führt dann für alle zu implementierenden Operatoren die Funktion BEREINIGE durch.
Wert: NIL

```
<B-PHASE
  <LAMBDA NIL
    (TERPRI)
    (PRINT "START B PHASE")
    (SETQ B-NAMEN (MAPCAR B-REGELN (FUNCTION CAR)))
    <MAPC IMPLEMOPS (FUNCTION (LAMVDA (X)
           (BEREINIGE X B-NAMEN B-REGELN
                        B-ATOMARE-SCHLUESSEL
                        B-NICHTATOMARE-SCHLUESSEL>
    (ENTCONC VERWENDET>>                                      ■
```

<L-PHASE <LAMBDA NIL ... >>
"Lisp-PHASE"

Aufruf: Von ADT-Implem.
Vorauss.: Die vorherigen Phasen müssen abgeschlossen sein.
Ergebnis: Es wird die in [EKT 79, Kapitel 6.4. (Aufruforganisation)] erklärte Funktion "adt-name" und die Assoziationsliste "adt-name" der Operatordefinitionen auf axiomatischer Ebene definiert. Die Merklisten für MAKEFILE werden so gesetzt, daß nur die Implementierung gerettet wird.
Bem.: Veränderungen gegenüber der Dokumentation in [EKT 79] sind in [BO 80] beschrieben.

```
<L-PHASE
 <LAMBDA NIL
  (TERPRI)
  (PRINT "START  L PHASE")
  (SET ADT-NAME)
  (MAPC IMPLEMOPS (FUNCTION EINFUNK))
   <COND
    <<GETP (EVALA (QUOTE *ADT)
                  URBINDUNGEN)
           (QUOTE GEDSTRUKTUR))
      <SETQ GEDFUNKS (CONS (CONS (QUOTE DO)
                                 (CAR (EKL (GETP (QUOTE DO)
                                                 (QUOTE FUNK>
     (MAPC BENOPS (FUNCTION BENDEF))
     (MAPC IMPLEMOPS (FUNCTION EINGEDFUNK))
     (SETQ OPNAMEN (MAPCAR GEDFUNKS (FUNCTION CAR)))
     (PUTD ADT-NAME
           (SUBST OPNAMEN (QUOTE OPNAMEN)
             (SUBST ADT-NAME (QUOTE ADT-NAME)
               (SUBST GEDFUNKS (QUOTE GEDFUNKS)
                 (COPY (QUOTE (NLAMBDA (OP S . LISTE)
                                (SETQ ARGUMENTE LISTE)
                                (SETQ DATSTRUK S)
                                (SETQ S (EVAL S))
                                (PROG ((URD (CAR S)))
                                      (COND
                                        ((MEMBER OP
                                                 (QUOTE OPNAMEN))
                                          NIL)
                                        (T (PRINTL
                                            "ANWENDUNGSFEHLER:"
                                            OP "IST KEINE"
                                            (QUOTE ADT-NAME)
                                            "OPERATION!")
                                           (TERPRI)
                                           (RETURN)))
                                      (RETURN
                                        (EVALA
                                          (QUOTE
                                            (APPLY (EVAL OP)
                                              (CONS S LISTE)))
                                          (QUOTE GEDFUNKS>
```



```
    (T (EINFUNK (QUOTE DO))
       (PUTD ADT-NAME
             (SUBST ADT-NAME (QUOTE ADT-NAME)
               (COPY (QUOTE (NLAMBDA (OP . DLISTE)
                              (SETQ ARGUMENTE (CDR DLISTE))
                              (SETQ DATSTRUK (CAR DLISTE))
                              (SETQ DLISTE
                                    (APPEND
                                      (LIST
                                        (EVAL (CAR DLISTE)))
                                            (CDR DLISTE)))
                              (PROG ((URD (CAR DLISTE)))
                                    (COND
                                     ((ASSOC OP ADT-NAME)
                                       NIL)
                                     (T (PRINTL
                                         "ANWENDUNGSFEHLER:"
                                         OP "IST KEINE"
                                         (QUOTE ADT-NAME)
                                         "OPERATION!")
                                        (TERPRI)
                                        (RETURN)))
                                    (RETURN
                                      (EVALA
                                       (QUOTE (APPLY (EVAL OP)
                                                     DLISTE))
                                       ADT-NAME>
(PUT ADT-NAME (QUOTE PARAMETER)
     (GETP (QUOTE D)
           (QUOTE PARAMETER)))
(PUT ADT-NAME (QUOTE STRUKTUR)
     (COND
       ((EQUAL (GETP (QUOTE D)
                     (QUOTE PARAMETER))
               NIL)
        ADT-NAME)
       (T DATENTYP)))
(SET ADT-NAMEFNS (CONS ADT-NAME))
(SET ADT-NAMEVARS (LIST ADT-NAME (QUOTE VERWENDET)))
(RPLACA (PACK (LIST (QUOTE IMPLEM.)
                    ADT-NAME
                    (QUOTE COMS)))
        (LIST (LIST (QUOTE FNS)
                    ADT-NAME)
              (LIST (QUOTE VARS)
                    ADT-NAME
                    (QUOTE VERWENDET))
              (LIST (QUOTE PROP)
                    (QUOTE ALL)
                    ADT-NAME)))
(REMPROPS (QUOTE (D))
          (QUOTE PARAMETER>>
```

5.1.1.2. Die untergeordneten Funktionen

5.1.1.2.1. Hilfsfunktionen von RADT

<RATOM-NE-SEM <LAMBDA MELDUNGEN ... >>
"Read ATOM Not Equal SEMicolon"

Aufruf: Durch RADT.
Param.: MELDUNGEN - beliebig viele Texte und Formen.
Ergebnis: Es wird mittels RATOM ein Atom gelesen. ist es ein ";", so werden die Meldungen, falls vorhanden, gedruckt, und es wird zur Marke NEUSCHLUESSEL in RADT gesprungen, um die Angaben unter dem nächsten ADT-Schlüsselwort zu lesen.
Wert: Das gelesene Atom, wenn es kein ";" war.

```
<RATOM-NE-SEM
  <LAMBDA MELDUNGEN
    (SETQ ATOM (RATOM))
    (COND
      ((EQ ATOM (QUOTE ;))
        (AND MELDUNGEN (CAR MELDUNGEN)
             (PRINTL MELDUNGEN))
        (SELECTQ ADT-SCHLUESSEL
                 <DATENTYP
                   (COND
                    ((EQ ADT-TRAEGER (QUOTE S))
                      (SETQQ SPRUNG (GO NEUSCHLUESSEL))
                      (RETURN SPRUNG))
                    (T (GO NEUSCHLUESSEL>
                 (KOMPONENTENMENGE (GO NEUSCHLUESSEL))
                 ((AXIOME BENUTZUNG MENGEN OPERATOREN
                   HILFSOPERATOREN VORGEGEBEN)
                  (SETQ SPRUNG (GO NEUSCHLUESSEL))
                  (RETURN SPRUNG))
                 (PRINTL "FEHLER IN RATOM-NE-SEM")))
      (T ATOM>>                                          ■
```

<RATOM-NE-KOMSEM <LAMBDA (NEUZEIL . MELDUNGEN) ... >>
"Read ATOM Not Equal KOMma or SEMicolon"

Aufruf: Von den Alternativen in RADT, wo mehrere durch Komma getrennte Angaben unter demselben ADT-Schlüsselwort gelesen werden können: BENUTZUNG, OPERATOREN, HILFOPERATOREN, VORGEGEBEN, AXIOME.
Param.: Für NEUZEIL muß eine gequotete Marke angegeben werden;
für MELDUNGEN beliebig viele Texte und Formen.
Ergebnis: Es wird mittels RATOM-NE-SEM ein Atom eingelesen. Ist es ein ";" oder ",", so werden die Meldungen, falls angegeben, gedruckt. Bei "," wird zu der durch NEUZEIL bezeichneten Marke gesprungen, bei ";" zur

Marke NEUSCHLUESSEL.
Wert: Das gelesene Atom, wenn es kein "," oder ";" war.

```
<RATOM-NE-KOMSEM
  <LAMBDA (NEUZEIL . MELDUNGEN)
    (SETQ ATOM (RATOM-NE-SEM MELDUNGEN))
    (COND
      ((EQ ATOM (QUOTE ,))
        (AND MELDUNGEN (PRINTL MELDUNGEN))
        (APPLY* (FUNCTION GO)
                NEUZEIL))
      (T ATOM>>                                          ■
```

<SKIPSEM <LAMBDA NIL ... >>
"SKIP to SEMicolon"

Aufruf: Von der Alternative DATENTYP in RADT.
Ergebnis: Überliest mittels RATOM-NE-KOMSEM bis zum nächsten Semikolon und springt daher dann zur Marke NEUSCHLUESSEL in RADT.

```
<SKIPSEM
  <LAMBDA NIL
    (PROG LOOP
          (RATOM-NE-SEM)
          (GO LOOP>>                                     ■
```

<SAMMELGED <NLAMBDA NIL ... >>
"SAMMEL GEDächtnismengen"

Aufruf: Von Alternative DATENTYP in RADT, nach Einlesen von "S=D".
Ergebnis: Bewirkt Registrierung der folgenden Mengen als Gedächtnismengen:
Variablen: Die Variable MENGEN aus RADT wird erweitert.
Eigenschaften: Die Eigenschaft GEDSTRUKTUR wird an S, die Eigenschaft GEDAECHTNIS wird an die Mengen gehängt.
Bem.: Liefert keinen Wert, sonder springt bei ";" zu NEUSCHLUESSEL.

```
<SAMMELGED
  <NLAMBDA NIL
    (PROG (MENGE I)
          (SETQ I 0)
      NEUMENG
          (SETQ I (ADD1 I))
          (SETQ MENGE (RATOM-NE-SEM))
          (SETZEIG* (QUOTE S)
                    (QUOTE GEDSTRUKTUR)
                    MENGE)
```

```
        (RPLACD MENGE)
        (SETZEIG* MENGE (QUOTE GEDAECHTNIS)
                  I)
        (TCONC MENGEN MENGE)
        (GO NEUMENG>>                                    ■
```

<BENUTZUNGSIMPLEM <LAMBDA (FORM) ... >>
"erstelle IMPLEMentierungsfunktion einer BENUTZUNGSdefinition"

Aufruf: Von RADT in Alternative BENUTZUNG.
Param.: FORM ist eine Form, in der nur LAMBDA-Funktionen vorkommen.
Wert: Die aus FORM dadurch entstehende Form, daß in jedem Funktionsaufruf der "Funktionsname" durch APPLY* "Funktionsname" ersetzt wird.

```
<BENUTZUNGSIMPLEM
  <LAMBDA (FORM)
    (COND
      ((ATOM FORM)
        FORM)
      (T (CONS (QUOTE APPLY*)
               (CONS (CAR FORM)
                     (MAPCAR (CDR FROM)
                             (FUNCTION BENUTZUNGSIMPLEM>>    ■
```

<NAMBILD <LAMBDA (ZEICH NR) ... >>
"BILDung eines Funktionalitäts/Axiomen-NAMens"

Aufruf: Von RADT beim Einlesen eines neuen Prüflings.
Param.: ZEICH - ein Artzeichen: F oder A;
NR - eine natürliche Zahl.
Wert: Ein Atom, welches aus der Konkatenation des Artzeichens und der mindestens 2-ziffrigen Darstellung der angegebenen Zahl entsteht.
Bem.: Die zweiziffrige Darstellung der Zahl NR ist notwendig, um die gewünschte Sortierreihenfolge durch ALPHORDER abfragen zu können.

```
<NAMBILD
  <LAMBDA (ZEICH NR)
    (PACK (LIST ZEICH (COND
                        ((AND (NUMBERP NR)
                               (LESSP NR 10))
                          "0")
                        (T ""))
                NR>>                                     ■
```


<CHRESET <NLAMBDA NIL ... >>
"CHaracter interpretation RESET"

Aufruf: Am Ende von RADT.
Ergebnis: Die Zeichendeutung für "+", "-" und ">" wird wieder auf den Normalwert gesetzt.
Bem.: Muß vom Benutzer aufgerufen werden, wenn durch Abbruch während RADT (z.B. durch Dateiende) noch die geänderte Zeichendeutung gilt.

```
<CHRESET
  <NLAMBDA NIL
    (SETSEPR (QUOTE (78)) 0)
    (SETBREAK (QUOTE (126 180)) 0>>                      ■
```

5.1.1.2.2. Hilfsfunktionen der ->- und =-PHASE

<FUNKSKLASS <NLAMBDA NIL ... >>
"FUNKtionalitäten-KLASSifizierung"

Aufruf: Am Anfang von ->-PHASE.
Vorauss.: RADT muß schon durchgeführt worden sein. Wert von FUNKS muß eine geordnete Assoziationsliste der Funktionalitäten sein.
Ergebnis: Die Funktionalitäten werden danach klassifiziert, ob sie ein D, X oder eine sonstige Menge im Definitionsbereich (links) oder im Wertebereich (rechts) haben.
Eigenschaften: "->" erhält unter der Eigenschaft FUNKSKLASSEIGS den Wert (DL XL ML DR XR MR). Die Eigenschaften DL, XL, ML DR, XR, MR von "=>" erhalten als Wert die entsprechenden alphabetisch geordneten Funktionalitätsnamen.
Wert: Die Liste <ADT-Name>VARS der für den ADT definierten Variablen.

```
<FUNKSKLASS
  <NLAMBDA NIL
    <SETQ FUNKSKLASSEIGS (COPY (QUOTE FUNKNAMEN
                                      DL XL ML DR XR MR>
    (RPLACD (QUOTE -%>))
    (MAPC FUNKS (FUNCTION FUNKKLASS))
    (NCONC1 (EVAL ADT-NAMEVARS)
            (COPY (QUOTE FUNKSKLASSEIGS>>                ■
```

<FUNKKLASS <LAMBDA (FUNKTAET ...) ... >>
"FUNKtionalitäts-KLASSifizierung"
"FUNKTAET - FUNKtionaliTAET"

Aufruf: Wiederholt von FUNKSKLASS.
Ergebnis: Ordnet den Namen der angegebenen Funktionalität in die entsprechenden Eigenschaftswerte ein.

```
<FUNKKLASS
  <LAMBDA (FUNKTAET SEITE)
    (SETQ SEITE (QUOTE L))
    (ADDPROP (QUOTE -%>)
             (QUOTE PRUEFLINGNAMEN)
             (CAR FUNKTAET))
    (MAPC (CDDR FUNKTAET)
          (FUNCTION MENGKLASS>>
```

<MENGKLASS LAMBDA (MENGE ...) ...)
"gemäß MENGe KLASSifiziere Funktionalität"

Aufruf: Von FUNKSKLASS aus für jedes Element von FUNKTAET.
Vorauss.: SEITE muß den Wert L oder R haben;
FUNKTAET muß eine Funktionalität darstellen.
Param.: MENGE ist ein Mengenname.
Ergebnis: Gemäß SEITE und MENGE wird der Name der Funktionalität R gesetzt.
Wert: Der Name der Liste, in die eingeordnet wurde.

```
<MENGKLASS
  <LAMBDA (MENGE MENGZEICH LISTENNAME)
    (SETQ MENGZEICH MENGE)
    (AND (GETP MENGE (QUOTE MENGE))
         (SETQ MENGZEICH (QUOTE M)))
    (SETQ LISTENNAME (PACK (LIST MENGZEICH SEITE)))
    (SELECTQ MENGZEICH
             ((D X M)
               (OR (MEMB (CAR FUNKTAET)
                         (GETP (QUOTE -%>)
                               LISTENNAME))
                   (ADDPROP (QUOTE -%>)
                            LISTENNAME
                            (CAR FUNKTAET)))
               LISTENNAME)
             (-%> (SETQQ SEITE R))
             (PRINTL "MENGE" MENGE "AUS FUNTIONALITAET
                     FUNKTIONALITAET "NICHT VEREINBART">>
```


<AXIOMENKLASS <NLAMBDA NIL ... >>
"AXIOMENKLASSifizierung"

Aufruf: Von =-PHASE.
Vorauss.: RADT und ->-PHASE müssen erfolgreich abgeschlossen sein. In AXIOME steht die Assoziationsliste der Axiome. In =-CHAREIGS stehen die Namen der in der ->-Phase eingeführten Eigenschaften, die auswahlcharakteristisch für die von den Axiomen definierten Operatoren sind.
Ergebnis: Die Axiome werden alle klassifiziert, d.h. ihre Namen in folgende Listen eingeordnet:

Variablen:

L1	-	Axiome, die links genau ein Klammerpaar haben.
L2	-	Axiome, die links mindestens zwei Klammerpaare haben.
R1	-	Axiome, die rechts mindestens ein Klammerpaar haben.
RC	-	Axiome, die rechts eine Konstante (UNDEF oder Zahl) stehen haben.
RX	-	Axiome, die rechts eine auf der linken Seite gebundene Variable oder DO haben.
<Eigname>	-	Für alle Eigenschaftsnamen aus der Liste wird eine Liste mit diesem Namen <Eigname> gebildet, bestehend aus den Namen aller Axiome, deren definierter äußerer Operator die Eigenschaft <Eigname> in der ->-Phase erhalten hat. Außerdem werden die Namen der erzeugten Listen eingeordnet in:
<ADT-Name>VARS	-	MAKEFILE-Liste aller auf dem aktuellen Stand der Bearbeitung wichtigen Variablen und Eigenschaften.

```
<AXIOMENKLASS
  <NLAMBDA NIL
    <SETQ AXIOMENKLASSEIGS (COPY (QUOTE (AXIOMNAMEN
                                         L1 L1 RC RX R1>
    <NCONC AXIOMENKLASSEIGS (MAPCAR =-CHAREIGS
                                    (FUNCTION (LAMBDA (L)
                                      (PACK (STRICHLISTE L>
    (RPLACD (QUOTE =))
    (MAPC AXIOME (FUNCTION AXIOMKLASS))
    (NCONC1 (EVAL ADT-NAMEVARS)
            (COPY (QUOTE AXIOMENKLASSEIGS>>
```

<AXIOMKLASS <LAMBDA (AXIOM ...) ... >>
"AXIOMKLASSifizierung"

Aufruf: Von AXIOMENKLASS für jedes Axiom.
Param.: AXIOM ist ein AXIOM mit NAMEN.
Ergebnis: Das Axiom wird überprüft und an die entsprechenden Listen mit TCONC angefügt.

```
<AXIOMKLASS
  <LAMBDA (AXIOM ERG)
    (SETQ ERG (LINKSTEST AXIOM))
    (ADDPROP (QUOTE =)
             ERG
             (CAR AXIOM))
    (SETQ ERG (RECHTSTEST AXIOM))
    (ADDPROP (QUOTE =)
             ERG
             (CAR AXIOM))
    (ADDPROP (QUOTE =)
             (QUOTE PRUEFLINGNAMEN)
             (CAR AXIOM))
    (MAPC =-CHAREIGS (FUNCTION PRUEFCHAREIGAX>>
```

<LINKSTEST <LAMBDA (AXIOM ...) ... >>
"TESTe LINKE Axiomenseite"

Aufruf: Von AXIOMENKLASS.
Param.: AXIOM ist ein Axiom mit Namen.
Wert: L0, L1 bzw. L2, je nachdem, wieviele öffnende Klammern die linke Axiomenseite enthält.

```
<LINKSTEST
  <LAMBDA (AXIOM ERGEB)
    (SETQ ERGEB (QUOTE L0))
    (AND (EQ (CADR AXIOM)
             (QUOTE %()
         (SETQ ERGEB (QUOTE L1)))
    (AND (EQ (CADR (CDDR AXIOM))
             (QUOTE %()
         (SETQ ERGEB (QUOTE L2)))
    ERGEB>>
```

<RECHTSTEST <LAMBDA (AXIOM) ... >>
"TESTe RECHTe Axiomseite"

Aufruf: Von AXIOMENKLASS.
Param.: AXIOM ist ein Axiom mit Namen.
Wert: RC, RX, bzw. R1 je nach der Art der rechten Axiomenseite.


```
<RECHTSTEST
  <LAMBDA (AXIOM)
    (PROG (A HILF RECHTS)
          (SETQ A (COPY AXIOM))
          (SETQ HILF (MEMB (QUOTE =)
                           A))
          (SETQ RECHTS (CADR HILF))
          (AND (EQ RECHTS (QUOTE %())
               (RETURN (QUOTE R1)))
          (RPLACD HILF NIL)
          (AND (MEMB RECHTS (CONS (QUOTE DO)
                                  A))
               (NULL (MEMB RECHTS OPERATOREN))
               (RETURN (QUOTE RX)))
          (RETURN (QUOTE RC>>
```

<PRUEFCHAREIGAX <LAMBDA (CHAREIG ...) ... >>
"PRUEFe CHARakteristische EIGenschaften von AXiomen"
"CHAREIG - CHARakteristische EIGenschaftsangabe"

Aufruf: Von AXIOMENKLASS.
Vorauss.: Die Variable AXIOM muß ein Axiom mit Namen als Wert haben, die TCONC-Liste mit (EVAL (CAR EIG)) als Namen muß existieren.
Param.: CHAREIG - Liste, deren 1. Element Name einer Eigenschaft ist, deren 2. Element eventuell Name eines Wertes ist.
Ergebnis: Es wird geprüft, ob der vom Axiom definierte Operator die Eigenschaft (mit eventuell einem Wert) EIG hat. Wenn ja, wird der Axiomenname an die entsprechende Liste mit dem Namen <Eigenschaft> bzw. <Eigenschaft>-<Wert> mit TCONC angehängt.
Wert: Die neue TCONC-Liste.

```
<PRUEFCHAREIGAX
  <LAMBDA (CHAREIG WERT)
    (SETQ WERT (GETP (CADDR AXIOM)
    (AND WERT (OR (NOT (CDR CHAREIG))
                  (ASSOC (CADR CHAREIG)
                         WERT))
          (ADDPROP (QUOTE =)
                   (PACK (STRICHLISTE CHAREIG))
                   (CAR AXIOM>>
```

<PROBIERPRUEFLING <LAMBDA (PRUEFLINGNAME ...) ... >>

Aufruf: In ->- und =-Phase wiederholt von PROBIERREGEL.
Vorauss.: NREGEL, PRUEFLINGNAME und PHASE müssen definiert sein.
Param.: PRUEFLINGNAME - Ein Funktionalitätsname Fmm oder ein Axiomenname Ann.
Ergebnis: Wendet die in NREGEL stehende, benannte Regel auf den durch PRUEFLINGNAME bezeichneten Prüfling mittels REGELPRUEF an, wobei die Bindung für *OP1 als Vorbindung übergeben wird.
Wert: Wie bei REGELPRUEF.

```
<PROBIERPRUEFLING
  <LAMBDA (PRUEFLINGNAME PRUEFLING)
    (*COM* (QUOTE PROBIERPRUEFLING)
           (QUOTE REGEL)
           REGELNAME
           (QUOTE AUF)
           (QUOTE PRUEFLING)
           PRUEFLINGNAME)
    <SETQ PRUEFLING
          (CDR (ASSOC PRUEFLINGNAME
                      (SELECTQ PHASE
                          (-%> FUNKS)
                          (= AXIOME)
                          (& NIL)
                          (*COM* "UNGUELTIGE PHASE"
                                 PHASE>
    (REGELPRUEF (CDR NREGEL)
                PRUEFLING
                (CONS (QUOTE *OP1)
                      (SELECTQ PHASE
                               (-%> (CAR PRUEFLING))
                               (= (CADR PRUEFLING))
                               (& PRUEFLINGNAME)
                               NIL))
                URBINDUNGEN>>
```

5.1.1.2.3. Hilfsfunktionen der &-, I- und B-Phase

<UEBERS <LAMBDA (OPI ...) ... >>
"UEBERSetze"
"OPI - Operator, der implementiert werden soll"

Aufruf: Kontrollprogramm.
Vorauss.: Es muß die &-Phase beendet sein, folgende Variablen sind bekannt: I-NICHT-ATOMARE-SCHLUESSEL, I-ATOMARE-SCHLUESSEL, I-NAMEN, I-REGELN; außerdem alle Werte der Elemente der Listen I-NICHT-ATOMARE-SCHLUESSEL und I-ATOMARE-SCHLUESSEL.

Param.: OPI ist ein Operator mit der Eigenschaft OP IMPLEM.
Ergebnis: Der Wert der Eigenschaft FUNK des Operators OPI wird ersetzt durch seine "Übersetzung".
Wert: Die "Übersetzung".
Bem.: Die Prozedur UEBERS setzt die Variable PRUEFLINGNAME, ersetzt variable Schlüssel durch ihre aktuellen Werte (falls vorhanden), sonst durch NIL. Dann wird die Prozedur DURCHL aufgerufen, die den Prüfling, d.h., den Wert der Eigenschaft FUNK des behandelten Operators durchläuft und nach Schlüsselworten sucht. DURCHL übergibt den geänderten Prüfling, der unter der Eigenschaft FUNK gespeichert wird.

```
<UEBERS
  <LAMBDA (OPI HATOM HNATOM FOPI RESULT IPOINTER IPOI2
           OHNEZWEI TIEFE HILFSOP EL OBJ IPRUEFL SCHLUESSEL)
    (TERPRI)
    (PRINTL "—— ICH BEARBEITE" OPI "——")
    (*COM* "ERSETZEN DER VARIABLEN,NICHTATOMAREN SCHLUESSEL:")
    (SETQ PRUEFLINGNAME (SETQ FOPI OPI))
    <SETQ HNATOM (MAPCAR I-NICHTATOMARE-SCHLUESSEL
                         (FUNCTION (LAMBDA (X)
                                   (COND
                                     ((VARIABLEP X)
                                       (SUCHBI X FOPI))
                                     (T X>
    (*COM* "ERSETZEN DER VARIABLEN ATOMAREN SCHLUESSEL:")
    <SETQ HATOM (MAPCAR I-ATOMARE-SCHLUESSEL
                        (FUNCTION (LAMBDA (X)
                                  (COND
                                    ((VARIABLEP X)
                                      (SUCHBI X FOPI))
                                    (T X>
    (*COM* "HOLEN DES ZU ERSETZENDEN PRUEFLINGS UND UEBERGABE
            AN DIE DURCHLAUF- UND ERSETZFKT. D U R C H L:")
    <DURCHL (COPY (GETP OPI (QUOTE FUNK>
    (PUT OPI (QUOTE FUNK)
         RESULT)
    (PRINTL "ERGEBNIS DER" PHASE "PHASE FUER" OPI (QUOTE :))
    <PUTD (QUOTE ERG-FUNK)
          (CAR (EKL (GETP OPI (QUOTE FUNK>
    (PRETTYPRINT (QUOTE (ERG-FUNK>>
```

<OPERATORENKLASS <LAMBDA NIL ... >>

Aufruf: Von &-PHASE.
Bem.: Wie FUNKSKLASS oder AXIOMENKLASS.

```
<OPERATORENKLASS
  <LAMBDA NIL
    <SETQ OPERATORENKLASSEIGS
      (MAPCAR &-CHAREIGS (FUNCTION (LAMBDA (L)
                                   (PACK (STRICHLISTE L>
    (RPLACD (QUOTE &))
    (MAPC OPERATOREN (FUNCTION OPERATORKLASS))
    (NCONC1 (EVAL ADT-NAMEVARS)
            (COPY (QUOTE OPERATORENKLASSEIGS>>                ■
```

<OPERATORKLASS <LAMBDA (PRUEFLINGNAME) ... >>

Aufruf: Von OPERATORENKLASS.
Bem.: Wie FUNKKLASS oder AXIOMKLASS.

```
<OPERATORKLASS
  <LAMBDA (PRUEFLINGNAME)
    (MAPC &-CHAREIGS (FUNCTION PRUEFCHAREIGOP>>            ■
```

<PRUEFCHAREIGOP <LAMBDA (EIG) ... >>

Aufruf: Von OPERATORKLASS.
Bem.: Wie PRUEFCHAREIGAX.

```
<PRUEFCHAREIGOP
  <LAMBDA (EIG)
    (AND (SUCHWERT (CDR EIG)
                   (GETP PRUEFLINGNAME (CAR EIG)))
         (ADDPROP (QUOTE &)
                  (CAR EIG)
                  PRUEFLINGNAME>>                          ■
```

<DURCHL <LAMBDA (IPRUEFL ...) ... >>
"DURCHLäuft"
"IPRUEFL - Prüfling für die I-Phase"

Aufruf: von UEBERS.
Vorauss.: HATOM, HNATOM, I-NAMEN und FOPI müssen definiert sein. Dies wird durch die vorausgehende Funktion UEBERS geregelt.
Ergebnis: Zwischenergebnis für UEBERS.
Wert: "Übersetzung"
Bem.: Die Prozedur DURCHL durchläuft die zu verfeinerte (oder auch zu übersetzende) Liste von links nach rechts. Findet sie dabei einen Schlüssel, der mindestens eine Regel "anbietet", die bei dem vorliegenden Datentyp anwendbar sein könnte, so wird die Funktion WOWEITER aufgerufen, die die Regel anzuwenden versucht und einen Verweis auf die Stelle zurückgibt, an der DURCHL wieder einsetzen soll oder

selbst DURCHL aufruft.

```
<DURCHL
  <LAMBDA (IPRUEFL TR)
    (*COM* "BEKANNT SEIN MUESSEN: HATOM,HNATOM,I-NAMEN,FOPI")
    (COND
      ((NULL IPRUEFL)
        RESULT)
      ((<AND (NEQ (QUOTE %()
                  (CAR IPRUEFL))
             (MEMBER (CAR IPRUEFL)
                     HATOM)
             (SETQ TR (INTERSECTION I-NAMEN (CAAR IPRUEFL>
        (SETQ SCHLUESSEL (CAR IPRUEFL))
        (PRINTL "ATOMARER SCHLUESSEL" SCHLUESSEL " , DAZU
                EVTL. ANWENDBARE REGEL:" TR)
        (WOWEITER))
       ((<AND (EQ (QUOTE %()
                  (CAR IPRUEFL))
              (MEMBER (CADR IPRUEFL)
                      HNATOM)
              (SETQ TR (INTERSECTION I-NAMEN (CAADR IPRUEFL>
        (SETQ SCHLUESSEL (CADR IPRUEFL))
        (PRINTL "NICHTATOMARER SCHLUESSEL" SCHLUESSEL ", DAZU
                EVTL. ANWENDBARE REGEL:" TR)
        (WOWEITER))
       (T (*COM* "KEIN SCHLUESSEL (FUER DIESEN DATENTYP)
                 IN PHASE" PHASE)
          (SETQ RESULT (NCONC1 RESULT (CAR IPRUEFL)))
          (DURCHL (CDR IPRUEFL>>                           ■
```

<SUCHBI <LAMBDA (VS FOP ...) ... >>
"SUCHEBindung"
"VS - Variabler Schlüssel"
"FOP - Freier OPerator"

Aufruf: von UEBERS.
Param.: VS ist ein mit * beginnendes Atom;
FOP ist der Operator, für den die I-PHASE gerade läuft.
Vorauss.: Da evtl. VORAUSSTEST aufgerufen wird, PRUEFLINGNAME; durch UEBERS geregelt.
Ergebnis: Bindet einen variablen Schlüssel unter Verwendung von VARBIND.
Wert: Falls der Schlüssel gebunden werden konnte, der gebundene Wert, NIL sonst.
Bem.: Zum Binden des variablen Schlüssels wird, solange die Bindung nicht bekannt ist, die durch den Schlüssel bezeichnete Regel angewendet. Falls der Schlüssel mehrere Regeln bezeichnet, - dieser Fall liegt bei unserer Version nicht vor - so muß jede dieser Regeln angewendet werden, falls verschiedene Bindungen möglich sind, werden alle Alternativen in die Schlüsselliste aufgenommen werden müssen.

```
<SUCHBI
  <LAMBDA (VS FOP BINDE VEREINBARE KLAKO)
    (COND
      <(EQ VS (QUOTE *OP1))
        FOP
        (RPLACA FOP (CAR (QUOTE *OP1>
      (T (SETQ BINDE (LIST (CONS (QUOTE *OP1)
                                 FOP)))
         (COND
           ((CADR VS)
             (*COM* "HIER LIEGT EIN VARIABLER SCHLUESSEL VOR,
                     DER IN MEHREREN REGELN DURCH
                     UNTERSCHIEDLICHE WERTE ERSETZT WERDEN
                     KANN, D.H. DIE SCHLUESSELLISTE MUSS
                     ERWEITERT WERDEN DERART, DASS JEDER
                     REALE SCHLUESSEL DARIN AUFGENOMMEN WIRD")
             (PRINT " LIES DEN KOMMENTAR")
             NIL)
           (T (PROG ((REGEL (CDR (ASSOC (CAAR VS)
                                        I-REGELN)))
                     (REGELNAME (CAAR VS))
                     H)
                LOP (COND
                      ((NEQ (SETQ H (VARBIND VS NIL T))
                            VS)
                       (RETURN (PROGN (RPLACA H (CAR VS))
                                      H)))
                      ((VORAUSSTEST (CAR REGEL))
                       (SETQ REGEL (COND
                                     ((NEQ (CDR REGEL)
                                           (QUOTE =%>))
                                      (CDR REGEL))
                                     (T NIL)))
                       (*COM* "ES WERDEN DIE VORAUSSETZUNGEN
                               SOLANGE GEPRUEFT BIS DER
                               SCHLUESSEL GEB. WERDEN KANN")
                       (GO LOP))
                      (T (RETURN NIL>>                    ■
```

<SUCHE <LAMBDA (SW IP ...) ... >>

Aufruf: von DURCHL, bei Auffinden eines Schlüssels.
Vorauss.: Variablen TR, IPRUEFL (aus DURCHL), RESULT, FOPI (aus UEBERS) müssen definiert sein.
Ergebnis: Falls eine Regel anwendbar ist, so wird IPRUEFL (aus DURCHL) durch den neuen Wert ersetzt. Bei Erfolg wird DURCHL wieder für IPRUEFL aufgerufen, bei Mißerfolg fährt DURCHL mit (CDR IPRUEFL) fort.
Wert: ruft seinerseits DURCHL auf.
Bem.: Es wird versucht eine der möglichen Regeln anzuwenden, d.h., Aufruf von REGELPRUEF bis eine Anwendung erfolgreich ist, oder keine Regel mehr vorliegt.


```
<WOWEITER
 <NLAMBDA NIL
  (PROG ((R TR))
   LOP (AND (CAR R)
            (SETQ REGELNAME (CAR R)))
       (COND
        <(NULL R)
          (APPLY* (COND
                    (<AND (EQ PHASE (QUOTE I))
                          (NULL (MEMB SCHLUESSEL
                                      (APPEND B-ATOMARE-SCHLUESSEL
                                              B-NICHTATOMARE-SCHLUESSEL>
                      (QUOTE *COM*))
                     ((QUOTE PRINTL)))
                  "KEINE REGEL ANWENDBAR AUF PRUEFLING" FOPI
                  "UND SCHLUESSEL" SCHLUESSEL)
          (RETURN (PROGN (SETQ RESULT (NCONC1 RESULT
                                              (CAR IPRUEFL)))
                         (DURCHL (CDR IPRUEFL>
        ((REGELPRUEF (CDR (ASSOC (CAR R)
                                 I-REGELN))
                     FOPI
                     (CONS (QUOTE *OP1)
                           FOPI)
                     URBINDUNGEN)
          (REGELERFOLG EL OBJ)
          (RETURN (DURCHL IPRUEFL)))
        (T (*COM* " NAECHSTE REGEL TESTEN")
           (SETQ R (CDR R))
           (GO LOP>>                                      ■
```

<WOWEITER <NLAMBDA NIL ... >>

Aufruf: von VORAUSSTEST, aber nur, wenn REGELPRUEF von WOWEITER aufgerufen wurde.
Voraus.: Die Variablen VEREINBARE, REGEL, BINDE (REGELPRUEF) und IPRUEFL (DURCHL) muessen definiert sein.
Params.: SW - was in der Regel als zu Suchendes ausgezeichnet ist;
IP - der bereits behandelte Prüfling, d.h., er beginnt mit dem Schlüsselwort, welches auch in der Regel vorliegt.
Wert: T, falls Ersetzung erfolgreich, NIL sonst.
Bem.: testet, ob der Suchwert im Prüfling gematcht werden kann und führt, falls ja, die geforderte Ersetzung durch.

```
<SUCHE
 <LAMBDA (SW IP HIHIHILFE ERSATZ ZEIGER PROSW FOLGERUNG)
  (*COM* "SW - SUCHWERT, IP - PARAMETER FUER DEN IPRUEFLING")
  (SETQ PROSW (COPY SW))
  (SETQ HIHIHILFE T)
  <COND
   (<LISTP (CADR (SETQ ZEIGER (MEMBER (QUOTE ...)
                                      PROSW>
     (RPLACA ZEIGER (QUOTE *...))
     (SETQ VEREINBARE (CONS (QUOTE (*... . LISTE))
                            VEREINBARE))
     (PROG ((FUERSW PROSW))
           (SETQ FUERSW (REMOVE (QUOTE REP)
                                (EINEEBENE FUERSW)))
           <SETQ FUERSW (APPEND (REMOVE (QUOTE USW)
                                        FUERSW)
                                (QUOTE ((USW>
           (SETQ HIHIHILFE
                 (VERGLEICH FUERSW IP BINDE VEREINBARE>
 <SETQ ERSATZ
       (EINEEBENE (MAPCAR (SETQ FOLGERUNG
                                (CADR (MEMB (QUOTE =%>)
                                            REGEL)))
                          (FUNCTION ECKIG>
 (SETQ ERSATZ (VARBIND ERSATZ NIL T))
 (COND
  (HIHIHILFE (COND
              <(LISTP (CAR PROSW))
                (COND
                 (<AND (EQ (LENGTH (CDAR PROSW))
                           1)
                       (EQUAL (CADR PROSW)
                              (QUOTE (USW>
                  (*COM* "ES WIRD EIN ATOM,WELCHES DER
                          GESAMTE SUCHWERT IST,ERSETZT")
                  (SETQ PROSW ERSATZ)
                  (SETQ IPRUEFL
                        (EINEEBENE (RPLACA IPRUEFL PROSW)))
                  (SETQ IP (EINEEBENE (RPLACA IP PROSW)))
                  T)
                 (T (VERGLEICH PROSW IP BINDE VEREINBARE>
             (T (VERGLEICH PROSW IP BINDE VEREINBARE>>          ■
```

<BEREINIGE <LAMBDA (OPB I-NAMEN I-REGELN I-ATOMARE-SCHLUESSEL I-NICHTATOMARE-SCHLUESSEL) ... >>

"OPB - OPerator für die Bereinigungsphase"

Aufruf: von B-PHASE.
Params.: OPB - der zu bereinigende Operator;
I-NAMEN - Liste der für die BEREINIGUNGSphase beim vorliegenden Datentyp verfügbaren Regelnamen (B-NAMEN);
I-REGELN - Liste der BEREINIGUNGSregeln (B-REGELN);
I-ATOMARE-SCHLUESSEL - Liste der atomaren Schlüssel

für die BEREINIGUNGSphase (B-ATOMARE-SCHLUESSEL);
I-NICHTATOMARE-SCHLUESSEL - Liste der nichtatomaren Schlüssel für die BEREINIGUNGSphase (B-NICHTATOMARE-SCHLUESSEL).
Vorauss.: Die Werte der Elemente der Schlüssellisten sind bekannt.
Bem.: Die Regelanwendung in der Bereinigungsphase verläuft genauso wie in der Implementierungsphase, es werden lediglich statt I-NAMEN B-NAMEN, statt I-REGELN B-REGELN usw. verwendet. BEREINIGE ruft nun DURCHL mit dem Parameter OPB auf.

```
<BEREINIGE
  <LAMBDA (OPB I-NAMEN I-REGELN I-ATOMARE-SCHLUESSEL
           I-NICHTATOMARE-SCHLUESSEL)
    (UEBERS OPB>>                                          ■
```

<REGELVORAUSWAHL <NLAMBDA (Z) ... >>

Aufruf: Von &-PHASE und I=PHASE.
Param.: Z ist & oder I.
Vorauss.: Die Datei mit der internen Darstellung der Regeln und Vorauswahlsteuerungen ist mit "LOAD" geladen worden. Es ist also die <Z>-ADTEIGS-LISTE definiert und es sind die <Z>-NAMEN definiert.
Ergebnis: Die in <Z>-ADTEIGS-LISTE enthaltenen symbolischen Namen der ADTEIGS werden der Reihe nach durchgearbeitet. Für jeden symbolischen Namen wird die Entschlüsselung aus der Property ORIGINAL geholt. Es wird überprüft, ob die ADTEIG mit den bislang erzielten Verfeinerungen für D, S oder *E übereinstimmt. Wenn das der Fall ist, wird die Liste der Regelnamen <Z>-NAMEN mit der Liste der für die ADTEIG möglichen Regeln (= Wert der ADTEIG) geschnitten und das Ergebnis <Z>-NAMEN zugewiesen. In <Z>-NAMEN stehen also jetzt die Namen der noch möglichen anwendbaren Regeln. Wenn die Regelaufbereitung fertig ist, werden die noch möglichen Regeln ausgedruckt.
Wert: NIL

```
<REGELVORAUSWAHL
 <NLAMBDA (Z)
  (PROG (ADTEIGSL NMN ORIG OBJ EIG WERT)
        <SETQ ADTEIGSL (PACK (LIST Z (QUOTE -ADTEIGS-LISTE>
        <SETQ NMN (PACK (LIST Z (QUOTE -)
                                (QUOTE NAMEN>
        <MAPC (EVAL ADTEIGSL)
              (QUOTE (LAMBDA (N)
                       (SETQ ORIG (GETP N (QUOTE ORIGINAL)))
                       (SETQ OBJ (CAR ORIG))
                       (SETQ EIG (CADR ORIG))
                       (SETQ WERT (CADDR ORIG))
                       (AND (OR (AND WERT
```

```
                    (MEMB* WERT (GETP (CDR
                      (ASSOC OBJ
                             URBINDUNGEN))
                      EIG)))
                (AND (NULL WERT)
                     (GETP (CDR (ASSOC OBJ
                                       URBINDUNGEN))
                           EIG)))
           (SET NMN (INTERSECTION (EVAL NMN)
                                  (EVAL N>
   (PRINTREGLIST (EVAL NMN)
                 10 "NOCH MOEGLICHE REGELN:">>          ■
```

5.1.1.2.4. Hilfsfunktionen der L-PHASE

<EINFUNK <LAMBDA (OP) ... >>
"EINgliedern von FUNK-Wert"

Aufruf: Für alle IMPLEMOPS und DO von L-PHASE.
Param.: OP - Name eines Operators, der die Eigenschaft FUNK besitzt.
Ergebnis: Die Liste "adt-name" wird um ein ASSOC-Paar mit der Definition des Operators OP auf axiomatischer Ebene erweitert.

```
<EINFUNK
  <LAMBDA (OP)
    (SET ADT-NAME (CONS <CONS OP (CAR (EKL (GETP OP
                                                 (QUOTE FUNK>
                        (EVAL ADT-NAME>>                ■
```

<BENDEF <LAMBDA (OP) ... >>
"BENutzungsDEFinition implementieren"

Aufruf: Von L-PHASE bei Gedächtnisstrukturen für jeden Benutzungsoperator.
Param.: OP - Name eines Operators mit der Eigenschaft BENUTZUNGSDEF.
Ergebnis: In die Assoziationsliste GEDFUNKS wird ein Paar mit der Definition des betreffenden Operators durch andere Operatoren auf Gedächtnisebene aufgenommen.

```
<BENDEF
  <LAMBDA (OP)
    (NCONC1 GEDFUNKS
            (CONS OP (CAR (GETP OP
                                (QUOTE BENUTZUNGSDEF>>      ■
```


<GED-ZWISCHEN <NLAMBDA (OP-AX AL-AX) ... >>
"GEDächtnis-ZWISCHEN-Funktion"

Aufruf: Von den in der Liste "gedfunks" befindlichen Definitionen der Operatoren auf Gedächtnisebene.
Params.: OP-AX - Name eines Operators;
AL-AX - Name einer Assoziationsliste "adt-name" mit den Definitionen der Operatoren auf axiomatischer Ebene.
Ergebnis: Die Umgebung wird (nochmal) mittels EVALA um die Definition der Operatoren auf axiomatischer Ebene erweitert. Dadurch wird auch deren Definition auf Gedächtnisebene verdeckt. Dann wird der angegebene Operator mit D aufgerufen und allen Gedächtnissen als Einzelargumente.
Wert: wird vom Operatoraufruf übernommen.

```
<GED-ZWISCHEN
  <NLAMBDA (OP-AX AL-AX)
    (EVALA <QUOTE (APPLY (EVAL OP-AX)
                         (CONS URD (CONS (CDR S)
                                         (COND
                                          ((CDDR S)
                                            (CONS (CDDR S)
                                                  (CDR ARGS)))
                                          ((CDR ARGS>
           (EVAL AL-AX>>                                ■
```

<EINGEDFUNK <LAMBDA (OP) ... >>
"EINgliedern von FUNKdefinitionen auf GEDächtnisebene"

Aufruf: Für alle IMPLEMOPS von L-PHASE bei Gedächtnisstrukturen
Param.: OP - Name eines Operators, der die Eigenschaft FUNK besitzt.
Ergebnis: Die Assoziationsliste GEDFUNKS wir um ein Paar mit der Definition des betreffenden Operators auf Gedächtnisebene mithilfe der Funktion GED-ZWISCHEN erweitert.

```
<EINGEDFUNK
  <LAMBDA (OP)
    (NCONC1 GEDFUNKS (LIST OP (QUOTE LAMBDA)
                              (QUOTE ARGS)
                              (LIST (QUOTE GED-ZWISCHEN)
                                    OP ADT-NAME>>       ■
```

5.1.1.2.5. Allgemeine Hilfsfunktionen

<PROBIERREGEL <LAMBDA (NREGEL PRUEFLINGNAMEN ...) ... >>
"NREGEL - REGEL mit Namen"

Aufruf: Von ->-PHASE, =-PHASE und &-PHASE.
Params.: NREGEL - vollständige Regel mit Namen als 1.Element; PRUEFLINGNAMEN - vollständige Liste der Namen aller für die Phase vorgesehenen Prüflinge: F01 ... Fmm oder A01 ... Ann oder die Liste der Operatoren.
Ergebnis: Führt die Funktionalitäts-, Axiomen- bzw. Operatorenvorauswahl durch, d.h. schneidet mit PRUEFLINGNAMEN nacheinander alle unter der Eigenschaft VM des Regelnamens genannten Listen. In ->- und =-Phase wird die Regel auf die übriggebliebenen Prüflinge nacheinander mittels PROBIERPRUEFLING angewendet. In der &-Phase wird REGELPRUEF direkt aufgerufen und spaltet sich ggf. erst bei einer Voraussetzung für *OP1 auf die verschiedenen Operatoren auf.

```
<PROBIERREGEL
 <LAMBDA (NREGEL PRUEFLINGNAMEN PRUEFLINGE REGELNAME EL OBJ)
  (SETQ REGELNAME (CAR NREGEL))
  (SETQ PRUEFLINGNAMEN (VM-PRUEFLINGNAMEN PRUEFLINGNAMEN))
  (PRINTL "PROBIERREGEL:" REGELNAME "AUF PRUEFLINGE"
                          PRUEFLINGNAMEN)
  (SELECTQ PHASE
      ((-%> =)
        (MAPCAR PRUEFLINGNAMEN (FUNCTION PROBIERPRUEFLING)))
      (& (AND PRUEFLINGNAMEN
              (REGELPRUEF (CDR NREGEL)
                          (OR (CDR PRUEFLINGNAMEN)
                              (CAR PRUEFLINGNAMEN)
                          (OR (CDR PRUEFLINGNAMEN)
                              (CONS (QUOTE *OP1)
                                    (CAR PRUEFLINGNAMEN)))
                          URBINDUNGEN)))
      NIL>>                                                  ■
```

<VM-PRUEFLINGNAMEN <LAMBDA (PNAMEN ...) ... >>
"bestimmte PRUEFLINGNAMEN gemaß Vereinfachtem Muster"
"PrüflingNAMEN"

Aufruf: Am Anfang von PROBIERREGEL (in ->- =- und &-Phase).
Vorauss.: REGELNAME muß als Wert den Namen einer Regel haben. Dieser Name muß unter der Eigenschaft "VM" als Wert eine Liste von Namen von Listen von Prüflingnamen haben. Der gesamte Wert unter VM bildet das Vereinfachte Muster der Regel und bestimmt, auf welche Pruflinge die Regel probiert werden soll.
Param.: PNAMEN - Die Grundmenge der Prüflingnamen, aus der dann mittels des vereinfachten Musters ausgewählt werden soll. Je nach Phase wird eine Liste mit den

Funktionalitätsnamen, den Axiomennamen bzw. OPERATOREN übergeben.
Ergebnis: PNAMEN wird nacheinander mit den Prüflingnamen geschnitten, die in den unter der Eigenschaft VM des aktuellen Regelnamens angegebenen Listen stehen. Der Gesamtdurchschnitt wird als Wert übergeben.
Wert: Eine Liste der Namen der Prüflinge, auf die die aktuelle Regel gemäß ihrem vereinfachten Muster ausprobiert werden soll.

```
<VM-PRUEFLINGNAMEN
  <LAMBDA (PNAMEN VM)
    (SETQ PNAMEN (COPY PNAMEN))
    (SETQ VM (GETP REGELNAME (QUOTE VM)))
    <MAPC VM (FUNCTION (LAMBDA (X)
               (SETQ PNAMEN (INTERSECTION PNAMEN
                                          (GETP PHASE X>
    PNAMEN>>                                                 ■
```

<IMPLK <NLAMBDA (B1 B2) ... >>
"IMPLiKation"
"B1 - Bedingung 1"
"B2 - Bedingung 2"

Aufruf: Durch Patternmatcher.
Params.: Bedingungen in Form von S-Ausdrücken.
Wert: T, wenn gilt B1 = B2. Sonst NIL. (Mit Einschränkungen s.u.)

Arbeitsweise:
Seien A, B Bedingungen. A ungleich B. Es gibt folgende Fallunterscheidungen:

Fall 1: A => A. Wert: T.

Fall 2: (NOT A) => (NOT B). Wird neu aufgerufen als B => A.

Fall 3: A => B.
Es ergeben sich folgende Unterfälle:

Fall 3.1: B = (AND B1 B2). Wird neu aufgerufen als (AND A => B1 , A => B2).

Fall 3.2: B = (OR B1 B2). Wird neu aufgerufen als (OR A => B1 , A => B2).

Fall 3.3: B = (NOT B1). Wird aufgespalten in:

Fall 3.3.1: B1 = (NOT B2). Wird neu aufgerufen als A => B2.

Fall 3.3.2: B1 = (AND B11 B12). Wird neu aufgerufen als (OR A => (NOT B11) ,

A => (NOT B12)).

Fall 3.3.3: B1 = (OR B11 B12). Wird neu aufgerufen als (AND A => (NOT B11) , A => (NOT B12)).

Fall 4: A => B. Wird aufgespalten in:

Fall 4.1: A = (AND A1 A2). Wird neu aufgerufen als (OR A1 => B , A2 => B).

Fall 4.2: A = (OR A1 A2). Wird neu aufgerufen als (AND A1 => B , A2 => B).

Fall 4.3: A = (NOT A1). Wird unterteilt in:

Fall 4.3.1: A1 = (NOT A2). Wird neu aufgerufen als A2 => B.

Fall 4.3.2: A1 = (AND A11 A12). Wird neu aufgerufen als (AND (NOT A10) => B , (NOT A12) => B).

Fall 4.3.3: A1 = (OR A11 A12). Wird neu aufgerufen als (OR (NOT A11) => B , (NOT 12) => B).

Fall 5: Nun sind die komplizierten und zusammengesetzten Bedingungen beseitigt und nur noch drei Varianten übrig:

Fall 5.1: A => (NOT B); siehe dazu Funktion B1-NB2.

Fall 5.2: (NOT A) => B; siehe dazu Funktion NB1-B2.

Fall 5.3: A => B. Liefert T.(s.u. !)

Bem.: Die Korrektheit der Funktion "IMPLK" ist nur gewährleistet, wenn die elementaren Bedingungen im Axiomensystem (d.h. ohne Betrachtung von Verknüpfungen mit AND, OR und NOT) sich an folgende Form halten:

```
"bedingung" ::= "beliebiger s-ausdruck" |
                (EQ "nichtzahlwert" "nichtzahlwert")
                | ({EQ | LESSP | LESSEQP |
                    GREATERP | GREATEREQP}
                "zahlwert" {"zahlwert" | "zahl"})
"beliebiger s-ausdruck"
                kann beliebige Vergleichsfunktionen
                oder Relationen enthalten.
"nichtzahlwert" ist ein S-Ausdruck, der sich nicht
                zu einer Zahl berechnen läßt.
"zahlwert"      ist ein S-Ausdruck, der sich zu einer Zahl berechnen läßt.
```


"zahl" ist eine ganze Zahl.

Ferner gelten folgende semantischen Beschränkungen: Variablen innerhalb eines "nichtzahlwert" müssen immer in der gleichen Reihenfolge angegeben werden. (D.h. es ist unzulässig, in verschiedenen Alternativen eines Axioms anzugeben "Falls (EQ A1 A2)" und "Falls (EQ A2 A1)".) Dies lässt sich vermeiden, indem z.B. grundsätzlich die alphabetische Reihenfolge gewählt wird.
Wenn man Zahlenvergleichs-Bedingungen benutzt, so gibt es zwei Möglichkeiten:

1. Man kann die "zahl" oder den zweiten "zahlwert" in allen Alternativen des Axioms gleich lassen, und die gewünschten Fallunterscheidungen durch Benutzung der verschiedenen Vergleichsfunktionen ausdrücken.
2. Man kann die "zahl" ändern, darf dann aber nur die Vergleichsfunktion EQ benutzen.

Für die Reihenfolge der Argumente gilt das gleiche wie bei "nichtzahlwert".
Es ist also nicht erlaubt, in einem Axiom folgende Fallunterscheidungen zu treffen, z.B.: "Falls (LESSEQP A 1)", "Falls (EQ A 2)", "Falls (GREATEREQP A 3)". Stattdessen muß man scheiben: "Falls (LESSP A 2)", "Falls (EQ A 2)", "Falls (GREATERP A 2)". Allenfalls wäre auch noch erlaubt: "Falls (EQ A 1)" "falls (EQ A 2)", "Falls (EQ A 3)". Diese Beschränkungen erfolgen aus rein pragmatischem Grund. Andere als die erlaubten Fälle sind bislang nie zwingend aufgetreten.

Zu Fall 5: A => B sei T.
Dieser Fall ist genaugenommen nicht korrekt behandelt. Man muß sich jedoch hier fragen, um die Funktion nicht übermässig kompliziert zu machen, wozu IMPLK verwendet wird. Doch nur dazu, um festzustellen, ob bei einer COND-Konstruktion gewisse Bedingungen aufeinander folgen können. Bei vergleichbaren Bedingungen ist die Reihenfolge wichtig, daher die o.a. Fallunterscheidungen. Bei voneinander unabhängigen Bedingungen ist diese Reihenfolge jedoch unwesentlich: Es ist z.B. egal, ob die Funktion

```
(COND ((EQ D DO) ... )
      ((EQ A1 A2) ... ))
```

lautet, oder ob die beiden Bedingungen vertauscht sind. Deshalb wird (EQ D DO) => (NOT (EQ A1 A2)) zu T berechnet.

```
<IMPLK
  <NLAMBDA (B1 B2)
    (COND
      ((EQUAL B1 B2))
      ((AND (EQ (CAR B1)
                (QUOTE NOT))
            (EQ (CAR B2)
                (QUOTE NOT)))
        (APPLY* (QUOTE IMPLK)
                (CADR B2)
                (CADR B1)))
      ((SELECTQ (CAR B2)
                ((AND OR NOT)
                  T)
                NIL)
        (APPLY* (QUOTE B2-AUFLOESUNG)
                B1 B2))
      ((SELECTQ (CAR B1)
                ((AND OR NOT)
                  T)
                NIL)
        (APPLY* (QUOTE B1-AUFLOESUNG)
                B1 B2))
      (T>>                                          ■
```

<B2-AUFLOESUNG <NLAMBDA (B1 B2) ... >>

Aufruf: Von IMPLK.
Bem.: Behandelt den Fall 3.

```
<B2-AUFLOESUNG
  <NLAMBDA (B1 B2)
    (SELECTQ (CAR B2)
      <AND (AND (APPLY* (QUOTE IMPLK)
                        B1
                        (CADR B2))
                (APPLY* (QUOTE IMPLK)
                        B1
                        (CADDR B2>
      <OR (OR (APPLY* (QUOTE IMPLK)
                      B1
                      (CADR B2))
              (APPLY* (QUOTE IMPLK)
                      B1
                      (CADDR B2>
        (NOT (SELECTQ (CAADR B2)
                      <NOT (APPLY* (QUOTE IMPLK)
                                   B1
                                   (CAR (CDADR B2>
                      <AND (OR <APPLY* (QUOTE IMPLK)
                                       B1
                                       (LIST (QUOTE NOT)
                                             (CAR (CDADR B2>
                               (APPLY* (QUOTE IMPLK)
```



```
                                       B1
                                       (LIST (QUOTE NOT)
                                             (CADR (CDADR B2>
                      <OR (AND <APPLY* (QUOTE IMPLK)
                                       B1
                                       (LIST (QUOTE NOT)
                                             (CAR (CDADR B2>
                               (APPLY* (QUOTE IMPLK)
                                       B1
                                       (LIST (QUOTE NOT)
                                             (CADR (CDADR B2>
                          (APPLY* (QUOTE B1-NB2)
                                  B1 B2)))
        NIL>>                                       ■
```

<B1-AUFLOESUNG >NLAMBDA B1 B2) ... >>

Aufruf: Von IMPLK.
Bem.: Behandelt den Fall 4.

```
<B1-AUFLOESUNG
  <NLAMBDA (B1 B2)
    (SELECTQ (CAR B1)
      (AND (OR (APPLY* (QUOTE IMPLK)
                       (CADR B1)
                       B2)
               (APPLY* (QUOTE IMPLK)
                       (CADDR B1)
                       B2)))
      (OR (AND (APPLY* (QUOTE IMPLK)
                       (CADR B1)
                       B2)
               (APPLY* (QUOTE IMPLK)
                       (CADDR B1)
                       B2)))
      (NOT (SELECTQ (CAADR B1)
                    (NOT (APPLY* (QUOTE IMPLK)
                                 (CAR (CDADR B1))
                                 B2))
                    (AND (AND (APPLY* (QUOTE IMPLK)
                                      (LIST (QUOTE NOT)
                                            (CAR (CDADR B1)))
                                      B2)
                              (APPLY* (QUOTE IMPLK)
                                      (LIST (QUOTE NOT)
                                            (CADR (CDADR B1)))
                                      B2)))
                    (OR (OR (APPLY* (QUOTE IMPLK)
                                    (LIST (QUOTE NOT)
                                          (CAR (CDADR B1)))
                                    B2)
                            (APPLY* (QUOTE IMPLK)
                                    (LIST (QUOTE NOT)
                                          (CADR (CDADR B1)))
```

```
                              B2)))
                (APPLY* (QUOTE NB1-B2)
                        B1 B2)))
        NIL>>                                                    ■
```

<NB1-B2 <NLAMBDA (B1 B2) ... >>
"Not B1 => B2"

Aufruf: Durch IMPLK.
Wert: Liefert T, wenn B1 => B2, oder wenn B1 und B2 unabhängig sind.
Bem.: Diese Funktion behandelt nur den Fall, daß die Bedingung die Form (NOT B1) => B2 hat.

```
<NB1-B2
  <NLAMBDA (B1 B2 H1 H2)
    (SETQ H2 (CAR B2))
    (SETQ H1 (CAADR B1))
    (COND
      ((EQUAL (CDADR B1)
              (CDR B2))
       (SELECTQ H1
                (LESSP (SELECTQ H2
                                (GREATEREQP T)
                                NIL))
                (GREATERP (SELECTQ H2
                                   (LESSEQP T)
                                   NIL))
                (GREATEREQP (SELECTQ H2
                                     (LESSP T)
                                     NIL))
                (LESSEQP (SELECTQ H2
                                  (GREATERP T)
                                  NIL))
                NIL))
      (T>>                                                       ■
```

<B1-NB2 <NLAMBDA (B1 B2) ... >>
"B1 => Not B2"

Aufruf: Durch IMPLK.
Wert: Liefert T, wenn B1 => B2, oder wenn B1 und B2 unabhängig sind.
Bem.: Diese Funktion behandelt nur den Fall, daß die Bedingung die FORM B1 => (NOT B2) hat.


```
<B1-NB2
  <NLAMBDA (B1 B2 H1 H2)
    (COND
      ((SELECTQ (CAR B1)
                ((AND OR NOT)
                  T)
                NIL)
        (APPLY* (QUOTE B1-AUFLOESUNG)
                B1 B2))
      ((<AND (EQ (SETQ H1 (CAR B1))
                 (SETQ H2 (CAADR B2)))
             (EQ H1 (QUOTE EQ))
             (EQUAL (CADR B1)
                    (CAR (CDADR B2>
        <NUMBERP (SETQ H1 (CAR (LAST (CDR B1>
        <NUMBERP (SETQ H2 (CAR (LAST (CDADR B2>
        (NEQ H1 H2))
       ((EQUAL (CDR B1)
               (CDADR B2))
        (SELECTQ H1
                 (LESSP (SELECTQ H2
                                 ((EQ GREATEREQP GREATERP)
                                   T)
                                 NIL))
                 (LESSEQP (SELECTQ H2
                                   (GREATERP T)
                                   NIL))
                 (EQ (SELECTQ H2
                              ((LESSP GREATERP)
                                T)
                              NIL))
                 (GREATEREQP (SELECTQ H2
                                      (LESSP T)
                                      NIL))
                 (GREATERP (SELECTQ H2
                                    ((LESSP EQ LESSEQP)
                                      T)
                                    NIL))
                 NIL))
        (T>>                                                     ■
```

<OR-VEREINF <LAMBDA (A B) ... >>
"OR-VEREINFachung"

Aufruf: Durch Patternmatcher.
Params.: Zwei Bedingungen, mit zitierten Klammern, auf die ein logisches ODER angewendet werden soll.
Wert: Geoderte Bedingung mit zitierten Klammern, wobei evtl. vorhandene Redundanzen beseitigt sind.

```
<OR-VEREINF
  <LAMBDA (A B)
    (SETQ A (CAR (EKL A)))
    (SETQ B (CAR (EKL B)))
    (PROG (A1 A2 B1 B2)
          (OR (AND (EQ (CAR A)
                       (QUOTE AND))
                   (SETQ A1 (CADR A))
                   (SETQ A2 (CADDR A)))
              (SETQ A1 A))
          (OR (AND (EQ (CAR B)
                       (QUOTE AND))
                   (SETQ B1 (CADR B))
                   (SETQ B2 (CADDR B)))
              (SETQ B1 B))
          <AND (EQUAL A1 B1)
               (OR (NULL B2)
                   (NULL A2)
                   (EQUAL A2 (LIST (QUOTE NOT)
                                   B2))
                   (EQUAL B2 (LIST (QUOTE NOT)
                                   A2)))
               (RETURN (ZKL (CONS B1>
          <AND (EQUAL A2 B1)
               (OR (NULL B2)
                   (EQUAL A1 (LIST (QUOTE NOT)
                                   B2))
                   (EQUAL B2 (LIST (QUOTE NOT)
                                   A1)))
               (RETURN (ZKL (CONS B1>
          <AND (EQUAL A1 B2)
               (OR (NULL A2)
                   (EQUAL A2 (LIST (QUOTE NOT)
                                   B1))
                   (EQUAL B1 (LIST (QUOTE NOT)
                                   A2)))
               (RETURN (ZKL (CONS B2>
          <AND A2 B2 (EQUAL A2 B2)
               (OR (EQUAL A1 (LIST (QUOTE NOT)
                                   B1))
                   (EQUAL B1 (LIST (QUOTE NOT)
                                   A1)))
               (RETURN (ZKL (CONS B2>
          (RETURN (APPEND* (LIST (QUOTE %()
                                 (QUOTE OR))
                           (ZKL (CONS A))
                           (ZKL (CONS B))
                           (LIST (QUOTE %)>>
```


<ZKL <LAMBDA (L) ... >>

"Zitiere KLammer"
"L - Liste"

Aufruf: Durch OR-VEREINF.
Param.: Eine Liste mit beliebig vielen Teillisten.
Wert: Eine neue Liste, in der die Listenklammern der Teillisten durch zitierte (also atomare) Klammern ersetzt sind.

```
<ZKL
  <LAMBDA (L)
    (COND
      ((NULL L)
       NIL)
      <(LISTP (CAR L))
       (NCONC (CONS (QUOTE %()
                    (ZKL (CAR L)))
              (CONS (QUOTE %))
                    (ZKL (CDR L>
      (T (CONS (CAR L)
               (ZKL (CDR L>>
```

<SETZEIG <LAMBDA (L) ... >>

"SETZe EIGenschaft"
"L - Liste"

Param.: L muß eine bereinigte Eigenschaftsangabe sein, d.h. ohne Steuerfunktion und ohne Variablen. Das erste Atom von L wird als EIG-träger aufgefaßt, das zweite als EIG-name, die weiteren bilden zusammen den EIG-wert.
Ergebnis: Wenn keine Eigwerte angegeben werden, wird mittels PUT die Eigenschaft T gesetzt. Wenn Eigwerte angegeben sind, wird überprüft, ob in der Wertliste für die Eigenschaft schon ein Wert ist, der mit demselben Atom anfangt; falls ja wird dieser durch den neu angegebenen Wert ersetzt, falls nein wird der neu angegebene Wert mittels ADDPROP hinzugefügt.

```
<SETZEIG
  <LAMBDA (L)
    (PROG ((ATOM (CAR L))
           (EIG (CADR L))
           (WERT1 (CADDR L))
           (WERTE (CDDDR L))
           WERTL)
          (COND
           <WERT1 (PRINTL "ERGEBNIS    :      " ATOM EIG
                          (CONS WERT1 WERTE))
                  (COND
                   <(SETQ WERTL (ASSOC WERT1 (GETP ATOM EIG)))
                    (COND
                     ((EQ WERT1 (QUOTE FALLS))
```

```
                (SETQ ERG (SUCHEAUSDR (CDR WERTL)))
                (AND (CADR ERG)
                     (PMFEHLER "BEI SETZUNG" L "IN
                               ALTER BEDINGUNG" WERTL
                               "MEHR ALS 1
                                BEDINGUNGSAUSDRUCK
                                HINTER 'FALLS"))
                (RPLACD WERTL (OR-VEREINF (CAR ERG)
                                          WERTE))
                (PRINTL "GEODERTE BEDINGUNG ERGEBNIS:"
                        ATOM EIG WERTL))
           (T (RPLACD WERTL WERTE>
      (T (ADDPROP ATOM EIG (CONS WERT1 WERTE>
 (T (PRINTL "ERGEBNIS    :      " ATOM EIG)
    (PUT ATOM EIG (QUOTE T>>                              ■
```

<SETZEIG* <LAMBDA L ... >>

Aufruf: Z.B.: (SETZEIG* 'UP 'SETZT 'M)
Bem.: Wie SETZEIG, nur daß den Elementen des Parameters L hier die einzelnen Parameter aus L entsprechen.

```
<SETZEIG*
  <LAMBDA L
    (SETZEIG L>>                                          ■
```

<EINSORT <LAMBDA (AUSDR LISTENTRAEGER VOR) ... >>
"EINSORTieren"
"AUSDR - LISP-S-AUSDRuck"
"VOR - Vergleichsfunktion (optional)"

Aufruf: Von beliebig.
Vorauss.: Es muß eine Variable existieren mit einer Liste als Wert.
Params.: Der Name dieser Variablen muß Wert des aktuellen Parameters für LISTENTRAEGER sein;
für LISTENTRAEGER kann auch (QUOTE <LISTE>) angegeben werden, wenn die Variable <LISTE> eine Liste enthält;
AUSDR ist ein in die Liste einzusortierender Ausdruck;
VOR (optional) ist ein FUNARG-Ausdruck, z.B. (FUNCTION ALPHORDER), der eine Boolesche Funktion darstellt, die einer irreflexiven Relation entspricht. Wird VOR nicht angegeben, so gilt (FUNCTION ALPHORDER) standardmäßig.
Ergebnis: Der angegebene Ausdruck wird in die durch Listenname bezeichnete Liste an der durch VOR ??? bestimmten Stelle einsortiert.
Wert: NIL, wenn an der durch VOR ??? für das Einsortieren bestimmten Stelle noch nichts stand, andernfalls das durch AUSDR ersetzte Listenelement.

Bem.: Im Regelmanipulationspaket ist eine Variable AVOR definiert, die folgenden FUNARG-Ausdruck für VOR enthält:

```
(SETQQ AVOR
       (FUNARG (LAMBDA (PAAR1 PAAR2)
                       (ALPHORDER (CAR PAAR1)(CAR PAAR2)))
               NIL))                                      ■
```

wie er zum Einsortieren in Assoziationslisten benötigt wird.
Anwendung z.B.:
(EINSORT '(F . 1) 'AL AVOR) erweitert die Liste AL um das Paar (F . 1) an der für F durch ALPHORDER bestimmten Stelle.

```
<EINSORT
  <LAMBDA (AUSDR LISTENTRAEGER VOR)
    (PROG ((LISTE (EVAL LISTENTRAEGER))
           AUSDR2
           (DRINNEN))
          (OR VOR (SETQ VOR (FUNCTION ALPHORDER)))
      NEXTPOS
          (SETQ AUSDR2 (CAR LISTE))
          (COND
            ((OR (NULL LISTE)
                 (APPLY* VOR AUSDR AUSDR2))
              (APPLY* (COND
                        (DRINNEN (QUOTE RPLACD))
                        (T (QUOTE SET)))
                      LISTENTRAEGER
                      (CONS AUSDR LISTE)))
            ((APPLY* VOR AUSDR2 AUSDR)
              (SETQ LISTENTRAEGER LISTE)
              (SETQ LISTE (CDR LISTE))
              (SETQ DRINNEN T)
              (GO NEXTPOS))
            (T (RPLACA LISTE AUSDR)
               (RETURN AUSDR2>>                           ■
```

<REGELERFOLG <LAMBDA (WERT VAR) ... >>
"REGELERFOLGsmeldung"

Aufruf: Von REGELPRUEF nach erfolgreichem Abarbeiten aller Voraussetzungen, bzw. in I- und B-Phase von WOWEITER nach erfolgreicher Anwendung von REGELPRUEF.
Params.: Wenn Parameter angegeben sind, bedeutet dies, daß ein Zweig einer Aufspaltung von REGELPRUEF erfolgreich zuende geführt wurde;
VAR - Der Name der *-Variablen, für die aufgespalten wurde;
WERT - Der Wert für die *-Variable, bei dem REGELPRUEF erfolgreich war. (Identifikation des Zweiges)
Ergebnis: Ausgabe einer Erfolgsmeldung, evtl. mit Angabe über

den Zweig der Aufspaltung: Erweiterung der VERWENDET-Liste um den Regelnamen; Ausgabe des neuen FUNK-Wertes in I- und B-Phase.
Wert: T.

```
<REGELERFOLG
  <LAMBDA (WERT VAR)
    (COND
      (WERT (PRINTL "—— AUFSPALTUNG" WERT "FUER" VAR ":")
            (SPACES 6)))
    (PRIN1 "——REGEL    : ")
    (PRIN1 REGELNAME)
    (PRIN1 " ERFOLGREICH ANGEWANDT")
    (OR (MEMB REGELNAME (CAR VERWENDET))
        (TCONC VERWENDET REGELNAME))
    (COND
      (PRUEFLINGNAME (PRINTL " AUF PRUEFLING" PRUEFLINGNAME))
      ((TERPRI)))
    (SELECTQ PHASE
        <(I B)
          <PUTD (QUOTE NEUE-FUNK)
                (CAR (EKL (APPEND RESULT (CONS (QUOTE ***)
                                               IPRUEFL>
          (PRETTYPRINT (QUOTE (NEUE-FUNK>
        NIL)
    T>>                                                    ■
```

<u><PMFEHLER <LAMBDA L ... >></u>
"FEHLER beim Pattern Matchen"

Aufruf: Von verschiedenen Stellen beim Patternmatchen.
Param.: L - eine Liste von Formen.
Ergebnis: Es werden ausgedruckt: REGELNAME, PRUEFLINGNAME, BINDE, VEREINBARE und die Werte der angegebenen Formen.
Wert: T.
Bem.: Die Standardfunktion, mit der beim Pattern-matchen Fehlermeldungen ausgegeben werden.

```
<PMFEHLER
  <LAMBDA L
    (PRINTL "***PMFEHLER: REGEL =" REGELNAME
                          "PRUEFLING =" PRUEFLINGNAME)
    (PRINTL "BINDE =" BINDE
            "VEREINBARE =" VEREINBARE)
    <MAPC L (QUOTE (LAMBDA (X)
                     (PRIN1 X)
                     (SPACES 1>
    (TERPRI)
    T>>                                                    ■
```


<u><PRINTREGLIST <LAMBDA (REGELLISTE APZ TEXT) ... >></u>
"PRINT REGelLISTe"
"APZ - Anzahl Pro Zeile"

Aufruf: Von ADT-IMPLEM und REGELVORAUSWAHL.
Params.: REGELMENGE ist eine Liste der zu druckenden Regeln, i.d.R. die 'noch möglichen' bzw. die 'verwendeten' Regeln;
APZ muß eine natürliche Zahl sein und gibt an, wieviele Regeln maximal pro Zeile geschrieben werden sollen (i.A. auch von der Länge des TEXTes abhängig;
Text ist ein Textstring, der den zu druckenden Regeln vorangestellt wird.
Ergebnis: Schreibt die Regelliste in wohlstrukturierter Form mit vorangestelltem TEXT und maximal APZ Regeln pro Druckzeile.

```
<PRINTREGLIST
  <LAMBDA (REGELLISTE APZ TEXT REGELZAEHLER LEERZEICHEN)
  (* "SCHREIBT TEXT UND REGELLISTE BIS ZU APZ-ELEMENTEN
     PRO ZEILE")
    (SETQ REGELZAEHLER 0)
    (PRIN1 TEXT)
    (SPACES 2)
    <SETQ LEERZEICHEN
          (POSITION (COND
                      ((NOT (OPENP NIL (QUOTE OUTPUT)))
                       NIL)
                      (T (CAR (OPENP NIL (QUOTE OUTPUT>
    <MAPCAR REGELLISTE (FUNCTION (LAMBDA (REGELNAME)
                (COND
                  ((LESSP REGELZAEHLER APZ)
                    (PRIN1 REGELNAME)
                    (SPACES 1)
                    (SETQ REGELZAEHLER (ADD1 REGELZAEHLER)))
                  (T (TERPRI)
                     (SPACES LEERZEICHEN)
                     (PRIN1 REGELNAME)
                     (SPACES 1)
                     (SETQ REGELZAEHLER 1>
    (TERPRI)>>                                             ■
```

<u><WELOPS <LAMBDA (FORM OPLISTE) ... >></u>

Ergebnis: Betrachtet die Form FORM, welche nur EVAL-Typ-Funktionen enthalten darf und listet alle in ihr vorkommenden Operatoren (also jeweils das erste Atom nach einer sich öffnenden Klammer) in der Liste OPLISTE auf.
Wert: OPLISTE.

```
<WELOPS
  <LAMBDA (FORM OPLISTE)
    (COND
      ((NLISTP FORM)
        OPLISTE)
      ((ATOM (CAR FORM))
        (APPEND (LIST (CAR FORM))
                (FLAT (MAPCAR (CDR FORM)
                              (FUNCTION WELOPS)))
                OPLISTE))
      (T (PRINTL "AUSDRUCK" (CAR FORM)
                 "ALS FUNKTIONSBEZEICHNER NICHT ZULAESSIG,
                  NUR ATOME">>                                  ■
```

<FLAT <LAMBDA (L) ... >>

Aufruf: Von WELOPS.

```
<FLAT
  <LAMBDA (L)
    (COND
      ((NLISTP L)
        L)
      ((NULL (CAR L))
        (FLAT (CDR L)))
      <(ATOM (CAR L))
        (CONS (CAR L)
              (FLAT (CDR L>
      (T (APPEND (FLAT (CAR L))
                 (FLAT (CDR L>>                                 ■
```

<KOPUSH <LAMBDA (X) ... >>

Ergebnis: Verwendet PM-Parameter KLAKO. Kellert S in KLAKO ein, also als oberstes Element.

```
<KOPUSH
  <LAMBDA (X)
    (SETQ KLAKO (CONS X KLAKO>>                                 ■
```

<KOPOP <LAMBDA NIL ... >>

Ergebnis: Verwendet PM-Parameter KLAKO. Entfernt oberstes Element von KLAKO.

```
<KOPOP
  <LAMBDA NIL
    (SETQ KLAKO (CDR KLAKO>>                                    ■
```


<KOTOP <LAMNDA NIL ... >>

Ergebnis: Verwendet PM-Parameter KLAKO. Liest oberstes Kellerelement.

```
<KOTOP
  <LAMBDA NIL
    (CAR KLAKO>>                                                ■
```

<BAULISTE <LAMBDA (MI P) ... >>

"MI - Musteratom"
"P - Prüfling"

Aufruf: Von VERGLEICH beim Matchen von Mustern der Art "..." MI.

Params.: MI ist ein (schon evaluiertes) Atom eines Musters einschließlich der nicht bindungsfähigen Atome %(, %), -%>, = und FALLS;
P ist ein normaler Prüfling in Listendarstellung.

Ergebnis: Der Prüfling wird durchsucht nach dem 1. Auftreten von MI, welches nicht durch %(und %) eingeklammert ist (also nur auf Klammerungsebene 0).

Wert:
a) Existiert ein solches Auftreten, wird eine 2-elementige Liste übergeben, welche durch Teilen von P an der von MI gematchten Stelle entsteht; MI selbst ist in keiner der beiden Teillisten mehr enthalten. Der erste Teil matcht "..." bzw. die LISTE-Variable, der zweite Teil zeigt, was noch gematcht werden muß. BAULISTE, welches eigentlich "..." matchen soll, matcht MI also implizit mit.
b) Kommen noch vor diesem Auftreten die nicht bindungsfähigen Atome -%>, = und FALLS, oder
c) existiert kein solches Auftreten, wird NIL übergeben, da das Muster "..." MA nicht gematcht werden konnte.

```
<BAULISTE
 <LAMBDA (MI P)
  (PROG (L (N 0))
    LOP (COND
          ((NULL P)
            (RETURN NIL))
          <(AND (EQ N 0)
                (COND
                  <(EQ (CAR P)
                       MI)
                    (RETURN (LIST L (CDR P>
                  ((MEMB (CAR P)
                         (QUOTE (%) = -%> FALLS)))
                    (RETURN NIL>
          ((MEMB (CAR P)
                 (QUOTE (-%> = FALLS)))
           (PMFEHLER (QUOTE VORAUSSETZUNG)
```

```
                VORAUSS "FEHLER IM PRUEFLING: INMITTEN
                AUSDRUCK" (APPEND L P)
                "NICHTBINDUNGSFAEHIGES ATOM"
                (CAR P))
          (RETURN))
        (T (AND (EQ (CAR P)
                    (QUOTE %())
                (SETQ N (ADD1 N)))
           (AND (EQ (CAR P)
                    (QUOTE %)))
                (SETQ N (SUB1 N)))
           <SETQ L (APPEND L (CONS (CAR P>
           (SETQ P (CDR P))
           (GO LOP>>                                    ■
```

<SUCHEAUSDR <LAMBDA (P) ... >>
"SUCHE AUSDRuck"
"P - Prüfling"

Aufruf: Von VERGLEICH, wenn eine als AUSDRUCK vereinbarte Variable zu matchen ist.
Param.: P ist ein Prüfling in Listendarstellung.
Wert: a) Wenn P mit dem Atom Klammer-auf beginnt, wird eine 2-elementige Liste geliefert, deren 1.Teilliste dem Anfang von P bis einschließlich der zugehörigen Klammer-zu entspricht und deren 2.Teilliste dem Rest von P entspricht. Der 1.Teil liefert die Bindung für die AUSDRUCK-Variable, der 2.Teil den Restprüfling.
b) Beginnt P mit einem der nicht bindbaren Atome -%>, =, %) oder "FALLS", so wird NIL geliefert.
c) Beginnt P mit einem bindbaren Atom, wird daraus eine Liste gemacht und wie unter a) beschrieben, als 1.Teilliste übergeben, (CDR P) als 2.Teilliste.

```
<SUCHEAUSDR
 <LAMBDA (P)
  (PROG (AUSD (N 0))
        <COND
          ((MEMB (CAR P)
                 (QUOTE (%) -%> = FALLS)))
           (PMFEHLER (QUOTE VORAUSSETZUNG)
                     VORAUSS
                     "STATT AUSDRUCK NICHTBINDUNGFAEHIGES
                     ATOM" (CAR P))
           (RETURN NIL))
          <(NEQ (CAR P)
                (QUOTE %())
           (RETURN (LIST (LIST (CAR P))
                         (CDR P>
          (T (SETQ AUSD (LIST (CAR P)))
             (SETQ P (CDR P>
    LOP (COND
```



```
          <((AND (EQ (CAR P)
                     (QUOTE %)))
                 (EQ N 0))
            (NCONC1 AUSD (QUOTE %)))
            (RETURN (LIST AUSD (CDR P>
          (T (SELECTQ (CAR P)
                      (%) (SETQ N (SUB1 N)))
                      (%( (SETQ N (ADD1 N)))
                      ((-%> = FALLS)
                        (PMFEHLER (QUOTE VORAUSSETZUNG)
                                  VORAUSS
                                  "INMITTEN AUSDRUCK"
                                  (APPEND AUSD P)
                                  "NICHTBINDUNGSFAEHIGES ATOM"
                                  (CAR P)))
                      NIL)
             (NCONC1 AUSD (COPY (CAR P)))
             (OR (SETQ P (CDR P))
                 (RETURN NIL (PMFEHLER (QUOTE VORAUSSETZUNG)
                                       VORAUSS
                                       "PRUEFLING ENDET, BEVOR
                                       AUSDRUCK" AUSD "BIS ZUR
                                       PASSENDEN ) AUFGEBAUT")
                   ))
             (GO LOP>>                                  ■
```

<REGELPRUEF <LAMBDA (REGEL PRUEFL VORBINDUNG
BINDE VEREINBARE ...) ... >>
"PRUEFL - PRUEFLing"
"BINDE - BINDEliste"

Aufruf: Von den Kontrollprogrammen für die einzelnen Phasen, d.h. von PROBIERPRUEFLING für die ->- und =-Phase, von PROBIERREGEL in der &-PHASE und von UEBERS in der I- und B-Phase.
Vorauss.: Die Variablen REGELNAME und PRUEFLINGNAME müssen aktuell definiert sein.
Params.: REGEL - Regel in interner Darstellung ohne Namen;
PRUEFL - Prüfling (sofern vorhanden) in Listendarstellung ohne Namen;
VORBINDUNG - Ein CONS-Paar, in dem eine aufgrund der Vorauswahl schon festliegende Bindung steht, zumeist für *OP1;
BINDE - Liste der vorgegebenen Bindungen, und zwar URBINDUNGEN bei normalem Aufruf, BINDE selbst bei rekursivem Aufruf (Aufspaltung);
VEREINBARE - Liste der vorgegebenen Vereinbarungen, wird nur bei rekursivem Aufruf für Aufspaltung angegeben.
Ergebnis: Die Voraussetzungen der Regel werden nacheinander mittels VORAUSSTEST überprüft. Bei erfolgreichem Matchen entstehende Bindungen werden zu BINDE hinzugefügt. Wenn VORAUSSTEST NIL liefert, wird abgebrochen und ebenfalls NIL geliefert. Wenn VORAUS-

STEST T liefert, wird fortgefahren. Wenn VORAUSSTEST allerdings eine Liste liefert, bedeutet dies, daß die Voraussetzung nicht eindeutig war und für das erste Element als Variable alle Werte aus dem Rest der Liste möglich sind (Aufzählvorauss.). In diesem Fall spaltet sich REGELPRUEF auf, d. h. ruft sich nacheinander mit den verschiedenen möglichen Bindungen als VORBINDUNG und dem Rest der Regel (einschließlich der Aufzählvoraussetzung) als REGEL auf.

Eigenschaften:
Wenn die Regel bis zum Folgerungspfeil "=>" erfolgreich abgearbeitet ist, werden die in STREICHE festgehaltenen, in der Regel durch "(DEL ...)" bezeichneten Teile gestrichen und die Folgerungen, die in der ->-, =- und &-Phase allesamt Eigenschaftssetzungen sind, mittels FOLGERE durchgeführt. Bei I-Regeln wird die Änderung des FUNK-Wertes durch die Funktion SUCHE vorgenommen.

Variablen:
Bei Folgerung der Pseudo-Eigenschaft BEZEICHNET wird die Liste URBINDUNGEN erweitert (siehe FOLGERE).

Wert: T, wenn die Regel erfolgreich angewandt wurde (Im Falle der Aufspaltung, wenn mindestens ein Zweig erfolgreich abgeschlossen wurde). NIL, wenn ein Voraussetzungstest NIL lieferte und die Regel daher abgebrochen werden mußte. Die Funktion liefert ebenfalls T, wenn alle Voraussetzungen erfolgreich abgearbeitet wurden und keine Folgerungen existieren. Dies ist der Fall, wenn in der Funktion EXISTIERT ein Operator gesucht wird, der mehrere Voraussetzungen erfüllen soll und dazu REGELPRUEF mit einer Liste von Voraussetzungen aufgerufen wird.

```
<REGELPRUEF
 <LAMBDA (REGEL PRUEFL VORBINDUNG BINDE VEREINBARE ZERSTOERT
                KLAKO ERG)
  (SETQ BINDE (CONS VORBINDUNG BINDE))
  (PROG NIL
   NEUVORAUSS
    (COND
     ((NULL REGEL)
       (RETURN T))
     ((NULL (CAR REGEL))
       (*COM* "LEERE VORAUSS. IN REGEL" REGELNAME)
       (SETQ REGEL (CDR REGEL))
       (GO NEUVORAUSS))
     <(EQ (CAR REGEL)
          (QUOTE =%>))
       (MAPC ZERSTOERT (FUNCTION STREICHE))
       (COND
        ((NULL (KOTOP))
          (REGELERFOLG EL OBJ)
          (MAPC (CDR REGEL)
                (FUNCTION FOLGERE))
          (RETURN T))
        (T (*COM* "KLAKO =" KLAKO)
```



```
           (*COM* "BEREITS DURCH SUCHE ERL.")
           (RETURN T>
     (<NULL (SETQ ERG (VORAUSSTEST (CAR REGEL>
       (*COM* (QUOTE REGEL)
              REGELNAME "NICHT ANWENDBAR, ABBRUCH"))
     ((EQ ERG T)
       (SETQ REGEL (CDR REGEL))
       (GO NEUVORAUSS))
     (T (*COM* (CDR ERG)
               "MOEGLICH FUER"
               (CAR ERG)
               ". ALLES PROBIEREN - PM")
        (RETURN <MEMB T
                    (MAPCAR (CDR ERG)
                     (FUNCTION (LAMBDA (EL OBJ PRUEFLINGNAME)
                     (SETQ OBJ (CAR ERG))
                     <SETQ PRUEFLINGNAME (COND
                          ((AND (EQ PHASE (QUOTE &))
                                (EQ OBJ (QUOTE *OP1)))
                           EL)
                          (T (STKEVAL
                                (STKPOS
                                  (QUOTE PROBIERPRUEFLING))
                                (QUOTE PRUEFLINGNAME>
                             (REGELPRUEF REGEL PRUEFL
                                         (CONS OBJ EL)
                                         BINDE VEREINBARE
                                         ZERSTOERT>>>
```

<VORAUSSTEST <LAMBDA (VORAUSS ...) ... >>
"VORAUSSetzungsTEST"

Aufruf: Von REGELPRUEF für jede Voraussetzung der Regel.

Param.: VORAUSS - Eine Voraussetzung in interner Darstellung, und zwar ein Funktionalitäts-, Axiomen- oder Eigenschaftsmuster, eine Vereinbarung, ein Suchmuster oder ein LISP-Funktionsaufruf.
Alle weiteren Parameter dienen lediglich dazu, Variable, die in verschachtelten Funktionen global benutzt werden, nicht nach aussen wirken zu lassen.

Ergebnis: Gemäss (CAR VORAUSS) wird eine der folgenden Behandlungsarten ausgewählt:

F oder A: Einfaches Matchen eines (Funktionalitaets- oder Axiomen-) Musters mittels WERTVGL.

E: Matchen eines Eigenschaftsmusters mittels EIGVORAUSS.

V: Durchfuehren einer Vereinbarung, d.h., entweder Eintragung der Vereinbarung in die Liste VEREINBARE, oder Gewährleileistung eines Indexes als Negation eines anderen Indexes mittels NICHTIND.

SUCHE: Sucht in dem FUNK-Wert von IPRUEFL nach dem Muster aus VORAUSS mittels SUCHE und

bewirkt die Abarbeitung des Restes der Regel (Ersetzung).

CALL: Dient zum Aufruf von LISP-Funktionen, durch welche bestimmte Voraussetzungsarten realisiert sind (UNGLEICH, GLEICH, NOT, EXISTIERT). Organisiert die Verarbeitung der solche LISP-Funktionsaufrufe umgebenden äussersten "eckigen Klammer" mitsamt ihres Indexes. Die Verarbeitung der inneren eckigen Klammern, die nur schon gebundene Indizes enthalten dürfen, geschieht automatisch durch die Funktion ?. Fuer die Berechnung der angegebenen LISP-Form werden mittels VARBIND *-Variable durch ihre Werte ersetzt und atomare Klammern in die entsprechenden Listenstrukturen mittels EKL überführt.

CALLMATCH: wie bei CALL, nur werden atomare Klammern atomar belassen. Wird zur Zeit nicht benötigt, kann aber verwendet werden, wenn LISP-Funktionen aufgerufen werden sollen, die Matchvorgänge durchführen.

Variablen:

Wenn eine Voraussetzung erfolgreich abgearbeitet wurde, wird eventuell BINDE oder VEREINBARE erweitert.

Wert: T, wenn die Voraussetzung zutraf, NIL, wenn die Voraussetzung nicht zutraf, eine Liste, wenn die (Eigenschafts)voraussetzung wegen einer noch ungebundenen Variable nicht überprüft werden konnte. (s.a. EIGVORAUSS)

```
<VORAUSSTEST
 <LAMBDA (VORAUSS FUERP ERG I? OP ENAME VWERT SUCHER FP RPL
          HILFE B V FUERM FBINDE FVEREINB FORM FALLS)
  (KOPUSH (CAR VORAUSS))
  <SETQ ERG (SELECTQ (SQCDR VORAUSS)
                (A (WERTVGL VORAUSS PRUEFL BINDE VEREINBARE))
                (F (WERTVGL VORAUSS PRUEFL BINDE VEREINBARE))
                (E (SETQ ERG (EIGVORAUSS VORAUSS
                                 BINDE VEREINBARE))
                   (AND FBINDE (SETQ BINDE FBINDE)
                        (SETQ VEREINBARE FVEREINB))
                   ERG)
                (V (SELECTQ (CADR VORAUSS)
                     (NICHT (NICHTIND (CAR VORAUSS)
                                      (CADDR VORAUSS)))
                     (PROGN (SETQ VEREINBARE
                                  (CONS
                                    (CONS (CAR VORAUSS)
                                          (CADR VORAUSS))
                                    VEREINBARE))
                            T)))
                (SUCHE (KOPUSH (QUOTE SUCHE))
                       (SUCHE <APPEND VORAUSS (QUOTE ((USW>
                              IPRUEFL))
```



```
                <(CALL CALLMATCH)
                  (SETQ VORAUSS (CAR VORAUSS))
                  (COND
                    ((EQ (CAR VORAUSS)
                         (QUOTE ?))
                      (SETQ I? (CADR VORAUSS))
                      (*COM* (QUOTE ?)
                             I? "STEUERUNG IM CALL")
                      (SETQ FORM (CADDR VORAUSS))
                      (COND
                       ((NULL (SETQ ERG (ASSOC I? BINDE)))
                         (*COM* "1. AUFTRETEN VON IND" I?)
                         (*COM* (QUOTE EVALA)
                                FORM
                                (QUOTE IN)
                                BINDE)
                         (SETQ ERG
                               (EVAL
                                 (VARBIND FORM
                                          (EQ (KOTOP)
                                              (QUOTE CALL))
                                          T)))
                         (SETQ BINDE (CONS (CONS I? ERG)
                                           BINDE))
                         T)
                       ((CDR ERG)
                         (*COM* "HAT WERT" (CDR ERG))
                         (*COM* (QUOTE EVALA)
                                FORM
                                (QUOTE IN)
                                BINDE)
                         (EVAL (VARBIND FORM (EQ (KOTOP)
                                                 (QUOTE CALL))
                                        T)))
                       (T)))
                    (T (*COM* (QUOTE EVAL)
                              VORAUSS
                              (QUOTE IN)
                              BINDE)
                       (EVAL (VARBIND VORAUSS (EQ (KOTOP)
                                                  (QUOTE CALL))
                                      T>
                  (PMFEHLER "UNBEKANNTE VORAUSSETZUNGSART IN"
                            (CONS (KOTOP) VORAUSS>
  (KOPOP)
  ERG>>                                                    ■
```

<NICHTIND <LAMBDA (IND1 IND2 ...) ... >>
"Vereinbare IND1 sei NICHT IND2"

Aufruf: Von VORAUSSTEST bei Voraussetzungen der ART (V I1 NICHT I2).

Param.: Die beiden Indizes für eckige Klammern, welche gegeneinander negiert sein sollen.

Wert: Wenn IND1 noch nicht gebunden ist, wird IND1 an die Negation des mittels INDBIND fesgestellten Werts von

IND2 gebunden und T geliefert, andernfalls wird T oder NIL geliefert, je nachdem, ob die an IND1 und IND2 begundenen Werte verschieden oder gleich sind.

Bem.: Nach erfolgreicher Abarbeitung einer "I1 SEI NICHT I2"-Voraussetzung ist gewährleistet, daß beide Indizes an verschiedene Werte gebunden sind.

```
<NICHTIND
  <LAMBDA (IND1 IND2 PAAR)
    (COND
     <<SETQ PAAR (ASSOC IND1 BINDE))
       (EQ (CDR PAAR)
           (NOT (INDBIND IND2 "IN VEREINBARUNG" VORAUSS>
     ((SETQ BINDE (CONS (CONS IND1 (NOT (INDBIND IND2
                                                 "IN VEREINBARUNG"
                                                 VORRAUSS)))
                        BINDE))
      T>>                                                          ■
```

<INDBIND <LAMBDA (IND . L) ... >>
"BINDe KlammerINDex"

Aufruf: Verschiedentlich, u.a. von NICHTIND und ECKIG.
Param.: IND - eine Zahl, die als Index für eckige Klammern verwendet wird;
L - beliebige LISP-Formen.
Ergebnis: Es wird in BINDE nachgeprüft, ob ein Wert an IND gebunden ist. Wenn nicht, wird eine diesbezügliche Fehlermeldung mittels PMFEHLER ausgegeben, die durch L ergänzt wird.
Wert: T bzw. NIL, wenn diese an IND gebunden waren, NOBIND, wenn nichts an IND gebunden war.

```
<INDBIND
  <LAMBDA (IND . L)
    (PROG ((PAAR (ASSOC IND BINDE)))
          (RETURN (COND
                    (PAAR (CDR PAAR))
                    (T (PMFEHLER "INDEX" IND
                                 "NICHT GEBUNDEN" L)
                       (QUOTE NOBIND>>                           ■
```

<? <NLAMBDA (IND FORM ...) ... >>
"Realisierung schon gebundener Indizes"

Aufruf: Kann vorkommen, wenn in einer CALL-Voraussetzung in der zu berechnenden Form mehrfach geschachtelte eckige Klammern (realisiert durch mit ? beginnende Form) auftreten. Diese inneren eckigen Klammern müssen dann indiziert und der Index schon gebunden sein.

Param.: IND - eine in BINDE gebundene Zahl;
FORM - beliebige LISP-Form, einschließlich Aufrufen der Funktion ?.
Wert: T, wenn IND in BINDE an NIL, der Wert der Berechnung von FORM, wenn IND in BINDE an T, T sowie Ausgabe einer Fehlermeldung, wenn IND nicht gebunden ist.

```
<?
 <NLAMBDA (IND FORM PAAR)
   (SETQ PAAR (ASSOC IND BINDE))
    (COND
     (PAAR (COND
            ((CDR PAAR)
             (EVAL (VARBIND FORM (EQ (KOTOP)
                                     (QUOTE CALL))
                             T)))
            (T)))
     (T (PMFEHLER "IM CALL INDEX AB TIEFE 2" IND
                  "IN FORM" FORM "NOCH NICHT GEBUNDEN">>          ■
```

<EXISTIERT <NLAMBDA (OP . VORAUSSETZUNGEN) ... >>

Aufruf: Als CALL-Voraussetzung von VORAUSSTEST.
Params.: OP - eine *-Variable, die einen Operator bezeichnen soll und in VORAUSSETZUNGEN vorkommt;
VORAUSSETZUNGEN - eine Liste von Voraussetzungen mit derselben Syntax wie der Voraussetzungsteil einer Regel.
Ergebnis: Prueft mittels SOME, ob es einen Operator für OP gibt, sodaß die VORAUSSETZUNGEN alle erfüllt sind.
Wert: T, wenn es einen Operator für OP gibt, der VORAUSSETZUNGEN erfüllt, Nil sonst.

```
<EXISTIERT
  <NLAMBDA (OP . VORAUSSETZUNGEN)
    (*COM* "EXISTIERT-AUFSPALTUNG FUER OPERATOREN" OPERATOREN
                                  "IN REGEL" REGELNAME)
    (PROG1 (AND <SOME OPERATOREN (FUNCTION (LAMBDA OP1)
                        (*COM* "EXISTIERT-ZWEIG" OP1 "FUER" OP)
                        (REGELPRUEF VORAUSSETZUNGEN OP1
                                    (CONS OP OP1) BINDE
                                    VEREINBARE>
           T>>                                                    ■
```

<EIGVORAUSS <LAMBDA (VORAUSS) ... >>

Aufruf: Von VORAUSSTEST, falls Eigenschaft geprüft werden muß.

Ergebnis: Kann Voraussetzungen der Formen:
eigträger eigenschaft wert
(NICHTEIG eigträger eigenschaft wert)
(? i eigträger eigenschaft wert)
eigträger (DEL eigenschaft wert) und
eigträger (ODER eigenschaft) wert
behandeln, wobei der wert jeweils auch leer sein kann; außerdem kann er wieder Steuerfunktionen enthalten. Der eigträger ist entweder eine Konstante oder eine bereits gebundene Variable. Ist der eigträger jedoch eine nichtgebundene Variable, so soll die Eigenschaft schon in der ;-Phase eingeführt worden sein: MENGE, GEDAECHTNIS, oder OP. Dann wird mittels AUFZAEHLVORAUSSLISTE festgestellt, welche Operatoren bzw. Mengen alle für Eigträger passen. Sind es mehr als 1, liefert EIGVORAUSS eine Liste, deren CAR der Name der Eigträgervariable ist und in deren CDR die möglichen Werte für Eigträger aufgeführt werden. Eine Eigenschaft ist entweder konstant, gebunden oder als Alternative vereinbart; dabei müssen die Alternativen sich gegenseitig ausschließen.

Wert: T, NIL oder die Liste aus Variablenname und den möglichen Bindungen. Wird übergeben an VORAUSSTEST und weiter an REGELPRUEF.

```
<EIGVORAUSS
 <LAMBDA (VORAUSS)
  (COND
   <<EQ (KOTOP)
        (QUOTE E))
     (*COM* "ES MUSS EIGTRAEGER GEFUNDEN WERDEN")
     (COND
       <<LISTP (CAR VORAUSS))
         (COND
           <<EQ (CAAR VORAUSS)
                (QUOTE NICHTEIG))
            (NULL (EIGVORAUSS (CDAR VORAUSS>
           ((EQ (CAAR VORAUSS)
                (QUOTE ?))
            (SETQ FBINDE BINDE)
            (SETQ FVEREINB VEREINBARE)
            (COND
              ((NUMBERP (SETQ I? (CADAR VORAUSS)))
               (COND
                 ((NULL (ASSOC I? BINDE))
                   (SETQ ERG (EIGVORAUSS (CDDAR VORAUSS)))
                   (COND
                     (ERG (SETQ BINDE (CONS (CONS I? T)
                                            BINDE))
                          (SETQ FBINDE NIL)
                          (SETQ FVEREINB NIL))
```



```
                     (T (SETQ BINDE (CONS (CONS I?)
                                          FBINDE))
                        (SETQ FBINDE NIL)))
                   T)
                  ((CDR (ASSOC I? BINDE))
                   (SETQ FBINDE)
                   (SETQ FVEREINB)
                   (EIGVORAUSS (CDDAR VORAUSS)))
                  (T)))
              (T (COND
                   ((EIGVORAUSS (CDAR VORAUSS))
                    (SETQ FBINDE NIL)
                    (SETQ FVEREINB NIL)))
                 T)))
         (T (PMFEHLER "VERBOTENE STEUERUNG FUER EIGTRAEGER:"
                      VORAUSS>
       (T (*COM* "LIEGT KEINE STEUERUNG VOR")
          <SETQ OP (COND
              ((VARIABLEP (CAR VORAUSS))
                (CDR (ASSOC (CAR VORAUSS)
                            BINDE)))
              (T (CAR VORAUSS>
            (COND
              (OP (KOPUSH (QUOTE OP))
                  (EIGVORAUSS (CDR VORAUSS)))
              ((NULL (SETQ AVORAUSS
                           (AUFZAEHLVORAUSS-LISTE VORAUSS)))
                NIL)
              ((NULL (CDR AVORAUSS))
                (KOPUSH (QUOTE OP))
                (SETQ OP (CAR AVORAUSS))
                (SETQ BINDE (CONS (CONS (CAR VORAUSS)
                                        (CAR AVORAUSS))
                                  BINDE))
                (EIGVORAUSS (CDR VORAUSS)))
              (T (CONS (CAR VORAUSS)
                       AVORAUSS>
  ((EQ (KOTOP)
       (QUOTE OP))
    (COND
      <<LISTP (CAR VORAUSS))
        (COND
          <<EQ (CAAR VORAUSS)
               (QUOTE ODER))
           (KOPOP)
           (PROG ((LISTE (CDAR VORAUSS)))
             LOP (SETQ ENAME (SQCDR LISTE))
                 (COND
                   ((NULL ENAME)
                     (RETURN ENAME))
                   <(GETP OP ENAME)
                     (RETURN (SUCHWERT (CDR VORAUSS)
                                       (GETP OP ENAME>
                   (T (GO LOP>
          ((EQ (CAAR VORAUSS)
               (QUOTE DEL))
```

```
      (AND (EIGVORAUSS (CDAR VORAUSS))
           (SETQ ZERSTOERT (CONS (CONS OP ENAME)
                                 ZERSTOERT))
           T))
     (T (PMFEHLER "VERBOTENE STEUERUNG FUER EIGNAME:"
                  (CAR VORAUSS>
(T (*COM* "DIE EIG MUSS ERKANNT WERDEN")
   (KOPOP)
   (COND
     ((NULL (VARIABLEP (CAR VORAUSS)))
       (SETQ ENAME (CAR VORAUSS))
       (*COM* "EIGNAME =" ENAME))
     <(ASSOC (CAR VORAUSS)
             BINDE)
       (SETQ ENAME (CDR (ASSOC (CAR VORAUSS)
                               BINDE>
     <(ASSOC (CAR VORAUSS)
             VEREINBARE)
       (SETQ VWERT (CDR (ASSOC (CAR VORAUSS)
                               VEREINBARE)))
       (COND
         <(EQ (CAR VWERT)
              (QUOTE ODER))
          (SETQ VWERT (CDR VWERT))
          (*COM* VWERT "FUER EIGNAME MOEGLICH")
          (PROG NIL
            LOP (SETQ ENAME (SQCDR VWERT))
                (COND
                  ((NULL ENAME)
                    (RETURN ENAME))
                  ((GETP OP ENAME)
                    (SETQ BINDE
                          (CONS (CONS (CAR VORAUSS)
                                      ENAME)
                                BINDE)))
                  (T (GO LOP>
         (T (*COM* "UNZULAESSIGE VEREINBARUNG" VWERT
                   (QUOTE FUER) (CAR VORAUSS>
     (T (*COM* "EIGENSCHAFT NICHT ERKENNBAR")
        NIL))
   (COND
     ((AND ENAME (SUCHWERT (CDR VORAUSS)
                           (GETP OP ENAME)))
       (SETQ VEREINBARE (REMOVE (ASSOC (CAR VORAUSS)
                                       VEREINBARE)
                                VEREINBARE))
       T)
     (T NIL>>                                          ■
```


<AUFZAEHLVORAUSS-LISTE (EIG ...) ... >>
"Erstelle LISTE möglicher Bindungen bei AUFZAEHLVORAUSSetzung"

Aufruf: Von EIGVORAUSS, wenn EIG-träger eine nicht gebunden Variable.
Param.: EIG ist eine Eigenschaftsvoraussetzung, deren EIG-träger eine nicht gebundene Variable ist.
Ergebnis: Wenn die Eigenschaft aus der ;-Phase stammt (MENGE, GEDACHTNIS, OP) ist dadurch bekannt, welche Grundmenge (MENGEN oder OPERATOREN) nach Elementen, auf die die Voraussetzung zutrifft, durchsucht werden soll. Ist der angegebene EIG-name nicht aus der ;-Phase, so wird OPERATOREN durchsucht, falls der EIG-träger *OP1 ist, andernfalls MENGEN.
Wert: Eine Liste, deren 1. Atom der Name der zu bindenden Variable ist und deren weitere Elemente (bei Eigenschaftsvorauss.: Atome) die dafür möglichen Werte sind. Wird von REGELPRUEF benötigt, um sich aufspalten zu können.

```
<AUFZAEHLVORAUSS-LISTE
 <LAMBDA (EIG PROBELS WERTL)
  <PROG (GRUNDMENGE)
        <SETQ GRUNDMENGE
              (SELECTQ (CADR EIG)
                  ((MENGE GEDAECHTNIS)
                    MENGEN)
                  (OP OPERATOREN)
                  (COND
                   ((EQ (CAR EIG)
                        (QUOTE *OP1))
                     (SETQ PROBELS (VM-PRUEFLINGNAMEN
                                    OPERATOREN))
                     (RETURN))
                   (T MENGEN>
        (MAPC GRUNDMENGE (FUNCTION (LAMBDA (EL)
                  (AND (SUCHWERT (CDDR EIG)
                                 (GETP EL (CADR EIG)))
                       (SETQ PROBELS (CONS EL PROBELS>
  (*COM* PROBELS "MOEGLICH FUER" (CAR EIG))
  PROBELS>>                                            ■
```

<SUCHWERT (M PLISTE) ... >>
"M - Muster"
"PLISTE - PrüflingLISTE"

Bem.: Wird von EIGVORAUSS aufgerufen mit dem Eigenschaftswertmuster (M), einer Liste von Atomen und der real existierenden Eigenschaftswertliste des gefundenen Operators unter der angegebenen Eigenschaft (PLISTE). Ist M leer, so ist der Wert irrelevant, PLISTE darf nur nicht NIL sein; ist M = (T), so wird geprüft, ob PLISTE = T. Bei dem Aufruf von EIGVORAUSS aus bedeutet dies, daß

(EIGVORAUSS '(OP EIG T)) nur zutrifft, wenn OP EIG, aber ohne besonderen Wert; ist M weder NIL noch T, so muß aus PLISTE ein geeigneter Wert herausgesucht werden. Die Suche erfolgt über das erste Nichtsteuerungsatom des Wertmusters M mittels ASSOC. Ist dies Atom "..." oder eine ungebundene Variable, wird nur der erste Wert in PLISTE zu matchen versucht. Die restlichen werden per Fehlermeldung ausgegeben. Wenn sich das Atom jedoch evaluieren läßt, findet SUCHWERT den ersten, damit beginnenden Wert (druckt -falls vorhanden- weitere mit einer Fehlermeldung aus) und versucht, ihn durch WERTVGL mit M zu matchen.

Wert: T bei Erfolg, NIL bei Mißerfolg.

```
<SUCHWERT
 <LAMBDA (M PLISTE)
  (COND
   ((NULL PLISTE)
     (*COM* "EIG NICHT VORHANDEN")
     NIL)
   ((NULL M)
     (*COM* "EIGWERT EGAL")
     T)
   ((EQ (CAR M)
        T)
     (EQ PLISTE T))
   (T (PROG (SUCHVAR SUCHER)
            (*COM* "ES WIRD EIN WERT VON MOEGLICHERWEISE
                    MEHREREN GESUCHT")
            <SETQ SUCHVAR (PROG ((VORN (CAR M)))
                                (*COM* "SUCHE NACH 1.ATOM")
                          LOP (COND
                                ((LISTP VORN)
                                  <SETQ VORN (COND
                                               ((AND (EQ (CAR VORN)
                                                         (QUOTE ?))
                                                     (NUMBERP
                                                       (CADR VORN)))
                                                (CADDR VORN))
                                               (T (CADR VORN>
                                  (GO LOP))
                                (T (RETURN VORN>
            (SETQ SUCHER (VARBIND SUCHVAR NIL T))
            <COND
              <(AND (EQ SUCHER SUCHVAR)
                    (VARIABLEP SUCHVAR))
                (SETQ FP (CAR PLISTE))
                (COND
                  ((CDR PLISTE)
                    (SETQ V (ASSOC SUCHVAR VEREINBARE))
                    (COND
                      <(EQ (CADR V)
                           (QUOTE ODER))
                       (RETURN
                         (WERTVGL
```



```
                    M
                    (ASSOC <CAR
                             (SOME (CDDR V)
                                   (FUNCTION
                                     (LAMBDA (MOEGL)
                                       (ASSOC MOEGL
                                              PLISTE>
                                   PLISTE>
                  (T (PRINTL "SUCHWERT - MUSTER =" M "IST
                             ASSOC-UNGENUEGEND WEGEN
                             VORNE VARIABLE" SUCHER)
                     (PRINTL "RESTLICHE WERTE" (CDR PLISTE)
                             "IGNORIERT">
                (T (SETQ RPL (ASSOCMEMB SUCHER PLISTE))
                   (SETQ FP (CAR RPL))
                   (AND (SETQ RPL (ASSOCMEMB SUCHER (CDR RPL)))
                        (*COM* "SUCHWERT - MUSTER =" M "IST
                               ASSOC-MEHRDEUTIG. RESTLICHE
                               PRUEFLINGE RPL
                               "WERDEN IGNORIERT">
                   (RETURN (WERTVGL M FP>>                    ■
```

<WERTVGL <LAMBDA (M P) ... >>
"WERTVerGLeich"
"M - Muster"
"P - Prüfling"

Bem: Vergleicht zwei Listen auf Übereinstimmung, wobei die eine (P) nur Atome enthalten darf, die andere zusätzlich einige Steuerfunktionen, nämlich:

- ? welche anzeigt, daß ein String nicht unbedingt gematcht werden muß. Ist das erste Zeichen des Strings eine Zahl, so dient diese als Index, welcher angibt, daß der String genau dann gematcht werden muß, wenn schon früher einer mit dieser Zahl gematcht werden konnte.
- ODER welche besagt, daß eins der folgenden Atome gematcht werden soll.
- DEL welche anzeigt, daß das Folgende bei erfolgreicher Abarbeitung der Regel zerstört werden muß (durch die Funktion STREICHE); WERTVGL berücksichtigt diese Steuerfunktion. Für den Vergleich von Atomen wird die Funktion VERGLEICH aufgerufen.

Wert: T bei Erfolg, NIL bei Mißerfolg.

```
<WERTVGL
 <LAMBDA (M P)
  (COND
   ((ATOM (CAR M))
     (VERGLEICH M P BINDE VEREINBARE FALLS))
   (T (SELECTQ (CAAR M)
         (? <COND
             <(NUMBERP (SETQ I? (CADAR M)))
               (*COM* "INDEX" I? "VORHANDEN")
               (COND
                ((NULL (ASSOC I? BINDE))
                  (*COM* "1.AUFTRETEN VON IND" I?)
                  (SETQ ERG
                        (VERGLEICH (APPEND (CDDAR M)
                                           (QUOTE (?)))
                                   P BINDE VEREINBARE FALLS))
                  <COND
                   (ERG (SETQ BINDE (CONS (CONS I? T)
                                          BINDE))
                        (SETQ P FUERP))
                   (T (SETQ BINDE (CONS (CONS I? NIL)
                                        BINDE>
                  (SETQ M (CDR M)))
                <(CDR (ASSOC I? BINDE))
                  (*COM* "MUSS GEMATCHT WERDEN")
                  (SETQ M (APPEND (CDDAR M)
                                  (CDR M>
                 (T (*COM* "NICHT MATCHEN")
                    (SETQ M (CDR M>
              (T (*COM* "KEIN INDEX VORHANDEN")
                 (SETQ ERG
                       (VERGLEICH (APPEND (CDAR M)
                                          (QUOTE (?)))
                                  P BINDE VEREINBARE FALLS))
                 (AND ERG (SETQ P FUERP))
                 (SETQ M (CDR M>
             (WERTVGL M P))
         (ODER (COND
                 ((MEMB (CAR P)
                        (CDAR M))
                   (WERTVGL (CDR M)
                            (CDR P)))
                 (T NIL)))
         (DEL (SETQ ERG (WERTVGL (CDAR M)
                                 P))
              (COND
                (ERG (COND
                       ((EQ (KOTOP)
                            (QUOTE E))
                         (SETQ HILFE (APPEND (LIST OP ENAME)
                                             P)))
                       (T (SETQ HILFE PRUEFLINGNAME)))
                     (SETQ ZERSTOERT (CONS HILFE ZERSTOERT))
                     (*COM* "ERWEITERN DER ZERSTOERT-LISTE
                            UM" HILFE)
                     T)
```



```
                (T NIL)))
         (USW (*COM* "PRUEFL DARF LAENGER SEIN")
              (AND (EQ (QUOTE REP)
                       (KOTOP))
                   (SETQ IPOI2 P))
              T)
         (REP <SETQ ERG (COND
                  <(EQ (LENGTH (CDAR M))
                       1)
                   (COND
                     ((VERGLEICH (CONS (CADAR M)
                                       (CDR M))
                                 P BINDE VEREINBARE FALLS)
                       (RPLACA P (VARBIND ERSATZ))
                       (SETQ IPRUEFL (EINEEBENE IPRUEFL>
                  ((VERGLEICH <APPEND (CDAR M)
                                      (QUOTE ((USW>
                              P BINDE VEREINBARE FALLS)
                   (COND
                     ((VERGLEICH (CDR M)
                                 IPOI2 BINDE
                                 VEREINBARE FALLS)
                       (RPLACA P (EINEEBENE
                                   (CONS(VARBIND ERSATZ)
                                        IPOI2)))
                       (RPLACD P NIL)
                       (SETQ IPRUEFL (EINEEBENE IPRUEFL)))
                     (T NIL>
         (KOPOP)
         (COND
           ((NULL ERG)
            NIL)
           (T)))
  (*COM* "UNBEKANNTE STEUERUNG">>
```

<u>\<VERGLEICH \<LAMBDA (M P BLIST VLIST VFALLS) ... >></u>

"M - Muster"
"P - Prüfling"
"BLIST - BindeLISTe"
"VLIST - VereinbarungsLISTe"
"VFALLS - Vergleich-Variable für FALLS"

Aufruf: Von WERTVGL aus, nach Abarbeitung der ersten Steuerung.

Params.: M - Muster in Listendarstellung;
P - Prüfling in Listendarstellung;
BLIST - lokale Bindeliste, wird von außen mit BINDE initialisiert, bei rekursivem Aufruf aber wieder mit BLIST, um bei letztendlich erfolglosen Matchversuchen nicht BINDE in REGELPRUEF zu verändern;
VLIST - lokale Vereinbarungsliste, wird von außen mit VEREINBARE initialisiert, bei rekursivem Aufruf aber wieder mit VLIST, s.o.;
VFALLS - wird von außen mit NIL initialisiert, bei

rekursivem Aufruf aber mit VFALLS, d.h. mit T, wenn das Atom FALLS schon gematcht wurde.

Ergebnis: Prüft Muster und Prüfling auf Übereinstimmung. Vergleicht die einzelnen Listenelemente von M und P; wird in M eine Steuerfunktion gefunden, so wird WERTVGL aufgerufen.
Ansonsten bedeutet Prüfen auf Übereinstimmung:
Ist das in M betrachtete erste Atom (im Folgenden kurz "ma" genannt)

- eine Konstante (nicht (VARIABLEP ma)), so muß das in P betrachtete erste Atom (kurz: pa) gleich ma sein.
- eine bereits gebundene Variable (in BLIST ex. Paar (ma . x)), so muß pa mit dem gebundenem Wert gleich sein (pa = x), sofern x ein Atom; ist x eine Liste (wenn ma als LISTE oder AUSDRUCK vereinbart ist), dann muß pa = (CAR x) sein.
- eine Variable, über die etwas vereinbart ist (in VLIST ex. Paar (ma . x)), so muß, wenn x
  - eine Liste beginnend mit dem Atom ODER ist, pa gleich einem der Listenelemente sein, ma wird an dieses Element gebunden,
  - das Atom LISTE ist, so können an ma 0 bis n Elemente von P ab pa gebunden werden, bis das auf ma folgende Atom gematcht wird.
  - das Atom AUSDRUCK, so muß an ma ein Ausdruck gebunden werden können; PA muß ungleich -%>, %), = oder FALLS sein.
- das Atom "..." ist, so können hieran 0 bis n Atome von P ab pa gebunden werden, bis das auf ma folgende Atom gematcht werden kann.
- eine Variable ist, über die weder eine Vereinbarung noch eine Bindung existiert, so wird pa an ma gebunden, falls pa ungleich -%>, %(, %), = oder FALLS ist. Nach Matchen von FALLS allerdings wird eine weder vereinbarte noch gebundene Variable implizit wie AUSDRUCK behandelt, um Bedingungsausdrücke zu matchen. Eine Behandlung als einfache Variable kann dann nur durch Vereinbarung als ATOM bewirkt werden.

Wert: T, wenn erfolgreich gematcht wurde, andernfalls NIL.

```
<VERGLEICH
 <LAMBDA (M P BLIST VLIST VFALLS)
  (COND
   ((OR (PROG1 (EQ (CAR M)
                   (QUOTE ?))
               (SETQ FUERP P))
        (AND (NULL M)
             (NULL P)))
    (SETQ BINDE BLIST)
    (SETQ VEREINBARE VLIST)
    (SETQ FALLS VFALLS)
    T)
   (<OR (PROGN (AND (EQ (CAR M)
                        (QUOTE FALLS))
```



```
                (SETQ VFALLS T)
                (*COM* "FALLS ANGETROFFEN, VARIABLEN AB
                       JETZT STANDARDMAESSIG ALS
                       AUSDRUCK"))
           (EQ (CAR M)
               (CAR P)))
       (AND <OR (VARIABLEP (CAR M))
                (MEMB (CAR M)
                      (QUOTE (... ---->
            (*COM* "ES LIEGT VARIABLE" (CAR M)
                   "VOR")
            (OR (AND <NULL (SETQ B (CDR (ASSOC (CAR M)
                                               BLIST>
                     (PROGN (SETQ V (CDR (ASSOC (CAR M)
                                                VLIST)))
                            (*COM* (CAR M)
                                   "VEREINBART ALS" V)
                            (COND
                              ((EQ V (QUOTE ATOM))
                               T)
                              (V NIL)
                              (VFALLS (SETQQ V AUSDRUCK)
                                      NIL)
                              (T)))
                     <NULL (MEMB (CAR P)
                                 (QUOTE (-%> %( %) = FALLS>
                     <NULL (MEMB (CAR M)
                                 (QUOTE (... ---->
                     (*COM* "EINFACHES BINDEN AN" (CAR P))
                     (SETQ BLIST (CONS (CONS (CAR M)
                                             (CAR P))
                                       BLIST)))
                (AND B (*COM* "SCHON GEBUNDEN AN" B)
                     <COND
                       <(LISTP B)
                        (SETQ M (APPEND B (CDR M>
                       (T (SETQ M (CONS B (CDR M>
                     (EQ (CAR M)
                         (CAR P)))
                (AND (LISTP V)
                     (*COM* "ES SIND ALTERNATIVEN" (CDR V)
                            "VEREINBART")
                     <NULL (MEMB (CAR P)
                                 (QUOTE (-%> %( %) = FALLS>
                     (MEMB (CAR P)
                           (CDR V))
                     (PROGN (SETQ BLIST (CONS (CONS (CAR M)
                                                    (CAR P))
                                              BLIST))
                            (SETQ VLIST (REMOVE (CONS (CAR M)
                                                      V)
                                                VLIST))
                            T))
                <AND <OR (EQ V (QUOTE LISTE))
                         (MEMB (CAR M)
                               (QUOTE (... ---->
```

```
                <SETQ FUERM (COND
                    ((VARIABLEP (CADR M))
                      (EVALA (CADR M)
                             BLIST))
                    ((NULL (CDR M))
                      <SETQ P (APPEND P (QUOTE (;>
                      <SETQ M (APPEND M (QUOTE (;>
                      (QUOTE ;))
                     (T (CADR M>
                (SETQ ERG (BAULISTE FUERM P))
                (*COM* "WIRD GEBUNDEN AN FOLG.
                       ELEMENTE:" (CAR ERG))
                (OR (MEMB (CAR M)
                          (QUOTE (... ---)))
                    (PROGN (SETQ BLIST
                                 (CONS (CONS (CAR M)
                                             (CAR ERG))
                                       BLIST))
                           (SETQ VLIST
                                 (REMOVE (CONS (CAR M)
                                               V)
                                         VLIST))
                           T))
                (SETQ M (CDR M))
                (SETQ P (CONS NIL (CADR ERG>
          (AND (EQ V (QUOTE AUSDRUCK))
               (SETQ ERG (SUCHEAUSDR P))
               (*COM* (QUOTE AUSDRUCK)
                      (CAR M)
                      "WIRD GEBUNDEN AN"
                      (CAR ERG))
               (PROGN (SETQ BLIST
                            (CONS (CONS (CAR M)
                                        (CAR ERG))
                                  BLIST))
                      (SETQ VLIST (REMOVE (CONS (CAR M)
                                                V)
                                          VLIST))
                      (SETQ P (CONS NIL (CADR ERG)))
                      T>
  (VERGLEICH (CDR M)
             (CDR P)
             BLIST VLIST VFALLS))
((LISTP (CAR M))
  (*COM* (CAAR M)
         "STEUERUNG")
  <COND
    ((EQ (CAAR M)
         (QUOTE REP))
      (KOPUSH (QUOTE REP>
  (SETQ BINDE BLIST)
  (SETQ VEREINBARE VLIST)
  (SETQ FALLS VFALLS)
  (WERTVGL M P))
(T NIL>>                                                    ■
```


<STREICHE <LAMBDA (S ...) ... >>

Param.: S bezeichnet, was für die weitere Regelanwendung gestrichen werden soll. Wird von REGELPRUEF verwendet, um durch DEL gekennzeichnetes zu löschen. Ist S ein Atom, muß es ein Axiomen- oder Funktionalitätsname sein. Ist S eine Liste, muß sie eine Eigenschaftsspezifikation sein.

Ergebnis: STREICHE führt die angegebenen Streichungen durch.
Ist S ein Atom, wird es aus der Wertliste unter der Eigenschaft PRUEFLINGNAMEN von -> bzw. = je nach dem Wert von PHASE entfernt und steht damit für die weitere Regelanwendung nicht mehr zur Verfügung;
ist S eine 2-elementige Liste, so wird die gesamte damit bezeichnete Eigenschaft gelöscht;
ist S eine mehr als 2-elementige Liste, so wird nur der damit bezeichnete Eigenschaftswert gelöscht.

Bem.: STREICHE wird für jeden Eintrag in ZERSTOERT von REGELPRUEF aufgerufen.

```
<STREICHE
  <LAMBDA (S PNAMEN)
    (COND
      <(ATOM S)
        (*COM* "ENTFERNE PRUEFLING" S)
        (PUT PHASE (QUOTE PRUEFLINGNAMEN)
             (REMOVE S (GETP PHASE (QUOTE PRUEFLINGNAMEN>
      ((NULL (CDDR S))
        (*COM* "ENTFERNE EIGENSCHAFT" S)
        (REMPROP (CAR S)
                 (CDR S)))
      (T (*COM* "ENTFERNE EIGENSCHAFTSWERT" S)
         (PUT (CAR S)
              (CADR S)
              (REMOVE (CDDR S)
                      (GETP (CAR S)
                            (CADR S>>                       ■
```

<FOLGERE <LAMBDA (FOLGERUNG ...) ... >>

Param.: FOLGERUNG - eine Liste, die eine Eigenschaftsspezifikation bzw. eine Pseudoeigenschaftsspezifikation darstellt.

Ergebnis: Wird von REGELPRUEF verwendet, um die Folgerungen der Regel auszuführen, die im Normalfall Eigenschaftssetzungen sind. Erzeugt gewünschte Eigenschaft für angegebenen Eigenschaftsträger mittels SETZEIG. Wenn es sich bei der angegebenen Eigenschaft um eine Pseudoeigenschaft handelt, z.Z. nur: BEZEICHNET, wird bei BEZEICHNET der Eigträger nicht mittels VARBIND berechnet, sondern ihm direkt die Eigenschaft beigegeben (nur zu Dokumentationszwecken).
Ausserdem wird URBINDUNGEN um die Bindung des

Eigträgers an den berechneten Eigwert erweitert.
Bsp.: *OP1 sei an PUSH gebunden. Dann bewirkt

```
(FOLGERE (QUOTE (*E BEZEICHNET *OP1)))
```

eine Erweiterung von URBINDUNGEN um das Paar (*E . PUSH). Bei Folgerung der Eigenschaft FUNK wird der Wert nach dem SETZEIG auf sein CAR gesetzt und damit der weiteren Abfrage durch Eigenschaftsmuster entzogen. Er kann dann nur noch durch SUCHE-Muster in I- und B-Regeln verändert werden.

```
<FOLGERE
 <LAMBDA (FOLGERUNG WERT EIGTRAEGER)
  <SETQ FOLGERUNG
        (EINEEBENE (MAPCAR FOLGERUNG (FUNCTION ECKIG>
  (COND
   ((SETQ EIGTRAEGER (CAR FOLGERUNG))
     <SETQ ADTNEUEIG (MEMB EIGTRAEGER (MAPCAR URBINDUNGEN
                                              (QUOTE CAR>
    (SELECTQ (VARBIND (CADR FOLGERUNG))
        (BEZEICHNET (SETQ WERT (VARBIND (CADDR FOLGERUNG)))
                    (SETZEIG* EIGTRAEGER (QUOTE BEZEICHNET)
                              WERT)
                    (SETQ BINDE (CONS (CONS EIGTRAEGER WERT)
                                      BINDE))
                    (SETQ URBINDUNGEN
                          (CONS (CONS EIGTRAEGER WERT)
                                URBINDUNGEN)))
        <FUNK (SETQ EIGTRAEGER (VARBIND EIGTRAEGER))
              (SETZEIG (VARBIND FOLGERUNG))
              (PUT EIGTRAEGER (QUOTE FUNK)
                   (CAR (GETP EIGTRAEGER (QUOTE FUNK>
        (SETZEIG (VARBIND FOLGERUNG>>
```

<ECKIG <LAMBDA (S) ... >>

Ergebnis: Verwendet BINDE. Ersetzt Elemente der Liste S, die mit indizierten eckigen Klammern, also mit ? beginnen, durch NIL, falls der Wert des Index NIl, durch die auf den Index folgenden Atome, falls der Wert gleich T ist. Wird von FOLGERE aufgerufen. Ist S keine Liste, wird S übergeben.

Bem.: In S konnen indizierte ?-Formen auch geschachtelt vorkommen, was einer logischen Konjunktion ihrer Indizes entspricht.


```
<ECKIG
  <LAMBDA (S)
    (COND
     <(LISTP S)
       (AND (INBIND (CADR S)
                    "IN" S "IN FOLGERUNG" FOLGERUNG)
            (EINEEBENE (MAPCAR (CDDR S)
                               (FUNCTION ECKIG>
     (T S>>
```

<VARBIND <LAMBDA (L SEXPR SELBSTPRUEF ...) ... >>
"VARiablenBINDung"
"L - Liste"
"SEXPR - S-EXPRession-flag"

Aufruf: Von VORAUSSTEST bei CALL mit der zu evaluierenden Form für L und SEXPR = T;
von FOLGERE mit einer Eigenschaftsspezifikation für L und SEXPR = NIL;
von SUCHBI mit SEXPR = T und SELBSTPRUEF = NIL.

Vorauss.: Globale Bindeliste BINDE muß existieren.

Params.: L - ein S-Ausdruck, in dem Variablen gemäß BINDE zu binden sind.
SEXPR - T oder NIL.
SELBSTPRUEF - T oder NIL.

Wert: In L werden alle Variablen durch ihren Wert gemäß BINDE ersetzt. Dabei wird der Wert für Variablen mit Klammertiefenindex durch Konkatenation des Wertes der zugehörigen Mengenvariable mit dem Tiefenindex gebildet. Wenn kein Wert existiert, wird die Variable nicht ersetzt und bei SELBSTPRUEF = NIL eine Fehlermeldung ausgegeben. Ist L ein Atom, wird das evaluierte Atom geliefert. Ist der Wert einer Variablen eine Liste, so wird die Variable nicht durch diese Liste sondern durch die Elemente der Liste ersetzt. Ist zusätzlich SEXPR = T, werden in der Liste vorkommende Ausdrucksklammern %(und %) mittels EKL in echte Listenklammern umgewandelt.

```
<VARBIND
 <LAMBDA (L SEXPR SELBSTPRUEF OHNEZWEI TIEFE)
  (COND
   ((ATOM L)
    (COND
     ((VARIABLEP L)
      (COND
       ((AND (SETQ B (ASSOC OHNEZWEI BINDE))
              (MEMB (CDR B)
                    MENGEN))
        (PACK (LIST (CDR B)
                    TIEFE)))
       ((SETQ B (ASSOC L BINDE))
        (CDR B))
       (T (APPLY* (COND
```

```
                    (SELBSTPRUEF (QUOTE *))
                    ((QUOTE PRINTL)))
                   "VARIABLE" L "UNGEBUNDEN BEI REGEL"
                   REGELNAME "UND PRUEFLING" PRUEFLING)
            L)))
      (T L)))
  (T (PROG ((L1 (CONS))
            ATOM WERT)
       NEUATOM
           (SETQ ATOM (CAR L))
           <COND
            <(ATOM ATOM)
              (SETQ WERT (VARBIND ATOM NIL SELBSTPRUEF))
              (COND
                ((NULL WERT))
                ((ATOM WERT)
                  (NCONC1 L1 WERT))
                <SEXPR (NCONC L1 (COPY (EKL WERT>
                (T (NCONC L1 (COPY WERT>
             (T (NCONC1 L1 (VARBIND ATOM SEXPR SELBSTPRUEF>
           (SETQ L (CDR L))
           (AND L (GO NEUATOM))
           (RETURN (CDR L1>>
```

■

<u>\<VARIABLEP \<LAMBDA (ATOM ...) ... >></u>
"VARIABLEnPrädikat"

Aufruf: Von VARBIND und von PM-Funktionen, die Bindungen herstellen.
Param.: ATOM ist ein Atom mit vorne maximal einem * und hinten maximal einer 2 als Klammertiefenindex.
Ergebnis: Es werden die beiden globalen Variablen gesetzt:
OHNEZWEI - das Atom, jedoch ohne seinen Tiefenindex 2, und
TIEFE - der Tiefenindex des Atoms in Stringdarstellung: "" für Ebene 1, "2" für Ebene 2.
Dies ermöglicht eine einfache Wiederherstellung des Variablennamens aus ATOM durch Konkatenation von OHNEZWEI und TIEFE.
Wert: T, wenn ATOM eine Variable ist (z.Z.: mit * beginnend), NIL andernfalls.

```
<VARIABLEP
  <LAMBDA (ATOM L)
    (SETQ L (UNPACK ATOM))
    (COND
     ((MEMB 2 L)
      <SETQ OHNEZWEI (PACK (REVERSE
                            (CDR (MEMB 2 (REVERSE L>
      (SETQ TIEFE "2")
     (T (SETQ OHNEZWEI ATOM)
        (SETQ TIEFE "")))
    (EQ (QUOTE *)
        (CAR L>>
```

■

<u>\<EINEEBENE \<LAMBDA (L) ... >></u>

Aufruf: Von FOLGERE.
Ergebnis: Sind in der Liste L Elemente enthalten, die wiederum Listen sind, so werden deren Elemente auf eine Ebene mit den Elementen von L gehoben, z.B.
(EINEEBENE (a b NIL c (d e) f (g (h i)))) =
(a b c d e f g (h i)).

```
<EINEEBENE
  <LAMBDA (L)
    (PROG <(L1 (QUOTE (START>
      LOP <COND
            ((NULL L)
              (RETURN (CDR L1)))
            ((NULL (CAR L)))
            <(LISTP (CAR L)
                    (SETQ L1 (APPEND L1 (CAR L>
            (T (SETQ L1 (APPEND L1 (LIST (CAR L>
          (SETQ L (CDR L))
          (GO LOP>>
```

■

<u>\<GLEICH \<NLAMBDA (X Y) ... >></u>

Aufruf: Von VORAUSSTEST bei CALL-Voraussetzung mit GLEICH-Aufruf.
Bem.: NLAMBDA-Version von EQUAL.

```
<GLEICH
  <NLAMBDA (X Y)
    (EQUAL X Y>>
```

■

<u>\<UNGLEICH \<NLAMBDA (X Y) ... >></u>

Aufruf: wie GLEICH.
Bem.: NLAMBDA-Version von NEQ.

```
<UNGLEICH
  <NLAMBDA (X Y)
    (NEQ X Y>>
```

■

<u>\<ASSOCMEMB \<LAMBDA (A AL) ... >></u>

Aufruf: Von Suchwert.
Param.: A - eine ASSOC-Suchkriterium (Vorderteil);
AL - eine ASSOC-Liste.
Wert: Wie MEMB, nur wird nach dem 1.Auftreten des Paars mit CAR = A in AL gesucht, statt nach dem 1. Auftreten von A.

```
<ASSOCMEMB
  <LAMBDA (A AL)
    (PROG NIL
      NEXT (AND (EQ A (CAAR AL))
                (RETURN AL))
           (SETQ AL (CDR AL))
           (OR AL (RETURN))
           (GO NEXT>>                                    ■
```

<EKL <LAMBDA (L) ... >>

"Ersetze KLammern"

Wert: Abgeänderte Liste L, in der alle zitierten Klammern %(und %) durch unzitierte Klammern ersetzt werden (also zu echten Listen umgewandelt werden und zwar auf allen Ebenen und beliebig oft).

Bem.: Realisiert durch Aufruf der Funktionen LSTATTZ UND LBISEZ.

```
<EKL
  <LAMBDA (L)
    (LSTATTZ L (QUOTE %()
               (QUOTE %))>>
```

<LSTATTZ <LAMBDA L A E K ...) ... >>

"Liste STATT Zeichen"
"L - Liste"
"A - Anfangszeichen"
"E - Endzeichen (kann auch NIL sein)"
"K - Kennzeichen (kann auch NIL sein"

Wert: Abgeänderte Liste L, wobei alle Auftreten von A in allen Klammerungstiefen von L zu einer offenen Klammer und alle Auftreten von E zu einer geschlossenen Klammer werden. Wird E nicht angegeben, so wird ein Endzeichen am Ende der aktuellen Liste angenommen. Wenn das Kennzeichen K angegeben ist, wird es jeweils an den Anfang der neu erzeugten Listen gesetzt.

Beispiel:

```
(LSTATTZ (QUOTE (A + B + C * * D (+ E F *)))
         (QUOTE +)(QUOTE *)(QUOTE ?))
         liefert
         (A (? B (? C)) D ((? E F)))
```



```
<LSTATTZ
  <LAMBDA (L A E K AKTUELL DANACH)
    (COND
      ((NULL (MEMB* A L))
        L)
      ((LISTP (SETQ AKTUELL (CAR L)))
        (CONS (LSTATTZ AKTUELL A E K)
              (LSTATTZ (CDR L)
                       A E K)))
      <(EQ AKTUELL A)
        (SETQ DANACH (LBISEZ (CDR L)))
        (COND
          (K (CONS (CONS K (LSTATTZ (CAR DANACH)
                                    A E K))
                   (LSTATTZ (CDR DANACH)
                            A E K)))
          (T (CONS (LSTATTZ (CAR DANACH)
                            A E K)
                   (LSTATTZ (CDR DANACH)
                            A E K>
      (T (CONS AKTUELL (LSTATTZ (CDR L)
                                A E K>>                  ■
```

<LBISEZ <LAMBDA (L KLZ) ... >>

"Liste BIS EndZeichen"
"L - Liste"
"KLZ - KLammerZahl"

Aufruf: Nur von der Funktion LSTATTZ.

Ergebnis: LBISEZ ist eine Hilfsfunktion für LSTATTZ und nur dort sinnvoll. Sie wird aufgerufen, wenn in der Liste ein Anfangszeichen entdeckt wurde und liefert eine Liste ab.

Wert: Liste. CAR auf diese Liste angewendet ergibt die Liste bis zum entsprechenden Endzeichen; CDR ergibt die Liste nach dem Endzeichen.

```
<LBISEZ
  <LAMBDA (L)
    (PROG ((KLZ 0)
           (L1 (CONS)))
      OBEN(COND
            <(NULL L)
              (RETURN (CONS (ENTCONC L1)
                            (CDR L>
            ((EQ (CAR L)
                 A)
              (SETQ KLZ (ADD1 KLZ)))
            <(EQ (CAR L)
                 E)
              (COND
                <(ZEROP KLZ)
                  (RETURN (CONS (ENTCONC L1)
                                (CDR L>
```

```
          (T (SETQ KLZ (SUB1 KLZ>
       (T NIL))
     (TCONC L1 (CAR L))
     (SETQ L (CDR L))
     (GO OBEN>>
```

■

<STRICHLISTE <LAMBDA (L) ... >>

Aufruf: An verschiedenen Stellen im Kontext:
(PACK (STRICHLISTE L))
Param.: L ist eine Liste von Atomen.
Wert Eine neue Liste, wobei ein "-" zwischen die Atome von L gesetzt wird.
Beispiel:
(STRICHLISTE (QUOTE (A B C D))) liefert
(A - B - C - D)

```
<STRICHLISTE
  <LAMBDA (L)
    (COND
      <(CDR L)
        (CONS (CAR L)
              (CONS (QUOTE -)
                    (STRICHLISTE (CDR L>
      (T L>>
```

■

5.1.1.3. Testfunktionen

<PF <NLAMBDA (RNR FNR ...) ... >>
"Probiere Funktionalität"

Aufruf: Vom Bildschirm aus zum Testen einer ->-Regel.
Param.: FNR - eine Funktionalitätsnummer (ohne F);
RNR - eine Regelnummer (ohne ->).
Ergebnis: FNR und RNR werden zu vollständigen Funktionalitäts- bzw. Regelnamen ergänzt. Danach werden die damit bezeichnete Regel und der Prüfling aufeinander angewandt.

```
<PF
  <NLAMBDA (RNR FNR PRUEFLINGNAME REGELNAME FUNK EL OBJ)
    (SETQ PHASE (QUOTE -%>))
    (SETQ PRUEFLINGNAMEN (GETP PHASE (QUOTE PRUEFLINGNAMEN)))
    (SETQ PRUEFLINGNAME (NAMBILD (QUOTE F)
                                 FNR))
    (SETQ REGELNAME (NAMBILD PHASE RNR))
    (SETQ FUNK (EVALA PRUEFLINGNAME FUNKS))
    (REGELPRUEF (EVALA REGELNAME -%>-REGELN)
                FUNK
```



```
                (CONS (QUOTE *OP1)
                      (CAR FUNK))
                URBINDUNGEN>>
```

■

<PA (RNR ANR ...) ... >>
"Probiere Axiom"

Bem.: Wie PF, nur gibt ANR eine Axiomennummer an (ohne A).

```
<PA
  <NLAMBDA (RNR ANR PRUEFLINGNAME REGELNAME AXIOM EL OBJ)
    (SETQ PHASE (QUOTE =))
    (SETQ PRUEFLINGNAMEN (GETP (QUOTE =)
                               (QUOTE PRUEFLINGNAMEN)))
    (SETQ PRUEFLINGNAME (NAMBILD (QUOTE A)
                                 ANR))
    (SETQ REGELNAME (NAMBILD PHASE RNR))
    (SETQ AXIOM (EVALA PRUEFLINGNAME AXIOME))
    (REGELPRUEF (EVALA REGELNAME =-REGELN)
                AXIOM
                (CONS (QUOTE *OP1)
                      (CADR AXIOM))
                URBINDUNGEN>>
```

■

<PU <NLAMBDA (RNR PRUEFLINGNAME ...) ... >>
"Probiere Und-Regel"

Bem.: Wie PF, nur kann für PRUEFLINGNAME ein Operatorname angegeben werden, der dann nicht verändert wird. Wenn ein PRUEFLINGNAE angegeben wird, wird die angegebene Regel mit der Vorbindung von *OP1 an diesem ausprobiert, andernfalls ohne Vorbindung. Sollte in der Regel eine Eigenschaftsvoraussetzung für *OP1 auftauchen, und keine Vorbindung für *OP1 existieren, so spaltet sich REGELPRUEF auf, um alle Operatoren für *OP1 auszuprobieren. Daher kann bei PF die Angabe für PRUEFLINGNAME immer weggelassen werden. Die Angabe eines bestimmten Prüflings beschleunigt aber das Austesten.

```
<PU
  <NLAMBDA (RNR PRUEFLINGNAME PRUEFLINGE REGELNAME EL OBJ)
   (SETQQ PHASE &)
   (SETQ REGELNAME (CAR NREGEL))
   (REGELPRUEF (EVALA REGELNAME &-REGELN)
               PRUEFLINGNAME
               (AND PRUEFLINGNAME (CONS (QUOTE *OP1)
                                        PRUEFLINGNAME))
               URBINDUNGEN>>
```

```
<PI (RNR PNA ...) ... >>
"Probiere I-Regel"
"RNR - RegelNummeR"
"PNA - PrüflingNAme"

Aufruf:   vom Bildschirm zum Testen einer bestimmten I-Regel
          (nämlich RNR)
Bem.:     Der Prüflingname darf nicht NIL sein.

<PI
  <NLAMBDA (RNR PNA REGELNAME PRUEFLINGNAME EL OBJ I-NAMEN)
    (SETQQ PHASE I)
    (SETQ REGELNAME (NAMBILD PHASE RNR)
    (SETQ I-NAMEN (LIST REGELNAME))
    (UEBERS PNA>>                                          ■


<PB <NLAMBDA (RNR PNA ... ) ... >>
"Probiere B-Regel"
"RNR - RegelNummeR"
"PNA - PrüflingNAme"

Aufruf:   vom Bildschirm zum Testen einer bestimmten B-Regel
          (RNR).
Bem.:     Der Prüflingname darf nicht NIL sein.

<PB
  <NLAMBDA (RNR PNA REGELNAME PRUEFLINGNAME EL OBJ)
    (SETQQ PHASE B)
    (SETQ REGELNAME (NAMBILD PHASE RNR))
    (SETQ B-NAMEN (LIST REGELNAME))
    (BEREINIGE PNA B-NAMEN B-REGELN B-ATOMARE-SCHLUESSEL
               B-NICHTATOMARE-SCHLUESSEL>>                 ■


<LETZTEPH <NLAMBDA (DATENTYPNAME) ... >>
"LETZTE PHase"

Vorauss.: Es muß bereits ein MAKEFILE des Datentyps gemacht
          worden sein.
Param.:   DATENTYPNAME gibt den Dateinamen an, in der der Da-
          tentyp steht.
          Schreibweise: IMPLEM.<Datentypname>.
Ergebnis: Die Definition des Datentyps wird zurückgesetzt auf
          den Stand des letzten MAKEFILE-Kommandos.
Bem.:     Zur Erleichterung des Testens der einzelnen Phasen.

<LETZTEPH
  <NLAMBDA (DATENTYPNAME)
    (REMPROPS (APPEND (QUOTE (D S DO))
                      (APPEND MENGEN OPERATOREN)))
    (LOAD DATENTYPNAME>>                                   ■
```


5.1.2. Regeln des ADTCOMP-Systems

Im folgenden sind die Regeln des ADTCOMP-Subsystems des APE's dokumentiert. Auf die Werte der zur Regelanwendung notwendigen Systemvariablen wird hier nicht eingegangen, da sie durch Anwendung der Funktion REGELÜBERFÜHRUNG automatisch initialisiert werden.

Da es sich im wesentlichen um ein Kopie aus [EKT 79] handelt, sei zur Erläuterung des Codes dorthin verwiesen, sowie zur Erläuterung der durch das eingeführte Parametrisierungskonzept notwendigen Modifikationen gegenüber [EKT 79] auf Abschnitt 3.1.1.

-%>-Regeln:

```
(-%>01 (F *OP1 D (? 1 *Y)
               ... -%> D)
       (CALL (? 1 (UNGLEICH *Y D)))
       =%>
       (*OP1 FORTSETZBAR EINFACH))

(-%>02 (F *OP1 D D (? 1 *Y) ... -%> D)
       (CALL (? 1 (UNGLEICH *Y D)))
       =%>
       (*OP1 FORTSETZBAR ZWEIFACH))

(-%>02A (F *OP1 D ... -%> X)
        =%>
        (*OP1 LIEST))

(-%>03 (E *A MENGE)
       (E (? 1 (NICHTEIG *A GEDAECHTNIS)))
       (F *OP1 D ... *A ... -%> (ODER D X))
       =%>
       (*OP1 GEMAESS *A)
       (*A AUSWAHLMENGE)
       ((? 1 D GEMAESS *A)))

(-%>04 (E *OP1 OP VORGEGEBEN)
       (F (DEL *OP1 X -%> *A))
       (E *A AUSWAHLMENGE)
       =%>
       (*A AUSWAHLMENGE SCHLUESSEL))

(-%>05 (E *N GEDAECHTNIS)
       (F *OP1 ... *N ... -%> ...)
       =%>
       (*OP1 GEDENKT *N))

(-%>06 (F *OP1 ... -%> *N)
       (E *N GEDAECHTNIS)
       =%>
       (*OP1 SETZT *N))
```

```
(-%>07 (F *OP1 ... -%> *B)
       (E *B MENGE BOOLZAHL)
       =%>
       (*OP1 BOOLESCHE FUNKTION))

(-%>07A (F *OP1 D ... X -%> D)
        =%>
        (*OP1 PARAMCHECK))

(-%>08 (E *OP1 OP IMPLEM)
       (E (NICHTEIG *OP1 FORTSETZBAR ZWEIFACH))
       (E (? 9 D PARAMETER))
       (E (? 0 *OP1 PARAMCHECK T))
       (V *LISTE LISTE)
       (E (? 1 *NA GEDAECHTNIS 1))
       (E (? 2 *NB GEDAECHTNIS 2))
       (V 3 NICHT 1)
       (V 4 NICHT 2)
       (E (? 7 *OP1 FORTSETZBAR))
       (E (? 8 *OP1 SETZT))
       (F *OP1 (? D)
          (? 5 *NA)
          (? 6 *NB)
          *LISTE -%> ...)
       =%>
       (*OP1 FUNK %( LAMBDA %( D (? 1 *NA)
                  (? 2 *NB)
                  (? 3 (? 5 *NA))
                  (? 4 (? 6 *NB))
                  *LISTE %) (? 0 (? 9 CHECK))
                  FORM
                  (? 1 (? 7 S))
                  (? 8 S)
                  (? 3 (? 7 D))
                  %))
        (*OP1 PARAMS (? 1 *NA)
                  (? 2 *NB)
                  (? 3 (? 5 *NA))
                  (? 4 (? 6 *NB))
                  *LISTE>
```

=-REGELN:

```
(=01 (E *OP1 LIEST T)
     (A (DEL %( *OP1 %( *OP2 ... X2 ... %) ... %)
          = X2 (? 1 FALLS *BED)))
     =%>
     (*OP1 LIEST (? 1 FALLS *BED))
     (*OP2 SCHREIBT ELEM)
     (*E BEZEICHNET *OP2))
(=03 (V *AUSDR AUSDRUCK)
     (A (DEL %( *OP1 *AUSDR ... %) = UNDEF (? 1 FALLS *BED)))
     (CALL (? 2 (GLEICH *AUSDR DO)))
     =%>
     (*OP1 UNDEFINIERT FALLS (? 1 (? 2 %( AND))
           (? 2 %( EQ D DO %))
```



```
           (? 1 *BED)
           (? 1 (? 2 %)>

(=04 (E *E FORTSETZBAR EINFACH)
     (A %( *OP1 %( *E D2 ... %) ... %) = (? 1 %( *E)
        ... %( *OP1 D2 ... %) (? 1 ... %))
        (? 2 FALLS *BED))
     (CALL (UNGLEICH *OP1 *E))
     =%>
     (*OP1 DURCHLAEUFT (? 2 FALLS *BED)))

(=05 (A %( *E %( *E D2 ... X2 %) ... X %)
       = %( *E %( *E D2 ... X %) ... X2 %) (? 1 FALLS *BED))
     =%>
     (*E DURCHLAEUFT (? 1 FALLS *BED)))

(=05A (E *E FORTSETZBAR ZWEIFACH)
      (A %( *OP1 %( *E D12 D22 ... %) ... %) = (? 1 %( *E)
         ... %( *OP1 D12 ... %) (? 1 D22 ... %))
         FALLS *BED)
      =%>
      (*OP1 DURCHLAEUFT P1 FALLS *BED))

(=05B (E *E FORTSETZBAR ZWEIFACH)
      (A %( *OP1 %( *E D12 D22 ... %) ... %) = (? 1 %( *E D12)
         ... %( *OP1 D22 ... %) (? 1 ... %))
         FALLS *BED)
      =%>
      (*OP1 DURCHLAEUFT P2 FALLS *BED))

(=06A (A (DEL %( *OP1 DO (? 1 A)
               X %) = %( *E DO (? DO)
               X %) (? 2 FALLS *BED)))
      (V 3 NICHT 1)
      (V 4 NICHT 2)
      =%>
      (*OP1 SCHREIBT (? 1 (? 4 FALLS))
            (? 2 (? 3 FALLS))
            (? 1 (? 2 FALLS))
            (? 1 (? 2 %( AND))
            (? 1 %( EQ D DO %))
            (? 2 *BED %))))

(=06B (E *E FORTSETZBAR EINFACH)
      (A (DEL %( *OP1 %( *E D2 X2 %) ... X %) =
           = %( *E %( *E D2 X2 %) X %) FALLS *BED))
      =%>
      (*OP1 SCHREIBT FALLS *BED))

(=06C (E *OP1 FORTSETZBAR)
      (E *E FORTSETZBAR EINFACH)
      (E (NICHTEIG *OP1 SCHREIBT))
      (A %( *OP1 %( *E ... D2 ... %) ... %) = (? 1 %( *OP1)
         ... D2 (? 1 ... %))
         (? 2 FALLS *BED))
      =%>
```

```
     (*OP1 LOESCHT (? 2 FALLS *BED)))

(=07 (E *OP1 SCHREIBT)
     (A (DEL %( *OP1 %( *E ... X2 %) ... X %)
          = %( *E ... X %) (? 1 FALLS *BED)))
     (*OP1 SCHREIBT UEBER (? 1 FALLS *BED)))

(=08 (V *LISTE LISTE)
     (V *AKTION (ODER SCHREIBT LOESCHT))
     (E *OP1 *AKTION (? 1 FALLS *BED1))
     (A (DEL %( *OP1 %( *E *LISTE %) ... %)
          = %( *E *LISTE %) (? 2 FALLS *BED)))
     =%>
     (*OP1 *AKTION NICHTS (? 2 FALLS *BED)))

(=09 (E *OP1 FORTSETZBAR)
     (A (DEL %( *OP1 %( *E ... %) ... %)
              = DO (? 1 FALLS *BED)))
     =%>
     (*OP1 LOESCHT AB (? 1 FALLS *BED)))

(=10 (E *OP1 GEMAESS *A)
     (E *A AUSWAHLMENGE)
     (V *AU1 AUSDRUCK)
     (A %( *OP1 —— *A ... %) = (? 1 %( *E)
        ... %( *OP1 *AU1 %( SUB1 *A %) ... %) (? 1 ... %))
        FALLS *BED)
     =%>
     (*A AUSWAHLMENGE ZAEHLER)
     (*OP1 ZAEHLT FALLS *BED))

(=11 (E *OP1 ZAEHLT (DEL FALLS *BED))
     (A (DEL %( *OP1 %( *E D2 ... %) ... %)
      = %( *OP1 %( *OP2 %( *E D2 ... %) %) ... %) FALLS *BED))
     =%>
     (*OP1 ZAEHLT *OP2 FALLS *BED))

(=16 (E *E FORTSETZBAR ZWEIFACH)
     (E *OP1 GEMAESS *A)
     (E *A AUSWAHLMENGE)
     (E *OP1 DURCHLAEUFT P1 FALLS %( EQ %( ANF *A %) *I %))
     (V *AU1 AUSDRUCK)
     (A %( *OP1 —— *A ... %) = (? 1 %( *E)
        ... %( *OP1 *AU1 %( REST *A %) ... %) (? 1 ... %))
        FALLS %( EQ %( ANF *A %) *I %))
     =%>
     (*A AUSWAHLMENGE WORT)
     (*OP1 KNABBERT P1 FALLS %( EQ %( ANF *A %) *I %)))

(=17 (E *E FORTSETZBAR ZWEIFACH)
     (E *OP1 GEMAESS *A)
     (E *A AUSWAHLMENGE)
     (E *OP1 DURCHLAEUFT P2 FALLS %( EQ %( ANF *A %) *J %))
     (V *AU1 AUSDRUCK)
     (A %( *OP1 —— *A ... %) = (? 1 %( *E)
        ... %( *OP1 *AU1 %( REST *A %) ... %) (? 1 ... %))
```



```
        FALLS %( EQ %( ANF *A %) *J %))
     =%>
     (*A AUSWAHLMENGE WORT)
     (*OP1 KNABBERT P2 FALLS %( EQ %( ANF *A %) *J %)))

(=19 (E *OP1 SETZT (DEL *N))
     (V *AUSDR AUSDRUCK)
     (V *VAR AUSDRUCK)
     (A (DEL %( *OP1 *VAR ... %) = *AUSDR (? 1 FALLS *BED)))
     (CALL (UNGLEICH *VAR DO))
     (CALL (UNGLEICH *AUSDR UNDEF))
     =%>
     (*OP1 SETZT *N AUF *AUSDR (? 1 FALLS *BED)))

(=20 (E *OP1 SETZT (DEL *N))
     (V *AUSDR AUSDRUCK)
     (A (DEL %( *OP1 DO ... %) = *AUSDR (? 1 FALLS *BED)))
     (CALL (UNGLEICH *AUSDR UNDEF))
     =%>
     (*OP1 SETZT *N AUF *AUSDR FALLS (? 1 %( AND)
           %( EQ D DO %) (? 1 *BED %)>
```

&-REGELN:

```
  ((&01 (E *E FORTSETZBAR EINFACH)
     =%>
     (D LISTE))

(&02 (E *E FORTSETZBAR ZWEIFACH)
     (E *E OP HILFE)
     =%>
     (D BINBAUM))

(&04 (V *AKTION (ODER LIEST LOESCHT))
     (E *OP1 *AKTION T)
     =%>
     (*OP1 *AKTION NEUSTES))

(&05 (E D LISTE)
     (V *AKTION (ODER LIEST LOESCHT))
     (E *OP1 (DEL DURCHLAEUFT FALLS (? 1 %( AND)
                 %( NOT %( EQ D2 DO %) %) (? 1 *BED %))))
     (E *OP1 *AKTION (DEL FALLS (? 1 %( AND)
                          %( EQ D2 DO %) (? 1 *BED %))))
     =%>
     (*OP1 *AKTION AELTESTES (? 1 FALLS *BED)))

(&06 (E *OP1 (DEL DURCHLAEUFT FALLS *BED))
     (E *OP1 ZAEHLT (DEL FALLS *BED))
     =%>
     (*OP1 ZAEHLT NEUSTES FALLS *BED))

(&07 (V *AKTION (ODER LIEST LOESCHT))
     (E *OP1 ZAEHLT (DEL *OP2 FALLS %( NOT *BED %)))
```

```
     (E *OP1 *AKTION (DEL AELTESTES FALLS *BED))
     (E *OP2 LOESCHT AELTESTES (? FALLS ...))
     =%>
     (*OP1 ZAEHLT AELTESTES FALLS %( NOT *BED %))
     (*OP1 *AKTION FALLS *BED))

(&08 (E *OP1 (DEL DURCHLAEUFT T))
     (E *OP1 SCHREIBT (? 1 FALLS ...))
     =%>
     (*OP1 SCHREIBT AELTESTES))

(&09A (E *E GEMAESS *A)
      (E *A AUSWAHLMENGE T)
      =%>
      (*A AUSWAHLMENGE INDEX))

(&10 (E D LISTE T)
     (E D GEMAESS *A)
     (E *A AUSWAHLMENGE INDEX)
     =%>
     (D LISTE ASSOC))

(&10A (E D LISTE T)
      (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
      (V *STELLE (ODER AELTESTES NEUSTES))
      (E *OP1 GEDENKT *N)
      (E *OP1 (DEL *AKTION FALLS %( EQ *N 1 %)))
      (E *OP1 ZAEHLT *STELLE FALLS %( (? 1 NOT %( EQ)
         (? 2 GREATERP)
         *N 1 (? 1 %))
         %))
      =%>
      (*OP1 *AKTION VERWEIS *N)
      (*N VERWEIS))

(&11A (E (NICHTEIG D AUFGEBAUT))
      (E D LISTE T)
      (E *OP1 OP IMPLEM)
      (E (NICHTEIG D AUFGEBAUT))
      (V *STELLE (ODER NEUSTES AELTESTES))
      (E *OP1 LOESCHT *STELLE (? 1 FALLS *BED))
      =%>
      (D AUFGEBAUT)
      (*OP1 LOESCHT VORNE (? 1 FALLS *BED)))

(&11B (E D LISTE T)
      (E *OP1 OP IMPLEM)
      (E (NICHTEIG D AUFGEBAUT))
      (V *STELLE (ODER NEUSTES AELTESTES))
      (E *OP1 ZAEHLT *STELLE FALLS *BED)
      =%>
      (D AUFGEBAUT)
      (*OP1 ZAEHLT VORNE FALLS *BED))

(&12 (E (NICHTEIG D AUFGEBAUT))
     (E D LISTE T)
```



```
     (V *ART (ODER NICHTS UEBER))
     (E (NICHTEIG *E SCHREIBT (? *ART)
                FALLS *BED))
     =%>
     (*E SCHREIBT VORNE)
     (D AUFGEBAUT))

(&13A (E D LISTE T)
      (E *OP1 LOESCHT VORNE (? 1 FALLS *BED))
      (E *OP1 LOESCHT (DEL NEUSTES (? 1 FALLS *BED)))
      =%>
      (*E SCHREIBT VORNE))

(&13B (E D LISTE T)
      (E *OP1 ZAEHLT VORNE FALLS *BED)
      (E *OP1 ZAEHLT (DEL NEUSTES FALLS *BED))
      =%>
      (*E SCHREIBT VORNE))

(&14A (E D LISTE T)
      (E *OP1 LOESCHT VORNE)
      (E *OP1 LOESCHT (DEL AELTESTES))
      =%>
      (*E SCHREIBT HINTEN))

(&14B (E D LISTE T)
      (E *OP1 ZAEHLT VORNE FALLS *BED)
      (E *OP1 ZAEHLT (DEL AELTESTES FALLS *BED))
      =%>
      (*E SCHREIBT HINTEN))

(&15A (V *AKTION (ODER LIEST LOESCHT))
      (E *OP1 *AKTION (DEL NEUSTES))
      (E *E SCHREIBT VORNE)
      =%>
      (*OP1 *AKTION VORNE))

(&15B (E *OP1 ZAEHLT (DEL NEUSTES FALLS *BED))
      (E *E SCHREIBT VORNE)
      =%>
      (*OP1 ZAEHLT VORNE FALLS *BED))

(&16A (V *AKTION (ODER SCHREIBT LIEST LOESCHT))
      (E *E SCHREIBT HINTEN)
      (E *OP1 *AKTION (DEL NEUSTES (? 1 FALLS *BED)))
      =%>
      (*OP1 *AKTION HINTEN (? 1 FALLS *BED)))

(&16B (E *E SCHREIBT HINTEN)
      (E *OP1 ZAEHLT (DEL NEUSTES FALLS *BED))
      =%>
      (*OP1 ZAEHLT HINTEN FALLS *BED))

(&17A (E *E SCHREIBT VORNE)
      (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
      (E *OP1 *AKTION (DEL AELTESTES (? 1 FALLS *BED)))
```

```
      =%>
      (*OP1 *AKTION HINTEN (? 1 FALLS *BED)))

(&17B (E *E SCHREIBT VORNE)
      (E *OP1 ZAEHLT (DEL AELTESTES FALLS *BED))
      =%>
      (*OP1 ZAEHLT HINTEN FALLS *BED))

(&18A (E D LISTE T)
      (E *E SCHREIBT HINTEN)
      (V *AKTION (ODER SCHREIBT LIEST LOESCHT))
      (E *OP1 *AKTION (DEL AELTESTES (? 1 FALLS *BED)))
      =%>
      (*OP1 *AKTION VORNE))

(&18B (E D LISTE T)
      (E *E SCHREIBT HINTEN)
      (E *OP1 ZAEHLT (DEL AELTESTES FALLS *BED))
      =%>
      (*OP1 ZAEHLT VORNE FALLS *BED))

(&19A (E D LISTE T)
      (E *OP1 OP IMPLEM)
      (E *OP1 LOESCHT HINTEN)
      =%>
      (D LISTE DV))

(&19B (E D LISTE T)
      (E *OP1 OP IMPLEM)
      (E *OP1 ZAEHLT HINTEN FALLS ...)
      =%>
      (D LISTE DV))

(&21 (E D LISTE T)
     <CALL (NULL (EXISTIERT *OP (E *OP LOESCHT HINTEN
                                     (? FALLS ...))
                                (E *OP OP IMPLEM>
     <CALL (NULL (EXISTIERT *OP
                                (E *OP ZAEHLT HINTEN FALLS ...)
                                (E *OP OP IMPLEM>
     =%>
     (D LISTE EV))

(&22 (E D LISTE T)
     (E *OP1 OP IMPLEM)
     (E *OP1 SETZT *N AUF %( SUB1 *N %) (? FALLS ...))
     (E *N VERWEIS)
     =%>
     (D LISTE DV))

(&24 (E D LISTE ASSOC)
     (E D GEMAESS *A)
     (E *A MENGE NATZAHL)
     =%>
     (D LISTE ASSOC GEORDNET))
```



```
(&25 (E D LISTE ASSOC GEORDNET)
     (E D GEMAESS *A)
     (E *A AUSWAHLMENGE INDEX)
     (V *VERGLEICH (ODER LESSP LESSEQP))
     (E *OP1 LOESCHT (DEL FALLS %( *VERGLEICH *A *A2 %)))
     (E *OP1 DURCHLAEUFT FALLS %( NOT %( EQ D2 DO %) %))
     =%>
     (*OP1 LOESCHT AB FALLS %( *VERGLEICH *A *A2 %)))
(&27 (E D LISTE DV)
     =%>
     (D DO-FUNK %( NLAMBDA %( OBEN %) %( SETQ OBEN
          %( CONS %( CONS %) %) %) %( RPLACA %( CAR OBEN %)
       OBEN %)   OBEN %)))

(&29 (E D LISTE DV)
     =%>
     (D DO-FUNK %( NLAMBDA %( OBEN %) %( SETQ OBEN
          %( CONS %) %) %( RPLACA OBEN OBEN %) %)))

(&29A (E D LISTE ASOC ... )
      =%>
      (D DO-FUNK %( NLAMBDA %( %) %( CONS %) %)))

(&30 (E D BINBAUM)
     =%>
     (D DO-FUNK %( NLAMBDA %( %) %( CONS %) %)))

(&31 (E S GEDSTRUKTUR)
     (E *N GEDAECHTNIS)
     (E (? 1 *N MENGE NATZAHL))
     (V 2 NICHT 1)
     (E (? 3 *N VERWEIS))
     (V 4 NICHT 3)
     =%>
     ((? 1 (? 4 *N INIT 0)))
     ((? 1 (? 3 *N INIT %( %))))
     ((? 2 *N INIT %( %))))

(&32 (E D NAME)
     (V *FUNK AUSDRUCK)
     (E D (DEL DO-FUNK *FUNK))
     =%>
     (DO FUNK *FUNK))

(&33 (E S GEDSTRUKTUR)
     (E *NA GEDAECHTNIS 1)
     (V *NAINIT AUSDRUCK)
     (V *NBINIT AUSDRUCK)
     (E *NA INIT *NAINIT)
     (E (? 1 *NB GEDAECHTNIS 2))
     (E (? 1 *NB INIT *NBINIT))
     (V *PARAMS AUSDRUCK)
     (V *FORMEN LISTE)
     (E D (DEL DO-FUNK %( NLAMBDA *PARAMS *FORMEN %)))
     =%>
     (DO FUNK %( NLAMBDA *PARAMS %( LIST
```

```
        %( PROGN *FORMEN %) *NAINIT (? 1 *NBINIT)
        %) %)>
```

I-REGELN:

```
(IO1 (E D LISTE)
     (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
     (V *STELLE (ODER VORNE HINTEN))
     (E *OP1 *AKTION *STELLE)
     (SUCHE (REP FORM))
     =%>
     (%( COND UNDEFINIERT %( T *AKTION *STELLE %) %)))

(IO1A (E S GEDSTRUKTUR)
      (E D LISTE)
      (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
      (E *OP1 *AKTION VERWEIS *N)
      (SUCHE (REP FORM))
      =%>
      (%( COND UNDEFINIERT %( T *AKTION VERWEIS *N %) %)))

(IO2 (E D LISTE)
     (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
     (V *STELLE (ODER VORNE HINTEN))
     (V *ART (ODER NICHTS UEBER))
     (E (NICHTEIG *OP1 *AKTION *STELLE))
     (E (NICHTEIG *OP1 *AKTION VERWEIS *N))
     (E (? 1 *OP1 *AKTION FALLS *BED1))
     (E (? 2 *OP1 LOESCHT AB FALLS *BED2))
     (E (? 3 *OP1 *AKTION *ART FALLS *BED3))
     (E (? 4 *OP1 DURCHLAEUFT FALLS *BED4))
     (V 5 NICHT 4)
     (E (? 5 *OP1 ZAEHLT VORNE FALLS *BED4))
     <CALL (? 1 (IMPLK *BED1 (NOT *BED4>
     <CALL (? 1 (? 2 (IMPLK *BED1 (NOT *BED2>
     <CALL (? 1 (? 3 (IMPLK *BED1 (NOT *BED3>
     <CALL (? 2 (? 3 (IMPLK *BED2 (NOT *BED3>
     <CALL (? 1 (? 3 (IMPLK *BED1 (NOT *BED3>
     <CALL (? 1 (IMPLK *BED1 (NOT *BED4>
     <CALL (? 2 (IMPLK *BED2 (NOT *BED4>
     <CALL (? 3 (IMPLK *BED3 (NOT *BED4>
     (SUCHE (REP FORM))
     =%>
     (%( COND UNDEFINIERT (? 1 %( *BED1 *AKTION VORNE %))
         (? 2 %( *BED2 LOESCHT AB VORNE %))
         (? 3 %( *BED3 *AKTION *ART VORNE %))
         %( T %( APPLY* *OP1 VORNE PARAMS %) %) %)))

(IO3 (E D LISTE DV)
     (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
     (V *STELLE (ODER VORNE HINTEN))
     (V *ART (ODER NICHTS UEBER))
     (E (NICHTEIG *OP1 *AKTION *STELLE))
     (E (NICHTEIG *OP1 *AKTION VERWEIS *N))
     (E *OP1 *AKTION FALLS *BED1)
```



```
     (E (? 1 *OP1 *AKTION *ART FALLS *BED2))
     (E *OP1 ZAEHLT HINTEN FALLS *BED3)
     (CALL (IMPLK *BED1 (NOT *BED3)))
     <CALL (? 1 (IMPLK *BED1 (NOT *BED2>
     <CALL (? 1 (IMPLK *BED2 (NOT *BED3>
     (SUCHE (REP FORM))
     =%>
     (%( COND UNDEFINIERT %( *BED1 *AKTION HINTEN %)
         (? 1 %( *BED2 *AKTION *ART HINTEN %))
         %( T %( APPLY* *OP1 HINTEN PARAMS %) %) %)))

(IO4 (E D BINBAUM)
     (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
     (V *ART (ODER NICHTS UEBER AB))
     (E (? 1 *OP1 UNDEFINIERT FALLS *B1))
     (E (? 2 *OP1 *AKTION FALLS *B2))
     (E (? 3 *OP1 *AKTION *ART FALLS *B3))
     (E *OP1 KNABBERT P1 FALLS *B4)
     (E *OP1 KNABBERT P2 FALLS *B5)
     <CALL (? 1 (? 2 (IMPLK *B1 (NOT *B2>
     <CALL (? 1 (? 3 (IMPLK *B1 (NOT *B3>
     <CALL (? 1 (IMPLK *B1 (NOT *B4>
     <CALL (? 1 (IMPLK *B1 (NOT *B5>
     <CALL (? 2 (? 3 (IMPLK *B2 (NOT *B3>
     <CALL (? 2 (IMPLK *B2 (NOT *B4>
     <CALL (? 2 (IMPLK *B2 (NOT *B5>
     <CALL (? 3 (IMPLK *B3 (NOT *B4>
     <CALL (? 3 (IMPLK *B3 (NOT *B5>
     (CALL (IMPLK *B4 (NOT *B5)))
     (SUCHE (REP FORM))
     =%>
     (%( COND (? 1 %( *B1 %( RETURN %( QUOTE UNDEF %) %) %))
         (? 2 %( *B2 *AKTION VORNE %))
         (? 3 %( *B3 *AKTION *ART VORNE %))
         %( *B4 %( APPLY* *OP1 %( CAR VORNE %) PARAMS %) %)
            %( T %( APPLY* *OP1 VORNE PARAMS %) %) %)))

(IO5 (V *AUSDRUCK AUSDRUCK)
     (E *OP1 SETZT *N AUF *AUSDRUCK (? FALLS *BED))
     (E *N GEDAECHTNIS)
     (SUCHE (REP FORM))
     =%>
     (%( COND UNDEFINIERT
         %( T %( RPLACA *N *AUSDRUCK %) %) %)))

(IO6 (V *LISTE LISTE)
     (E *OP1 PARAMS *LISTE)
     (SUCHE (REP PARAMS))
     =%>
     (*LISTE))

(IO7 (SUCHE (REP %( EQ D2 DO %)))
     =%>
     (%( NULL %( CDR VORNE %) %)))

(IO8 (E *OP1 (DEL UNDEFINIERT FALLS *BED))
```

```
     (E (? 1 *OP1 SCHREIBT))
     (SUCHE (REP UNDEFINIERT))
     =%>
     (%( *BED %( RETURN %( QUOTE UNDEF %) %) %)
        (? 1 UNDEFINIERT)))

(I08B (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
      (E (NICHTEIG *OP1 *AKTION VERWEIS *N))
      (E (NICHTEIG *OP1 ZAEHLT HINTEN FALLS ...))
      (SUCHE (REP %( EQ D DO %)))
      =%>
      (%( NULL VORNE %)))

(I08C (V *AKTION (ODER LIEST LOESCHT SCHREIBT))
      (E *OP1 *AKTION VERWEIS *N)
      (SUCHE (REP %( EQ D DO %)))
      =%>
      (%( NULL %( CDAR *N %) %)))

(I09 (E *OP1 ZAEHLT HINTEN FALLS ...)
     (SUCHE (REP %( EQ D DO %)))
     =%>
     (%( EQ HINTEN URD %)))

(I10 (E (NICHTEIG *OP1 SCHREIBT VORNE (? FALLS)
                ...))
     (E (NICHTEIG *OP1 ZAEHLT HINTEN))
     (E (NICHTEIG *OP1 SCHREIBT HINTEN))
     (E (NICHTEIG *OP1 SCHREIBT VERWEIS *N))
     (E (NICHTEIG *OP1 UNDEFINIERT FALLS ...))
     (E *OP1 SCHREIBT)
     (SUCHE (REP UNDEFINIERT))
     =%>
     (%( %( NULL VORNE %) SCHREIBT VORNE %)))

(I10B (E (NICHTEIG *OP1 UNDEFINIERT FALLS *BED))
      (SUCHE (REP UNDEFINIERT))
      =%>)

(I12B (E D LISTE ASSOC GEORDNET)
      (E (NICHTEIG *OP1 LOESCHT AB FALLS ...))
      (E *OP1 GEMAESS *A)
      (E *A AUSWAHLMENGE INDEX)
      (SUCHE (REP %( NULL VORNE %)))
      =%>
      (%( OR %( NULL %( CDR D %) %) %( LESSP *A *A2 %) %)))

(I13 (V *STELLE (ODER VORNE HINTEN))
     (SUCHE (REP LIEST *STELLE))
     =%>
     (%( KOMP *STELLE %)))

(I13A (SUCHE (REP LIEST VERWEIS *N))
      =%>
      (%( KOMP VERWEIS %( CAR *N %) %)))
```



```
(I15 (E D LISTE EV)
     (SUCHE (REP SCHREIBT VORNE))
     =%>
     (%( REPLACE VORNE %( CONS X VORNE %) %) %( AND
        %( EQ %( CAR URD %) D %) %( RPLACA URD VORNE %) %) D))

(I16 (E D LISTE DV)
     (SUCHE (REP SCHREIBT VORNE))
     =%>
     (%( REPLACE VORNE %( CONS %( CONS D X %) VORNE %) %)
        %( COND %( %( NULL %( CDR VORNE %) %) %( RPLACA %(
        CAR URD %) VORNE %) %) %( T %( RPLACA %( CAR VORNE
        %) VORNE %) %) %) ))

(I17 (E D LISTE ASSOC ...)
     (E *OP1 GEMAESS *A)
     (E *A AUSWAHLMENGE INDEX)
     (SUCHE (REP SCHREIBT VORNE))
     =%>
     (%( REPLACE VORNE %( CONS %( CONS *A X %) VORNE %) %)))

(I17A (E D BINBAUM)
      (SUCHE (REP SCHREIBT VORNE))
      =%>
      (%( REPLACE VORNE %( CONS %( CONS X %) %) %)))

(I18 (E D LISTE EV)
     (SUCHE (REP SCHREIBT HINTEN))
     =%>
     (%( REPLACE HINTEN %( CDR
        %( RPLACD HINTEN %( CONS X %) %) %) %)))

(I19 (E D LISTE DV)
     (SUCHE (REP SCHREIBT HINTEN))
     =%>
     (%( REPLACE HINTEN %( CDR %( RPLACD HINTEN %( CONS
        %( CONS HINTEN X %) %( CDR HINTEN %) %) %) %) %) D))

(I20 (E D LISTE EV)
     (SUCHE (REP SCHREIBT VERWEIS *N))
     =%>
     (%( RPLACD *N %( CONS X %( CDR *N %) %) %) %( AND
        %( EQ HINTEN *N %) %( RPLACA D %( CDR *N %) %) %)))

(I21 (SUCHE (REP SCHREIBT UEBER VORNE))
     =%>
     (%( REPLACE %( KOMP VORNE %) X %) D))

(I22A (V *STELLE (ODER VORNE HINTEN))
      (SUCHE (REP LOESCHT NICHTS *STELLE))
      =%>
      (D))

(I22B (SUCHE (REP LOESCHT NICHTS VERWEIS *N))
      =%>
      (D))
```

```
(I22C (V *STELLE (ODER VORNE HINTEN))
      (SUCHE (REP SCHREIBT NICHTS *STELLE))
      =%>
      (D))

(I22D (SUCHE (REP SCHREIBT NICHTS VERWEIS *N))
      =%>
      (D))

(I23 (E D LISTE EV)
     (SUCHE (REP LOESCHT VORNE))
     =%>
     (%( AND %( EQ %( CAR URD %) VORNE %) %( RPLACA URD D %)
        %( %( REPLACE VORNE %( CDR VORNE %) %)))

(I24 (E D LISTE DV)
     (SUCHE (REP LOESCHT VORNE))
     =%>
     (%( COND %( %( NULL %( CDR VORNE %) %) %( RPLACA
        %( CAR URD %) D %) %) %( T %( RPLACA %( CADR VORNE %)
        D  %) %) %) %( REPLACE VORNE %( CDR VORNE %) %)))

(I25 (E D LISTE ASSOC ...)
     (SUCHE (REP LOESCHT VORNE))
     =%>
     (%( REPLACE VORNE %( CDR VORNE %) %)))

(I25A (E D LISTE ASSOC GEORDNET)
      (SUCHE (REP LOESCHT AB VORNE))
      =%>
      (%( REPLACE VORNE %)))

(I25B (E D BINBAUM)
      (SUCHE (REP LOESCHT AB VORNE))
      =%>
      (%( REPLACE VORNE %)))

(I26 (E D LISTE DV)
     (SUCHE (REP LOESCHT HINTEN))
     =%>
     (%( REPLACE HINTEN %( CAAR HINTEN %) %) %( COND
         %( %( EQ URD D %) %( RPLACD HINTEN %) %) %( T %(
         RPLACD HINTEN D %) %) %) D))


(I27 (E D LISTE DV)
     (SUCHE (REP LOESCHT VERWEIS *N))
     =%>
     (%( AND %( EQ HINTEN %( CDR *N %) %) %( RPLACA D *N %) %)
          %( RPLACD *N %( CDDR *N %) %)))

(I28 (E D LISTE ASSOC ...)
     (E *OP1 GEMAESS *A)
     (SUCHE (REP *A2))
     =%>
```



```
     (%( CAAR VORNE %)))

(I29 (SUCHE (REP X2))
     =%>
     (%( KOMP VORNE %)))

(I30 (E D LISTE)
     (E *N VERWEIS)
     (E *OP1 GEDENKT *N)
     (SUCHE (REP %( LESSP *N 1 %)))
     =%>
     (%( EQ D %( CDAR *N %) %)))

(I30A (E D LISTE)
      (E *N VERWEIS)
      (E *OP1 GEDENKT *N)
      (SUCHE %( RPLACA *N (REP 0)
           %))
      =%>
      (D))

(I30B (E S GEDSTRUKTUR)
      (E *NA GEDAECHTNIS)
      (E (NICHTEIG *NA VERWEIS))
      (E *NB GEDAECHTNIS)
      (E (NICHTEIG *NB VERWEIS))
      (CALL (UNGLEICH *NA *NB))
      (SUCHE (REP %( LESSP *NA *NB %)))
      =%>
      (%( LESSP %( CAR *NA %) %( CAR *NB %) %)))

(I30C (E S GEDSTRUKTUR)
      (E *N GEDAECHTNIS)
      (E (NICHTEIG *N VERWEIS))
      (SUCHE %( LESSP 0 (REP *N)
           %))
      =%>
      (%( CAR *N %)))

(I31 (E D LISTE)
     (E *N GEDAECHTNIS)
     (E (? 1 *N VERWEIS))
     (V 2 NICHT 1)
     (E *OP1 GEDENKT *N)
     (SUCHE %( RPLACA *N (REP %( ADD1 *N %))
          %))
     =%>
     ((? 1 %( CDAR *N %) %))
      (? 2 %( ADD1 %( CAR *N %) %) %))))

(I32 (E D LISTE)
     (E *N GEDAECHTNIS)
     (E (? 1 *N VERWEIS))
     (V 2 NICHT 1)
     (E *OP1 GEDENKT *N)
     (SUCHE %( RPLACA *N (REP %( SUB1 *N %))
```

```
        %))
    =%>
    ((? 1 %( CAAAR *N %))
     (? 2 %( SUB1 %( CAR *N %) %))))

(I33 (SUCHE (REP VORNE))
    =%>
    (%( CDR D %)))

(I34 (E D LISTE EV)
    (SUCHE (REP HINTEN))
    =%>
    (%( CAR D %)))

(I35 (E D LISTE DV)
    (SUCHE (REP HINTEN))
    =%>
    (%( CAR %( CAR D %) %)))

(I36 (E D LISTE EV)
    (SUCHE (REP KOMP))
    =%>
    (CAR))

(I37 (V *ART (ODER DV ASSOC))
    (E D LISTE *ART ...)
    (V *LISTE LISTE)
    (SUCHE (REP KOMP *LISTE %)))
    =%>
    (CDR %( CAR *LISTE %) %)))

(I37A (E D BINBAUM)
     (V *LISTE LISTE)
     (SUCHE (REP KOMP *LISTE %)))
     =%>
     (CAR %( CAR *LISTE %) %)))

(I38 (SUCHE (REP CHECK))
    =%>
    (%( COND %( %( TEST 1 *ADT-NAME %) NIL %)
       %( T %( FEHLER 1 *ADT-NAME %) %( RETURN %) %) %)>
```

B-REGELN:

```
(B01 (V *VAR AUSDRUCK)
    (SUCHE (REP REPLACE %( CAR *VAR %)))
     =%>
    (RPLACA *VAR))

(B02 (V *VAR AUSDRUCK)
    (SUCHE (REP REPLACE %( CDR *VAR %)))
    =%>
    (RPLACD *VAR))
```



```
(B03 (E D LISTE)
    (SUCHE (REP VERWEIS))
    =%>)

(B04 (V *VAR1 LISTE)
    (SUCHE (REP %( COND %( T *VAR1 %) %)))
    =%>
    (*VAR1))

(B07 (E *OP1 (DEL ZAEHLT))
    (E *OP1 GEMAESS *A)
    (E *A AUSWAHLMENGE ZAEHLER)
    (SUCHE %( APPLY* *OP1 ... (REP *A)
          ... %))
    =%>
    (%( SUB1 *A %)))

(B08 (E D BINBAUM)
    (E *OP1 KNABBERT)
    (E *OP1 GEMAESS *A)
    (E *A AUSWAHLMENGE WORT)
    (SUCHE %( APPLY* *OP1 ... (REP *A)
          ... %))
    =%>
    (%( REST *A %)>
```

5.2. Systembeschreibung ALGCOMP

In 5.2.1. werden zunächst die Funktionen des ALGCOMP-System dokumentiert. Da wir einen Teil des ADTCOMP-Pattern-Matchers auch an dieser Stelle verwenden, nämlich die Funktion REGEL-PRUEF mit ihren Unterfunktionen, die schon in 5.1.1. besprochen wurden, werden diese Funktionen hier nicht mehr aufgeführt. Die entsprechenden Funktionen sind in der Datei LISP.DATA.PMREST.nn zugreifbar; man muß also, um einen Algorithmus mit ALGCOMP zu implementieren, die folgenden drei Dateien laden:

LISP.DATA.PM-ALG.nn,

LISP.DATA.PMREST.nn und

LISP.DATA.ALG-REGELN.nn

Anschließend werden in 5.2.2. die Regeln des ALGCOMP-System besprochen.

5.2.1. Funktionen des ALGCOMP-Systems

Hauptfunktion zur Algorithmus-Implementierung ist die Funktion IMPLEMENTIERE-ALG, die durch die drei Unterfunktionen RALGSPEC, ALG-KONSTRUKTIONS-PHASE und LISP-PHASE realisiert wird.
Mittels der Funktion RALGSPEC wird die Spezifikation eines zu implementierenden Algorithmus eingelesen und auf Korrektheit überprüft. Außerdem werden die für eine Algorithmus-Implementierung globalen Variablen, also die innerhalb der Funktion IMPLEMENTIERE-ALG lokalen, initialisiert und teilweise mit Werten belegt.
Die Funktion ALG-KONSTRUKTIONS-PHASE steuert den Aufruf der sechs Schritte zur Implementierung des eigentlichen Algorithmus. Das Ergebnis von ALG-KONSTRUKTIONS-PHASE wird unter dem Indikator "Algorithmus" an der Property-List des literalen Atoms "ALGNAME" abgelegt.
Die Funktion LISP-PHASE erzeugt die zu einem Algorithmus gehörige Arbeitsumgebung, die an die Funktions-Definitions-Zelle des literalen Atoms "ALGNAME" gebunden wird.

Zunächst wird in 5.2.1.1. die Hauptfunktion IMPLEMENTIERE-ALG mit den für eine Algorithmus-Implementierung globalen Variablen besprochen. Es folgen dann

- in 5.2.1.2. die Einlesefunktion RALGSPEC mit ihren Unterfunktionen,
- in 5.2.1.3. die Hauptfunktion der Algorithmus-Konstruktions-Phase ALG-KONSTRUKTIONS-PHASE mit ihren Unterfunktionen und
- in 5.2.1.4. die Environment-Implementierungs-Funktion LISP-PHASE mit ihren Unterfunktionen.

5.2.1.1. Hauptfunktion zur Algorithmus-Implementierung und globale Variablen

<IMPLEMENTIERE-ALG <LAMBDA (EINGABEDATEI GBC-FLAG ...) ...>>

Zweck: Hauptfunktion zur Algorithmus-Implementierung. IMPLEMENTIERE-ALG steuert den Aufruf der drei Phasen einer Algorithmus-Implementierung und bildet die Umgebung, in der alle für einen Implementierungslauf globalen Variablen lokal definiert sind.

Aufruf: Benutzer

Param.: EINGABEDATEI ist der Name der Datei, von der eine Algorithmus-Spezifikation eingelesen werden soll. Falls EINGABEDATEI=NIL, erwartet RALGSPEC die Eingabe der Spezifikation über den Bildschirm.
GBC-FLAG#NO-GBC aktiviert nach jeder Phase bzw. jedem Schritt der Algorithmus-Konstruktions-Phase den Garbage-Collector des LISP-Systems (unter Umständen bei umfangreichen Implementierungen und kleinen LISP-Systemen notwendig).

Ergebnis: Falls syntaktische Fehler in der Spezifikation erkannt werden, wird die Ausführung von IMPLEMENTIERE-ALG mit einer entsprechenden Fehlermeldung abgebrochen. Sonst wird der spezifizierte Algorithmus implementiert, sofern in den Regeln genügend Wissen codiert ist.

Wert: NIL

```
<IMPLEMENTIERE-ALG
  <LAMBDA (EINGABEDATEI GBC-FLAG URBINDUNGEN ALGPARAM ALGNAME
                        PARAM-TYP-LISTE ALGORITHMUS-WERT
                        RELATION HILFSALGORITHMEN
                        HILFSFUNKTIONEN VORAUSSETZUNGEN
                        ALGORITHMUS KOMMENTAR
                        LINKESEITEN RECHTESEITEN
                        PRA REK SUCC-CNT ATTEMPT-CNT)
   (* "IMPLEMENTIERE-ALG RUFT FOLGENDE FUNKTIONEN AUF: RALGSPEC
       REMPROPS, ALG-KONSTRUKTIONS-PHASE, LISP-PHASE, PRINTL")
     (SETQ ATTEMPT-CNT 0)
     (SETQ SUCC-CNT 0)
     (PRINTL "START ALGORITHMUS-IMPLEMENTIERUNG AM" (DATE))
     (TERPRI)
     (PRINTL "START EINLESE-PHASE")
     (COND
       ((RALGSPEC EINGABEDATEI)
         (PRINTL "EINLESE-PHASE BEENDET")
         (TERPRI)
         (REMPROPS ALGNAME)
         (PRINTL "START ALG-KONSTRUKTIONS-PHASE")
         (TERPRI)
         (ALG-KONSTRUKTIONS-PHASE)
         (PRINTL "ALG-KONSTRUKTIONS-PHASE BEENDET")
         (TERPRI)
         (PRINTL "START LISP-PHASE")
         (LISP-PHASE)
```

```
    (PRINTL "LISP-PHASE BEENDET")
    (PRINTL "ALGORITHMUS-IMPLEMENTIERUNG
             BEENDET AM" (DATE)))
  (T NIL>>
```

■

Globale Variablen

Alle globalen Variablen werden durch IMPLEMENTIERE-ALG und RALGSPEC mit NIL, 0 oder (CONS) initialisiert.
Es folgt eine Liste der globalen Variablen, wobei jeweils angegeben wird, welche Funktion die Variable mit welchem Wert belegt.

GBC-FLAG: Falls GBC-FLAG ungleich NO-GBC ist, wird nach jeder Phase und jedem Schritt der Algorithmus-Konstruktions-Phase der Garbage-Collector des LISP-Systems aktiviert.
(Unter Umständen bei umfangreichen Implementierungen und kleinen LISP-System notwendig.)

URBINDUNGEN Def.: durch RALGSPEC
Wert: ist eine Assoziations-Liste mit den Paaren (*ALGNAME . ...) und (*PARLIST . (...)) .

ALGPARAM Def.: durch RALGSPEC
Wert: Liste der Parameter-Namen gemäß Angabe unter dem Schlüsselwort "Algorithmus"

ALGNAME Def.: durch RALGSPEC
Wert: (Algname)

PARAM-TYP-LISTE Def.: durch RALGSPEC
Wert: ist eine Assoziations-Liste mit Paaren (Parameter-Name . Datentyp) für alle Parameter.
Property-List: Indikator Benutzer:
Assoziations-Liste mit Paaren der Form (Parameter-Name . Datentyp) für alle Benutzer-Parameter.
Property-List: Indikator System:
Assoziations-Liste mit Tripeln der Form (Parameter-Name Datentyp Wert) für alle System-Parameter gemäß der Spezifikation

ALGORITHMUS-WERT Def.: durch RALGSPEC
Wert: Treten bei Algorithmen mit parallel-rekursiven Produktionen mehrere rechte Seiten mit nur einer Variablen auf, so muß hier diejenige angegeben werden, die bei Ausführung des Algorithmus dessen Wert enthalt; sonst Fehlermeldung.

RELATION Def.: durch RALGSPEC
Wert: Name des bei der Implementierung zu verwendenden Relationssatzes; bei Nichtangabe wird der Relationssatz "Standard" benutzt.

HILFSALGORITHMEN Def.: durch RALGSPEC
Wert: Liste der Namen der Hilfsalgorithmen gemäß Spezifikation.

HILFSFUNKTIONEN Def.: durch RALGSPEC und in den Voraussetzungen der Regeln S2-9 und S6-1.
Wert:
Liste der erkannten Hilfsfunktionen; hierbei kann RALGSPEC Hilfsfunktionen erkennen, die unter dem Schlüsselwort "Voraussetzungen" in der Spezifikation auftreten, während die in der Produktionsregelnmenge benutzten durch die Regeln S2-9 und S6-1 erkannt werden.

VORAUSSETZUNGEN Def.: durch RALGSPEC
Wert: Liste der Voraussetzungen gemäß Spezifikation.

ALGORITHMUS Def.: durch RALGSPEC
Wert: Darstellung der Produktionsregelmenge als Liste von Atomen; auf einer Kopie dieser Liste arbeitet die Funktion REGELPARSER bei der Syntax-Überprüfung der Produktionsregelmenge.

KOMMENTAR Def.: durch RALGSPEC
Wert: Darstellung des Kommentars als Liste gemäß Spezifikation.

LINKESEITEN Def.: durch REGELPARSER und REGELP
Wert: Liste der Form (L1 L2 ... Ln).

RECHTESEITEN Def.: durch REGELPARSER und RECHTESEITEP
Wert: Liste der Form (R1 R2 ... Rn).

Bemerkung: REGELP erzeugt globale Atome L1,L2, ... Ln, RECHTESEITEP Atome R1,R2, ... Rn , deren Wert die jeweilige linke bzw. rechte Seite einer Produktion in interner Darstellung ist.

PRA Def.: durch PAR-REK-AUFRUFP
Wert: T, falls eine Produktion mit einem parallel-rekursiven Aufruf des Algorithmus erkannt wird; NIL sonst.

REK Def.: durch REGEL S4-1
Wert T, falls eine Produktion mit einem rekursiven Aufruf des Algorithmus erkannt wird; NIL sonst.

SUCC-CNT Def.: durch SCHRITT-i Funktionen.
Wert: Anzahl der erfolgreichen Regelanwendungen.

ATTEMPT-CNT Def.: durch SCHRITT-i Funktionen.
Wert: Anzahl ver versuchten Regelanwendungen.

5.2.1.2. Funktionen der Einlesephase

Hauptfunktion der Einlese-Phase ist RALGSPEC. Unterfunktionen von RALGSPEC werden in 5.2.1.2.2. besprochen (insbesondere REGELPARSER).

5.2.1.2.1. Einlesefunktion RALGSPEC

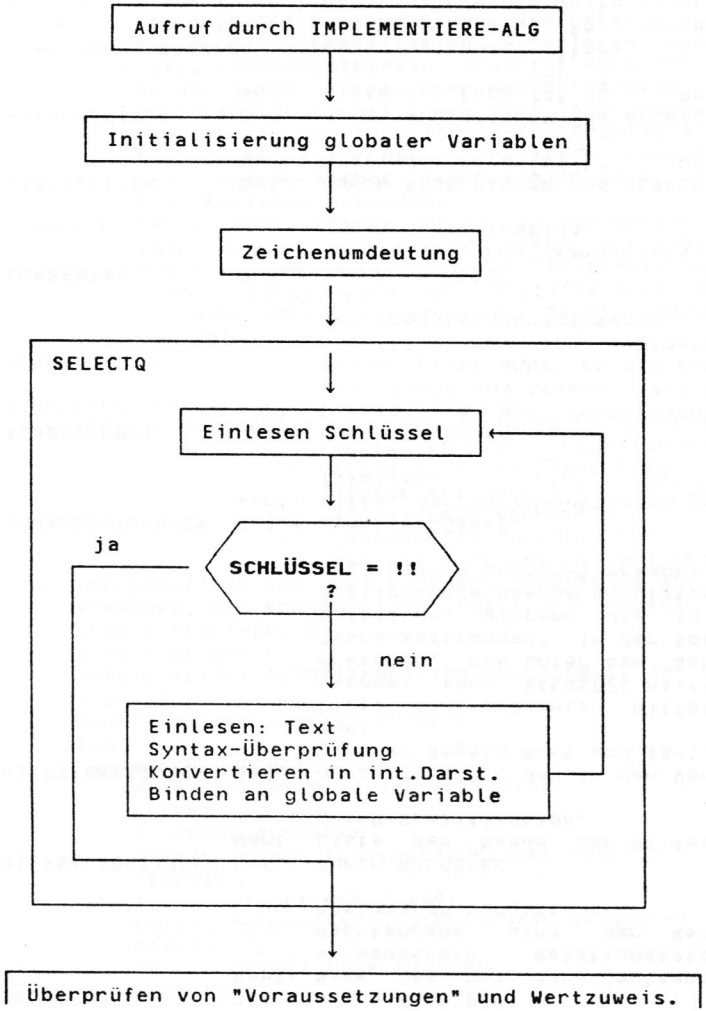
<RALGSPEC <LAMBDA (EINGABEDATEI ...) ... >>

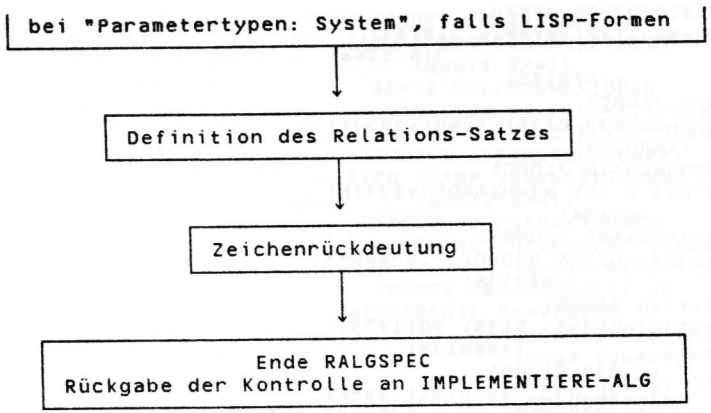
Zweck: Einlesen der Algorithmus-Spezifikation von einer Datei oder vom Bildschirm und überprüfen der syntaktischen Korrektheit durch Aufruf der Funktion REGELPARSER.
Aufruf: durch IMPLEMENTIERE-ALG
Param.: EINGABEDATEI wird von IMPLEMENTIERE-ALG übernommen; (s.a. 5.2.1.1. IMPLEMENTIERE-ALG)
Ergebnis: Bindung von Werten an globale Variablen gemäß der Spezifikation (siehe 5.2.1.1. Globale Variablen). Im Fehlerfall Ausgabe einer entsprechenden Fehlermeldung, Abbruch der Implementierung und Rückkehr zum Top-Level des LISP-Systems.
Wert: T, im Fehlerfall NIL
Bem.: Die Spezifikation wird atomweise eingelesen. Ein ";" bewirkt Interpretation des folgenden Atoms als Schlüsselwort mittels SELECTQ. Ausnahme: Innerhalb der Produktionsregelmenge schließt ein ";" eine Produktion ab. "!!" beendet das Einlesen.

Innerhalb von RALGSPEC sind die Zeichen ", ; !" als Break-Character vereinbart, "< > '" sind normale Zeichen, ":" ist ein Separator.
Für jedes Schlüsselwort wird der in der Spezifikation folgende Text, soweit möglich, syntaktisch überprüft (s.a. 2.4.2. Spezifikationssprachen), in interne Darstellung konvertiert und an die entsprechende(n) globale(n) Variable(n) gebunden. Wird ein Fehler erkannt, erfolgt eine entsprechende Fehlermeldung und der Implementierungslauf wird abgebrochen.

Die Produktionsregelmenge muß der in 2.4.2. angegebenen Grammatik genügen. Die einzelnen Produktionsregeln werden in linke und rechte Seiten (Test und Aktion) aufgespalten und in interner Darstellung an entsprechend numerierte globale Variablen L1,L2, ... Ln, R1,R2, ... Rn gebunden.
Für jedes Schlüsselwort wird das Ergebnis ausgedruckt.

Blockdiagramm für RALGSPEC:






```
<RALGSPEC
  <LAMBDA (EINGABEDATEI ABBRUCH)
    (* "RALGSPEC RUFT FOLGENDE FUNKTIONEN AUF:
        RATOM-NE, REGELPARSER, PARAMETER-
        UEBERPRUEFUNG, ABBRUCH, EKL,
        FKT-SYMBOL-IN, PRINTL, *COM*")
    (PRINTL "ALG-EINGABE UEBER DATEI:"
            (COND (EINGABEDATEI EINGABEDATEI)
                  (T "BILDSCHIRM")))
 (*COM* "INITIALISIERUNG DER GLOBALEN VARIABLEN")
    (PROGN (SETQ PARAM-TYP-LISTE)
           (PUT (QUOTE PARAM-TYP-LISTE)
                (QUOTE BENUTZER)
                NIL)
           (PUT (QUOTE PARAM-TYP-LISTE)
                (QUOTE SYSTEM)
                NIL)
           (SETQ ALGORITHMUS-WERT)
           (SETQ RELATION)
           (SETQ HILFSALGORITHMEN)
           (SETQ VORAUSSETZUNGEN)
           (SETQ KOMMENTAR)
           (SETQ HILFSFUNKTIONEN)
           (SETQ PRA)
           (SETQ REK)
           (SETQ VERWENDET (CONS))
           (SETSEPR (CHCON (QUOTE :))
                    1)
           (SETBRK (CHCON (QUOTE ,;!))
                   1)
           (SETBRK (QUOTE (125))
                   0)
           (*COM* "ZEICHENBEDEUTUNG:
                   : ALS SEPARATOR UND , ; ! ALS BREAKS")
           (*COM* < , > und ' NICHT MEHR ALS BREAKS")
 (*COM* "EINLESE-PROG: SELEKTIERT NACH ALG-SCHLUESSEL.
```



```
        BEI UNZULAESSIGEM SCHLUESSEL WIRD DAS
        NAECHSTE EINGABEWORT ALS ALG-SCHLUESSEL
        INTERPRETIERT!")
    (PROG (ALG-SCHLUESSEL FEHLSCHLUESSEL LISTE
           (ALTEINGABE (INFILE EINGABEDATEI)))
      NEUSCHLUESSEL
          (SETQ UNGUELTIG NIL)
          (SETQ ALG-SCHLUESSEL (RATOM))
      FEHLSCHLUESSEL
 (SELECTQ ALG-SCHLUESSEL
(*COM* "ALG-SCHLUESSEL ALGORITHMUS-NAME:"
        DEF. VON URBINDUNGEN UND ALGNAME")
  <ALGORITHMUS-NAME (SETQ ALGNAME (RATOM-NE ;))
                    (SETQ URBINDUNGEN
                          (LIST (CONS (QUOTE *ALGNAME)
                                      ALGNAME)))
                    (PRINTL "ALGORITHMUS-NAME:" ALGNAME)
                    (COND
                      ((AND (LITATOM (CAR ALGNAME))
                            (NOT (CDR ALGNAME)))
                       (GO NEUSCHLUESSEL))
                      (T (PRINTL "***ERROR:
                          ALGORITHMUS-NAME FALSCH")
                         (ABBRUCH>
(*COM* "ALG-SCHLUESSEL PARAMETERTYPEN:
        DURCH PROG UNTERTEILT IN BENUTZER- UND
        SYSTEMPARAMETER. SELEKTION UEBER PARAM-SCHLUESSEL")
  (PARAMETERTYPEN (PROG (PARAM-SCHLUESSEL)
                        PARAMNEUSCHLUESSEL
                        (SETQ UNGUELTIG1 NIL)
                        (SETQ PARAM-SCHLUESSEL (RATOM))
                        (SELECTQ PARAM-SCHLUESSEL
(*COM* "PARAM-SCHLUESSEL BENUTZER:
        DEF. DER PROPERTY-LIST VON PARAM-TYP-LISTE UNTER
        DEM INDIKATOR BENUTZER UND DES WERTES")
                          (BENUTZER
                            (PUT (QUOTE PARAM-TYP-LISTE
                                 (QUOTE BENUTZER)
                                 (MAPLIST (RATOM-NE ;)
                                   <FUNCTION (LAMBDA (LIST)
                                     (CONS (CADR LIST)
                                           (CAR LIST>
                                     (FUNCTION CDDDR)))
                                  <SETQ PARAM-TYP-LISTE
                                   (APPEND
                                    PARAM-TYP-LISTE
                                    (MAPCAR (GETP (QUOTE
                                        PARAM-TYP-LISTE)
                                            (QUOTE
                                        BENUTZER))
                                        (FUNCTION
                                          (LAMBDA (PARAM)
                                            (CONS (CAR PARAM)
                                                  (CDR PARAM>
                                   (PRINTL "PARAMETERTYPEN:
                                    BENUTZER:"
```

```
                        (GETP (QUOTE PARAM-TYP-LISTE)
                              (QUOTE BENUTZER)))
                       (GO PARAMNEUSCHLUESSEL))
(*COM* "PARAM-SCHLUESSEL SYSTEM:
       DEF. DER PROPERTY-LIST VON PARAM-TYP-LISTE UNTER
       DEM INDIKATOR SYSTEM UND DES WERTES")
                  <SYSTEM
                       (SETSEPR (CHCON (QUOTE =))
                                1)
                       (PUT (QUOTE PARAM-TYP-LISTE)
                            (QUOTE SYSTEM)
                            (MAPLIST
                             (EKL (RATOM-NE ;))
                             <FUNCTION (LAMBDA (LIST)
                               (LIST (CADR LIST)
                                     (CAR LIST)
                                     (CADDR LIST>
                                (FUNCTION CDDDDR)))
                       (SETSEPR (CHCON (QUOTE =))
                                0)
                       <SETQ PARAM-TYP-LISTE
                         (APPEND PARAM-TYP-LISTE
                                 (MAPCAR
                                  (GETP (QUOTE
                                 PARAM-TYP-LISTE)
                                        (QUOTE
                                 SYSTEM))
                                  (FUNCTION
                                   (LAMBDA (PARAM)
                                    (CONS (CAR PARAM)
                                          (CADR PARAM>
                       (PRINTL "SYSTEM:   " (GETP (QUOTE
                                           PARAM-TYP-LISTE)
                                                  (QUOTE
                                           SYSTEM>
(*COM* "BEI DEN FOLGENDEN PARAM-SCHLUESSELN WIRD MITTELS
       FEHLSCHLUESSEL WIEDER AUF ALG-SCHLUESSEL
       GESCHALTET UND INS AEUSSERE PROG ZURUECKGESPRUNGEN")
                       (WERT (SETQ FEHLSCHLUESSEL
                                   (QUOTE WERT))
                             (RETURN))
                       (RELATION (SETQ FEHLSCHLUESSEL
                                       (QUOTE RELATION))
                                 (RETURN))
                       (VORAUSSETZUNGEN (SETQ FEHLSCHLUESSEL
                                        (QUOTE VORAUSSETZUNGEN))
                                        (RETURN))
                       (HILFSALGORITHMEN (SETQ FEHLSCHLUESSEL
                                         (QUOTE HILFSALGORITHMEN))
                                         (RETURN))
                       (ALGORITHMUS (SETQ FEHLSCHLUESSEL
                                          (QUOTE ALGORITHMUS))
                                    (RETURN))
                       (PROG NIL
                          (PRINTL "PARAMETERTYPEN-SCHLUESSEL"
                                  PARAM-SCHLUESSEL
```



```
                                  "NICHT ZULAESSIG")
                             (SETQ UNGUELTIG1 T)))
                        (COND
                          (UNGUELTIG1 (SETQ UNGUELTIG1 NIL)
                                      (GO PARAMNEUSCHLUESSEL))
                          (T NIL)))
                      (COND
                        (FEHLSCHLUESSEL (SETQ ALG-SCHLUESSEL
                                              FEHLSCHLUESSEL)
                                        (GO FEHLSCHLUESSEL))
                        (T NIL))
                       (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL WERT: DEF. VON ALGORITHMUS-WERT")
 (WERT (SETQ ALGORITHMUS-WERT (RATOM-NE ;)))
(*COM* "ALG-SCHLUESSEL RELATION: DEF: VON RELATION
        UND TEST, OB RELATIONSSATZ DEFINIERT")
 (RELATION (SETQ RELATION (RATOM-NE ;))
           (COND ((EQ (CAAR RELATION)
                      (QUOTE NOBIND))
                  (PRINTL "***ERROR: RELATIONSSATZ"
                          (CAR RELATION)
                          "NICHT DEFINIERT")
                  (PRINTL "DEFINITION DURCH FUNKTION:
                           MAKE-REL MOEGLICH")
                  (ABBRUCH))
                 (T T))
           (PRINTL "RELATION:" RELATION)
           (PRINTL "RELATIONSSATZ:" (CAAR RELATION))
           (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL VORAUSSETZUNGEN:
       DEF. VON VORAUSSETZUNGEN, ERSETZEN DER
       RELATIONSZEICHEN DURCH LISP-FKT. UND
       ERGAENZEN DER HILFSFUNKTIONENLISTE UM
       EBENDIESE LISP-FKT")
 (VORAUSSETZUNGEN
           (SETQ VORAUSSETZUNGEN
             (MAPLIST (EKL (RATOM-NE ;))
               <FUNCTION (LAMBDA (VORAUSSETZUNG)
                 (LIST (COND
                         ((EQ (CADR VORAUSSETZUNG)
                              (QUOTE =)
                          (QUOTE EQ)
                         ((EQ (CADR VORAUSSETZUNG)
                              (QUOTE /=))
                          (QUOTE NEQ))
                         ((EQ (CADR VORAUSSETZUNG)
                              (QUOTE %<))
                          (SETQ HILFSFUNKTIONEN
                            (NCONC1 HILFSFUNKTIONEN
                                    (QUOTE KLEINERP)))
                          (QUOTE KLEINERP))
                         ((EQ (CADR VORAUSSETZUNG)
                              (QUOTE %>))
                          (SETQ HILFSFUNKTIONEN
                            (NCONC1 HILFSFUNKTIONEN
                                    (QUOTE GROESSERP)))
```

```
                    (QUOTE GROESSERP))
                   ((EQ (CADR VORAUSSETZUNG)
                        (QUOTE %<=))
                    (SETQ HILFSFUNKTIONEN
                      (NCONC1 HILFSFUNKTIONEN
                              (QUOTE KLEINERGLEICHP)))
                    (QUOTE KLEINERGLEICHP))
                   ((EQ (CADR VORAUSSETZUNG)
                        (QUOTE %>=))
                    (SETQ HILFSFUNKTIONEN
                      (NCONC1 HILFSFUNKTIONEN
                              (QUOTE GROESSERGLEICHP)))
                    (QUOTE GROESSERGLEICHP))
                   (T (PRINTL "***ERROR:
                               RELATIONSZEICHEN"
                              (CADR VORAUSSETZUNG)
                              "IN VORAUSSETZUNG"
                              VORAUSSETZUNG
                              "NICHT ERLAUBT")
                      (ABBRUCH)))
                  (CAR VORAUSSETZUNG)
                  (CADDR VORAUSSETZUNG>
          (FUNCTION CDDDDR)))
      <SETQ HILFSFUNKTIONEN
          (REMOVE NIL
            (MAPCAR HILFSFUNKTIONEN
              (FUNCTION (LAMBDA (HILFSFKT)
                (COND
                  ((MEMB HILFSFKT LISTE)
                   NIL)
                  (T (SETQ LISTE (NCONC1 LISTE
                                         HILFSFKT))
                     HILFSFKT>
      (PRINTL "VORAUSSETZUNGEN: " VORAUSSETZUNGEN)
      (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL HILFSALGORITHMEN:
        DEF. VON HILFSALGORITHMEN")
  (HILFSALGORITHMEN <MAPC (RATOM-NE ;)
                    (FUNCTION (LAMBDA (ATOM)
                      (SETQ HILFSALGORITHMEN
                        (APPEND HILFSALGORITHMEN
                                (COND
                                  ((EQ ATOM (QUOTE ,))
                                   NIL)
                                  (T (LIST ATOM>
                    (PRINTL "HILFSALGORITHMEN:"
                            HILFSALGORITHMEN)
                    (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL ALGORITHMUS:
        ALGORITHMUSNAMEN-TEST, DEF. VON ALGPARAM,
        AUFRUF FKT. PARAMETER-UEBERPRUEFUNG,
        DEF. VON URBINDUNGEN UND ALGORITHMUS,
        AUFRUF DES REGELPARSERS")
  (ALGORITHMUS <COND
                 ((NEQ (CAR ALGNAME)
                       (RATOM))
```



```
                  (PRINTL "***ERROR: ALGORITHMUSNAME IN
                           ALGORITHMUSDEFINITION FALSCH")
                  (ABBRUCH))
                 (T (SETQ ALGPARAM (MAPLIST (CDR (RATOM-NE =))
                                            (FUNCTION <LAMBDA (LISTE)
                                              (CAR LISTE>)
                                            (FUNCTION CDDR)))
                    (PARAMETER-UEBERPRUEFUNG)
                    (SETQ URBINDUNGEN
                      (NCONC1 URBINDUNGEN
                              (CONS (QUOTE *PARLIST)
                                    ALGPARAM)))
                    (COND
                      ((EQ (RATOM)
                           (QUOTE {))
                       NIL)
                      (T (PRINTL "***ERROR: REGELMENGE DES
                                  ALGORITHMUS BEGINNT NICHT MIT {")
                         (ABBRUCH)))
                    (SETBRK (QUOTE (76 110))
                            0)
                    (SETQ ALGORITHMUS (RATOM-NE }))
                    (SETBRK (QUOTE (76 110))
                            1)
                    (SETQ ALGORITHMUS
                          (NCONC1 ALGORITHMUS (QUOTE }))))
                    (COND
                      ((REGELPARSER)
                       (GO NEUSCHLUESSEL))
                      (T (ABBRUCH>
                (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL KOMMENTAR: DEF. VON KOMMENTAR")
  (KOMMENTAR (SETQ KOMMENTAR (RATOM-NE !))
             (PRINTL "KOMMENTAR:       " KOMMENTAR)
             (GO NEUSCHLUESSEL))
(*COM* "ALG-SCHLUESSEL !: SCHLIESSEN DER EINGABEDATEI")
  <! (PRINTL "ALG-EINGABE BEENDET.")
     (CLOSEF EINGABEDATEI)
     (COND
       ((EQ ALTEINGABE T)
        (INFILE))
       (T (INFILE ALTEINGABE>
  (PROG NIL
        (PRINTL "ALG-SCHLUESSEL" ALG-SCHLUESSEL
                "NICHT ZULAESSIG")
        (SETQ UNGUELTIG T)))
(*COM* "TEST DER FUNKTIONEN IN VORAUSSETZUNG UND
        EVTL. SUBSTITUTION VON EVAL-SUBST IN
        INKARNATIONSAUFRUFEN DURCH FKT-SYMBOL-IN")
 (SETQ VORAUSSETZUNGEN (FKT-SYMBOL-IN VORAUSSETZUNGEN
                                      (QUOTE VORAUSSETZUNGEN)))
(*COM* "TEST DER FUNKTIONEN IN PARAM-TYP-LISTE, INDIKATOR
        SYSTEM UND EVTL. SUBSTITUTION VON EVAL-SUBST IN
        INKARNATIONAUFRUFEN DURCH FKT-SYMBOL-IN")

 <PUT (QUOTE PARAM-TYP-LISTE)
```

```
      (QUOTE SYSTEM)
      (MAP2CAR <MAPCAR (GETP (QUOTE PARAM-TYP-LISTE)
                             (QUOTE SYSTEM))
                       (FUNCTION (LAMBDA (PARAM)
                         (COND
                           ((LISTP (CADDR PARAM))
                             (CADDR PARAM))
                             (CADDR PARAM))
                           (T NIL>
               (GETP (QUOTE PARAM-TYP-LISTE)
                     (QUOTE SYSTEM))
               (FUNCTION (LAMBDA (WERT PARAM)
                 (COND
                   <WERT (LIST (CAR PARAM)
                               (CADR PARAM)
                               (FKT-SYMBOL-IN WERT
                                 (QUOTE
                                  PARAMETERTYPEN:_SYSTEM>
                   (T PARAM>
(*COM* "TEST AUF ALGORITHMUS-WERT BEI
        PARALLEL-REKURSIVEN ALGORITHMEN;
        S.A. GLOBALE VARIABLEN")
 (COND
   (PRA <COND
          (ALGORITHMUS-WERT NIL)
          (T <SETQ ALGORITHMUS-WERT
               (REMOVE NIL
                 (MAPCAR RECHTESEITEN
                   (FUNCTION (LAMBDA (RS)
                     (COND
                       ((CDAAR (EKL RS))
                         NIL)
                       (T (CAAR (EKL RS>
              (SETQ ALGORITHMUS-WERT
                (REMOVE NIL
                  (MAPCAR ALGORITHMUS-WERT
                    (FUNCTION (NLAMBDA (RS)
                      (COND
                        ((MEMBER RS MAPL)
                          NIL)
                        (T RS>
        (COND
          ((CDR ALGORITHMUS-WERT)
            (PRINTL "***ERROR:" (CAR ALGNAME)
                    "DARF NICHT MEHRERE ALGORITHMUS-
                     WERTE HABEN, NAEMLICH:"
                     ALGORITHMUS-WERT)
            (SETQQ ABBRUCH T)
            (GO ABBRUCH))
          (<NOT (OR (MEMB (CAAR ALGORITHMUS-WERT)
                          ALGPARAM)
                    (MEMB (CAAR ALGORITHMUS-WERT)
                          (QUOTE (UNDEF NIL T FALSE TRUE>
            (PRINTL "***ERROR: ALGORITHMUS-WERT"
                     (CAR ALGORITHMUS-WERT)
                    "NICHT ERLAUBT")
```



```
                (SETQQ ABBRUCH T)
                (GO ABBRUCH))
              (T NIL)))
     (T NIL))
   (COND
     (UNGUELTIG (SETQ UNGUELTIG NIL)
                (GO NEUSCHLUESSEL))
     (T NIL))
(*COM* "TEST AUF DEFINIERTEN RELATIONSSATZ;
        EVTL. DEF. VON REL-SATZ STANDARD")
   (COND
     (RELATION NIL)
     (T (SETQ RELATION (LIST (QUOTE STANDARD)))
        (PRINTL "RELATION       = STANDARD")
        (PRINTL "RELATIONSSATZ =" STANDARD)))
(*COM* "TEST AUF KOMMENTARTEXT")
   (COND
     (KOMMENTAR NIL)
     (T (PRINTL "KOMMENTAR FEHLT !!!")))
ABBRUCH
   )
(*COM* "ZEICHENRUECKDEUTUNG")
(PROGN (SETSEPR (CHCON (QUOTE :))
                0)
       (SETBRK (CHCON (QUOTE ,;!))
                0)
       (SETBRK (QUOTE (125))
                1)
    (*COM* "ZEICHENBEDEUTUNG: WIEDER NORMAL")
    (COND
      (ABBRUCH NIL)
      (T T>>
```

5.2.1.2.2. Unterfunktionen von RALGSPEC

<RATOM-NE <NLAMBDA (ZEICHEN) ...>>

Zweck: Einlesen von Atomen bis zum Atom "Zeichen".
Aufruf: durch RALGSPEC
Param.: ZEICHEN gibt das Atom an, bis zu dem gelesen werden soll.
Ergebnis: Neue Leseposition ist das auf "Zeichen" folgende Atom.
Wert: Liste der gelesenen Atome ausschließlich "Zeichen".

```
<RATOM-NE
  <NLAMBDA (ZEICHEN)
    (PROG (ATOM (GELESEN (CONS)))
      LOOP(SETQ ATOM (RATOM))
          (COND
            ((NEQ ATOM ZEICHEN)
              (NCONC1 GELESEN ATOM)
              (GO LOOP))
            (T (RETURN (CDR GELESEN>>
```

<PARAMETER-UEBERPRUEFUNG <LAMBDA (...) ... >>

Zweck: Überprüfung der unter den Schlüsselwörtern "Parameter-Typen" und "Algorithmus" (ohne Produktionsregelmenge) spezifizierten Parameterangaben auf Übereinstimmung und Zulässigkeit.

Aufruf: durch RALGSPEC

Ergebnis: Falls Fehler erkannt werden, erfolgt eine entsprechende Fehlermeldung und der Implementierungslauf wird abgebrochen.

Wert: T, im Fehlerfall NIL.

Bem.: PARAMETER-UEBERPRUEFUNG arbeitet mit Listen der z.Zt. von ADTCOMP implementierten sowie der von INTERLISP zur Verfügung gestellten Datentypen. Werden neue Datentypen eingeführt, so müssen diese Listen im LISP-Code der Funktion entsprechend modifiziert werden.

```
<PARAMETER-UEBERPRUEFUNG
  <LAMBDA (PARAM-DCL-LISTE)
  (* "PARAMETER-UEBERPRUEFUNG RUFT FOLGENDE FUNKTIONEN AUF:
      BOOL-INTERN, DATENTYPP, PRINTL")
    <SETQ PARAM-DCL-LISTE
          (MAPCAR
            (APPEND (GETP (QUOTE PARAM-TYP-LISTE
                          (QUOTE BENUTZER))
                    (GETP (QUOTE PARAM-TYP-LISTE)
                          (QUOTE SYSTEM)))
            (FUNCTION (LAMBDA (PARAM)
                        (CAR PARAM>
    (* "GLEICHE ANZAHL VON DEKLARIERTEN
        UND AUFRUF-PARAMETERN ?")
    (COND
      ((EQ (LENGTH PARAM-DCL-LISTE)
           (LENGTH ALGPARAM))
    (* "DEKLARIERTE UND AUFRUFPARAMETER GLEICH ?")
       (COND
         (<AND <EVERY PARAM-DCL-LISTE
                      (FUNCTION (LAMBDA (PARAM)
                        (COND
                          ((MEMB PARAM ALGPARAM)
                            T)
                          (T (PRINTL "***ERROR:
```



```
                                    DEKLARIERTER PARAMETER"
                                    PARAM "NICHT ALS
                                    AUFRUF-PARAMETER
                                    VERWENDET")
                                   (ABBRUCH>
                  (EVERY ALGPARAM
                         (FUNCTION (LAMBDA (PARAM)
                           (COND
                             ((MEMB PARAM PARAM-DCL-LISTE)
                               T)
                             (T (PRINTL "***ERROR:
                                 AUFRUF-PARAMETER"
                                 PARAM "NICHT
                                 DEKLARIERT")
                                (ABBRUCH>
          T)
        (T NIL)))
    (T (PRINTL "***ERROR: UNGLEICHE ANZAHL VON
                DEKLARIERTEN PARAMETERN UND
                AUFRUF-PARAMETERN")
       (ABBRUCH)))
  (* "DATENTYP-UEBERPRUEFUNG FUER BENUTZER-PARAMETER")
<MAPC (GETP (QUOTE PARAM-TYP-LISTE)
            (QUOTE BENUTZER))
      (FUNCTION (LAMBDA (PARAM)
          (SELECTQ (CDR PARAM)
                   ((LISPATOM LISPZELLE INTEGER REAL
                     LITATOM BOOL ARRAY STRING)
                    T)
                   (COND
                     ((DATENTYPP (CDR PARAM)
                                 (QUOTE YES))
                       T)
                     (T (PRINTL "DATENTYP" (CADR PARAM)
                                "KANN VON ALGCOMP NOCH
                                 NICHT BEARBEITET WERDEN")
                        (ABBRUCH>
(* "DATENTYP - WERT UEBERPRUEFUNG FUER SYSTEM-PARAMETER")
(MAPC (GETP (QUOTE PARAM-TYP-LISTE)
            (QUOTE SYSTEM))
      (FUNCTION (LAMBDA (PARAM)
          (SELECTQ (CADR PARAM)
                   <LISPATOM
          (* "LISPATOM := LITATOM ODER INTEGER")
                   (COND
                     ((OR (TYPEP (CADDR PARAM)
                                 30)
                          (TYPEP (CADDR PARAM)
                                 22)
                          (TYPEP (CADDR PARAM)
                                 23))
                       T)
                     (T (PRINTL "***ERROR: PARAMETER"
                         (CAR PARAM) "ALS TYP LISPATOM
                         VEREINBART, WERT ABER"
                        (CADDR PARAM))
```

```
        (ABBRUCH>
<LISPZELLE (COND
  ((TYPEP (CADDR PARAM)
          21)
    T)
  (T (PRINTL "***ERROR: PARAMETER"
      (CAR PARAM) "ALS TYP LISP-ZELLE
       VEREINBART, WERT ABER"
      (CADDR PARAM))
     (ABBRUCH>
<INTEGER (COND
  ((<OR (TYPEP (CADDR PARAM)
               22)
        (TYPEP (CADDR PARAM)
               23)
        (AND (EQ (CADDR PARAM)
                 (QUOTE UNDEF))
             (COND
               ((EQ RELATION
                    (QUOTE
                     UNDEF-KL-INTEGER)
                T)
               (T (PRINTL (CAR PARAM)
                   "ALS TYP INTEGER
                   VEREINBART. WERT
                   IST UNDEF, ABER
                   NICHT UNDEF-KL-
                   INTEGER ALS
                   RELATION DEKLARIERT")
                  (ABBRUCH>
    T)
   (T (*COM* "***ERROR: PARAMETER"
             (CAR PARAM) "ALS TYP
             INTEGER VEREINBART,
             WERT ABER:"
             (CADDR PARAM))
      (*COM* NIL>
 <REAL (COND
         ((OR (TYPEP (CADDR PARAM)
                     26)
              (TYPEP (CADDR PARAM)
                     27))
           T)
         (T (PRINTL "***ERROR: PARAMETER"
             (CAR PARAM) "ALS TYP REAL
              VEREINBART, WERT ABER
             (CADDR PARAM))
            (ABBRUCH>
  <LITATOM (COND
            ((TYPEP (CADDR PARAM)
                    30)
              T)
            (T (PRINTL "***ERROR:
                PARAMETER" (CAR PARAM)
                "ALS TYP LITATOM
                 VEREINBART, WERT ABER:"
```



```
                (CADDR PARAM))
               (ABBRUCH>
<BOOL (COND
        ((OR (EQ (CADDR PARAM)
                 (QUOTE TRUE))
             (EQ (CADDR PARAM)
                 (QUOTE FALSE)))
          (BOOL-INTERN (CAR PARAM)) T)
        (T (PRINTL "***ERROR: PARAMETER"
            (CAR PARAM) "ALS TYP BOOL
             VEREINBART, WERT ABER:"
            (CADDR PARAM))
           (ABBRUCH>
<ARRAY (COND
         ((TYPEP (CADDR PARAM)
                 31)
          T)
         (T (PRINTL "***ERROR: PARAMETER"
             (CAR PARAM) "ALS TYP ARRAY
              VEREINBART, WERT ABER:"
             (CADDR PARAM))
            (ABBRUCH>
<STRING (COND
          ((TYPEP (CADDR PARAM)
                  35)
           T)
          (T (PRINTL "***ERROR:
              PARAMETER" (CAR PARAM)
              "ALS TYP STRING
              VEREINBART, WERT ABER:"
             (CADDR PARAM))
            (ABBRUCH>
(COND
  (<(DATENTYPP (CADR PARAM)
               (QUOTE YES))
    (COND
      (<AND (EQ (CAR PARAM)
                (CADR (CADDR PARAM)))
            (EQ (CADR PARAM)
                (CADDR (CADDR PARAM)))
            (EQ (QUOTE INIT)
                (CAR (CADDR PARAM>
       T)
      (T (PRINTL "***ERROR: PARAMETER"
          (CAR PARAM) "ALS TYP"
          (CADR PARAM)
          "VEREINBART, WERT ABER"
          (CADDR PARAM)
          "STATT INITIALISIERUNG ( INIT"
          (CAR PARAM)
          (CADR PARAM) ")")
         (ABBRUCH>
  (T (PRINTL "DATENTYP" (CADR PARAM)
      "KANN VON ALG-COMP NOCH NICHT
       BEARBEITET WERDEN")
     (ABBRUCH>>
```

<BOOL-INTERN <LAMBDA (PARAM-NAME) ... >>

Zweck: Konvertierung von Wahrheitswerten in interne Darstellung: Die in der Spezifikation unter den Schlüsselwort "Parameter-Typen: System" evtl. angegebenen Wahrheitswerte TRUE und FALSE werden in T und NIL überführt.
Aufruf: durch PARAMETER-UEBERPRUEFUNG
Param.: Name eines zu konvertierenden boolschen Parameters.
Ergebnis: In der Property-List der globalen Variable PARAM-TYP-LISTE wird unter dem Indikator "System" die Konvertierung für den Parameter PARAM-NAME vorgenommen.
Wert: Die so veränderte Property-List.

```
<BOOL-INTERN
  <LAMBDA (PARAM-NAME)
    (PUT (QUOTE PARAM-TYP-LISTE)
         (QUOTE SYSTEM)
         (MAPCAR (GETP (QUOTE PARAM-TYP-LISTE)
                       (QUOTE SYSTEM))
                 (FUNCTION (LAMBDA (PARAM)
                     (COND
                       <(EQ PARAM-NAME (CAR PARAM))
                         (COND
                           ((EQ (CADDR PARAM)
                                (QUOTE TRUE))
                            (LIST (CAR PARAM)
                                  (CADR PARAM)
                                  T))
                           (T (LIST (CAR PARAM)
                                    (CADR PARAM)
                                    NIL>
                       (T PARAM>>
```

<DATENTYPP <LAMBDA (TYP PRINT? ...) ... >>

Zweck: DATENTYPP ist eine Prädikat-Funktion, die prüft, ob ein Datentyp zulässig ist. Ein Datentyp ist zulässig, wenn es sich entweder um einen in INTERLISP schon definierten Datentyp handelt, oder um einen mit ADTCOMP implementierten. Insbesondere werden auch parametrisierte Datentypen auf zulässige Parameter überprüft.
Aufruf: durch PARAMETER-UEBERPRUEFUNG und LISP-PHASE (siehe 5.2.1.4.)
Param.: TYP gibt den Namen des zu überprüfenden Datentyps an;
PRINT?=YES bewirkt das Ausdrucken von Meldungen, sonst werden diese unterdrückt.

Wert: T, falls TYP zulässig, NIL sonst.
Bem.: DATENTYPP arbeitet mit Listen der z.Zt. von ADTCOMP implementierten sowie der von INTERLISP zur Verfügung gestellten Datentypen. Werden neue Datentypen eingeführt, so müssen diese Listen im LISP-Code der Funktion entsprechend modifiziert werden.

```
<DATENTYPP
  <LAMBDA (TYP PRINT? PARAM PARAMZAEHLER DT?)
    (SETQ PARAMZAEHLER 0)
    (SETQ PARAM (CONS))
    <MAPC (UNPACK TYP)
          (FUNCTION (LAMBDA (LETTER)
            (COND
              <(EQ LETTER (QUOTE -))
                (COND
                  <(EQ PARAMZAEHLER 0)
                    (SELECTQ (PACK (CDR PARAM))
                       ((BINAERERBAUM EINSEITIGELISTE
                         FLEXREIHE FLEXVERBUND
                         GRENZKELLER KELLER MENGE
                         OFFENERKELLER OFFENELISTE
                         REIHE SCHLANGE SCHLUESSELLISTE
                         VERBUND ZWEISEITIGELISTE)
                         (SETQ PARAMZAEHLER
                               (ADD1 PARAMZAEHLER))
                         (SETQ PARAM (CONS))
                       (PROGN (COND
                                ((EQ PRINT? (QUOTE YES))
                                  (PRINTL "HAUPTTYP"
                                   (PACK (CDR PARAM))
                                   "UNZULAESSIG"))
                                T NIL)
                              (SETQ PARAMZAEHLER
                                    (ADD1 PARAMZAEHLER))
                              (SETQ PARAM (CONS))
                              (SETQQ DT? NO>
                  (T (SELECTQ (PACK (CDR PARAM))
                       ((LISPATOM LISPZELLE INTEGER REAL
                         LITATOM BOOL ARRAY STRING
                         BINAERERBAUM EINSEITIGELISTE
                         FLEXREIHE FLEXVERBUND
                         GRENZKELLER KELLER MENGE
                         OFFENERKELLER OFFENELISTE
                         REIHE SCHLANGE SCHLUESSELLISTE
                         VERBUND ZWEISEITIGELISTE)
                         (SETQ PARAMZAEHLER
                               (ADD1 PARAMZAEHLER))
                         (SETQ PARAM (CONS)))
                       (PROGN (COND
                                ((EQ PRINT? (QUOTE YES))
                                  (PRINTL (PACK (LIST
                                   (QUOTE PARAMETERTYP-)
                                    PARAMZAEHLER))
                                   (PACK (CDR PARAM))
                                   "UNZULAESSIG"))
```

```
                              (T NIL))
                             (SETQ PARAMZAEHLER
                                   (ADD1 PARAMZAEHLER))
                             (SETQ PARAM (CONS))
                             (SETQQ DT? NO>
                (T (SETQ PARAM (NCONC1 PARAM LETTER>
  <COND
    <(EQ PARAMZAEHLER 0)
      (SELECTQ (PACK (CDR PARAM))
         ((BINAERERBAUM EINSEITIGELISTE FLEXREIHE
           FLEXVERBUND GRENZKELLER KELLER
           MENGE OFFENERKELLER OFFENELISTE
           REIHE SCHLANGE SCHLUESSELLISTE
           VERBUND ZWEISEITIGELISTE)
           T)
         (PROGN (COND
                  ((EQ PRINT? (QUOTE YES))
                    (PRINTL "HAUPTTYP"
                            (PACK (CDR PARAM))
                            "UNZULAESSIG")
                  (T NIL))
                 (SETQQ DT? NO>
     (T (SELECTQ (PACK (CDR PARAM))
          ((LISPATOM LISPZELLE INTEGER REAL
            LITATOM BOOL ARRAY STRING
            BINAERERBAUM EINSEITIGELISTE
            FLEXREIHE FLEXVERBUND GRENZKELLER
            KELLER MENGE OFFENERKELLER
            OFFENELISTE REIHE SCHLANGE
            SCHLUESSELLISTE VERBUND ZWEISEITIGELISTE)
            T)
          (PROGN (COND
                   ((EQ PRINT? (QUOTE YES))
                     (PRINTL (PACK (LIST
                                    (QUOTE PARAMETERTYP-)
                                     PARAMZAEHLER))
                             (PACK (CDR PARAM))
                             "UNZULAESSIG"))
                   (T NIL))
                  (SETQQ DT? NO>
      (COND
        (DT? NIL)
        (T T>>
```

■

<FKT-SYMBOL-IN <LAMBDA (LISTE ORT ...) ... >>

Zweck: Voraussetzungen und die Wertzuweisung an System-Parameter können auch als ausführbare LISP-Formen spezifiziert werden. FKT-SYMBOL-IN überprüft solche LISP-Formen, insbesondere, ob die darin verwendeten Funktions-Symbole zulassig sind, d.h. ob es sich um INTERLISP-Funktionen, um selbstdefinierte Funktionen oder um bereits spezifizierte Hilfsalgorithmen oder

Hilfsfunktionen handelt.
Aufruf: durch RALGSPEC
Param.: LISTE ist die zu überprüfende LISP-Form;
ORT ist ein String, der für evtl. Fehlermeldungen benutzt wird um den Fehler zu lokalisieren.
Ergebnis: im Fehlerfall erfolgt eine Fehlermeldung und der Implementierungslauf wird abgebrochen. Sonst bleibt LISTE in der Regel unverändert.
Treten jedoch in LISTE Operations-Aufrufe selbstdefinierter Datentypen auf, so werden diese in ein EVAL-SUBST Konstrukt eingebettet, damit zur Ausführungszeit des implementierten Algorithmus an die Stelle des formalen Inkarnations-Namens des Datentyps der Name der aktuellen Inkarnation substituiert werden kann.
Wert: ursprüngliche oder gemäß Ergebnis veränderte LISTE

```
<FKT-SYMBOL-IN
  <LAMBDA (LISTE ORT FIRST? DT)
    (SETQQ FIRST? T)
    (SETQ DT)
    <SETQ LISTE
          (MAPCAR LISTE
            (FUNCTION (LAMBDA (ELEM)
              (COND
                ((LISTP ELEM)
                  (SETQ FIRST?)
                  (FKT-SYMBOL-IN ELEM ORT))
                <FIRST? (COND
                          ((OR (EQ (CAR (UNPACK
                                          (GETD ELEM)))
                                   (QUOTE &))
                               (EQ ELEM (QUOTE INIT))
                               (EQ ELEM (QUOTE T))
                               (MEMB ELEM HILFSALGORITHMEN)
                               (MEMB ELEM HILFSFUNKTIONEN))
                            (SETQ FIRST?)
                            ELEM)
                          (<AND (MEMB ELEM ALGPARAM)
                                (DATENTYPP
                                  (CDR (ASSOC ELEM
                                              PARAM-TYP-LISTE>
                            (SETQ FIRST?)
                            (SETQ DT ELEM)
                            ELEM)
                          (T (PRINTL "***ERROR:
                                FUNKTIONSSYMBOL" ELEM
                                "IN" ORT "UNBEKANNT")
                             (SETQQ ABBRUCH TRUE)
                             (GO ABBRUCH>
                T ELEM>
    (COND
      <DT (LIST (QUOTE EVAL)
                (LIST (QUOTE SUBST)
                      DT
                      (LIST (QUOTE QUOTE)
```

```
                    DT)
                (LIST (QUOTE QUOTE)
                      LISTE>
    (T LISTE>>
```

■

<ABBRUCH <LAMBDA NIL ... >>

Zweck: Schließen der Eingabedatei, falls innerhalb von RALGSPEC ein Fehler in der Spezifikation erkannt wird.
Aufruf: durch RALGSPEC und PARAMETER-UEBERPRUEFUNG
Ergebnis: EINGABEDATEI wird geschlossen und der Implementierungslauf beendet; weitere Eingaben über Bildschirm.
Wert: NIL

```
<ABBRUCH
  <LAMBDA NIL
    (CLOSEF EINGABEDATEI)
    (COND
      ((EQ ALTEINGABE T)
        (INFILE))
      (T (INFILE ALTEINGABE)))
    (SETQQ ABBRUCH TRUE)
    (GO ABBRUCH>>
```

■

<EKL <LAMBDA (L) ... >>

Zweck: Ersetzen aller zitierten Klammern, also "%(" und "%)", in einer Liste L durch echte Klammern "(" und ")".
Bem.: Die Funktion EKL wird zusammen mit ihren Unterfunktionen in Abschnitt 5.1.1.2.5, Seite 371 besprochen.

<REGELPARSER <LAMBDA NIL ... >>

Zweck: Syntax-Überprüfung der Produktionsregelmenge.
Die Funktion REGELPARSER stellt zusammen mit ihren Unterfunktionen, deren Namen mit den nichtterminalen Symbolen der Grammatik (siehe 2.4.2) übereinstimmen, ein erkennendes System für die Spezifikationssprache der Produktionsregelmenge dar.
Aufruf: durch RALGSPEC
Ergebnis: Die globalen Variablen Linkeseiten und Rechteseiten haben als Wert Listen mit Namen der linken bzw. rechten Seiten.
Nach Abarbeitung der Funktion REGELPARSER sind globale Variablen mit Namen L1,L2,...,Ln und

R1,R2,...Rn definiert. Wert von Li bzw. Ri ist die linke bzw. rechte Seite der i-ten Produktionsregel in interner Darstellung. (s.a. 5.2.1.1. Globale Variablen).
Wird ein Fehler entdeckt, erfolgt eine entsprechende Fehlermeldung und der Implementierungslauf wird abgebrochen.
Wert: T, im Fehlerfall NIL

```
<REGELPARSER
  <LAMBDA NIL
    (* "REGELPARSER RUFT FOLGENDE FUNKTIONEN AUF:
       REGELMENGE, PRINTL, *COM*")
    (SETQ LINKESEITEN (CONS))
    (SETQ RECHTESEITEN (CONS))
    (PRINTL "ALGORITHMUS:
             PARSING -AUFSPALTUNG DER PRODUKTIONSREGELN:")
    (PROG (REGELMENGE REGELZAEHLER ATOM AKT-SEITE)
          (SETQ REGELMENGE (COPY ALGORITHMUS))
          (SETQ REGELZAEHLER 1)
          (SETQ ATOM (CAR REGELMENGE))
          (COND
            ((REGELMENGEP)
              (SETQ LINKESEITEN (CDR LINKESEITEN))
              (SETQ RECHTESEITEN (CDR RECHTESEITEN))
              (PRINTL "LINKESEITEN: " LINKESEITEN)
              (PRINTL "RECHTESEITEN:" RECHTESEITEN)
              (RETURN T))
            (T (*COM* "END REGELPARSER NIL")
               (RETURN NIL)))
      PARSERSTOP
          (PRINTL "***ERROR: PARSING ENDED ABNORMALY")
          (RETURN NIL>>
```

■

Funktionsweise des Regelparsers

Jedem nicht-terminalen Symbol der Grammatik der Produktionsregelmenge (s.a. 2.4.2., Seite 27) entspricht genau eine Unterfunktion von REGELPARSER mit gleichem Namen, die dieses syntaktische Konstrukt erkennt. Terminale Symbole der Grammatik werden mit Hilfe der Funktionen RGLEICHP und RGLEICHP-NS erkannt. Variable Symbole, also Namen, Zahlen etc., werden mittels entsprechender INTERLISP-Prädikate einer Datentyp-Überprüfung unterzogen.
Beispiel: <litatom> wird durch die Funktion LITATOM erkannt, die ihrerseits das INTERLISP-Prädikat LITATOM aufruft. Allerdings sind bei <litatom> Sonderzeichen wie Klammern ausgeschlossen, die in INTERLISP als literale Atome gelten.
Die Produktionsregelmenge wird Atom für Atom mit Hilfe der Funktion NEXTATOM, die das gelesene Atom als Wert an das REGELPARSER-lokale Atom ATOM bindet gelesen. Insbesondere werden hierbei Klammern, die durch die INTERLISP-Lesefunktion RATOM in RATOM-NE in zitierte Klammern, also %(und %) umge-

wandelt worden sind, als Atome angesehen. Der Wert von ATOM ist also immer das zuletzt gelesene, aktuelle Atom.
Mit dem aktuellen Wert von ATOM wird der Syntax-Baum (siehe 2.4.2., Seite 32) durchlaufen, d.h. die entsprechende REGELPARSER-Unterfunktion wird aufgerufen und entscheidet, ob das aktuelle Atom gemäß der Grammatik an dieser Stelle zulässig ist oder nicht. Ist es zulässig, wird wieder NEXTATOM aufgerufen, d.h. Übergang zum nächsten Atom, andernfalls erfolgt eine Fehlermeldung und das Parsing wird beendet.
Es wird also jeweils ein Atom der spezifizierten Produktionsregelmenge betrachtet und direkt entschieden, ob es korrekt bezüglich der gegebenen Grammatik ist oder nicht.
Außerdem werden durch die Funktion REGELPARSER und ihre Unterfunktionen die globalen Variablen (s.a. 5.2.1.1.) Li und Ri sowie LINKESEITEN und RECHTESEITEN definiert:
Wert von LINKE- bzw. RECHTESEITEN ist nach dem Parsing jeweils eine Liste mit Atomen L1....Ln, R1....Rn, wobei n die Anzahl der Produktionsregeln angibt. An jedes dieser Atome ist die entsprechende linke bzw. rechte Seite einer Produktion in interner Darstellung gebunden. In interner Darstellung sind insbesondere die syntaktischen Konstrukte "linkeseite", "rechteseite", "vergleich" sowie jeder "parameter" einer Parameterbzw. Argumentliste nochmal geklammert, da der Pattern-Matcher nur solche Einheiten erkennen kann (näheres siehe 4.1.3.).

Unterfunktionen von REGELPARSER

Alle Unterfunktionen von REGELPARSER sind Prädikat-Funktionen, die entweder T, oder im Fehlerfall eine Fehlermeldung und NIL als Wert liefern.
Sie besitzen einen Anfangs- und Endkommentar, der durch vorherigen Aufruf der Funktionen COMMENT bzw. NOCOMMENT (siehe 5.4.) ein- bzw. ausgeschaltet werden kann.
Im übrigen entsprechen sie genau der in 2.4.2. angegebenen Grammatik, die die Spezifikationssprache der Produktionsregelmenge definiert, so daß nur Besonderheiten erläutert werden.
Nach Abarbeitung einer Regelseite wird diese unter Angabe ihres Namens in interner Darstellung ausgedruckt.

```
<REGELMENGEP
  <LAMBDA NIL
    (* "REGELMENGEP RUFT FOLGENDE FUNKTIONEN AUF:
        REGELP, RGLEICHP-NS, REGELMENGEP, PRINTL, *COM*")
    (*COM* "START REGELMENGEP")
    (COND
      (<AND (REGELP)
            (COND
              ((OR (AND (RGLEICHP-NS (QUOTE ;))
                        (REGELMENGEP))
                   (RGLEICHP-NS (QUOTE })))
               T)
              (T (PRINTL "***ERROR: ; ODER }
                          FEHLT IN REGELMENGE")
```



```
                 (PRINTL "GELESEN BIS:" ATOM
                         "IN REGEL" REGELZAEHLER)
                 (GO PARSERSTOP>
          T)
       ((*COM* "END REGELMENGE NIL")
        T NIL>>
```

REGELP: Bem.: Initialisiert die globale Variable Li als CONS-Zelle mit dem CAR-Verweis "%(" und dem CDR-Verweis NIL, ergänzt die globale Variable LINKESEITEN um Li und zählt die Produktionsregeln. Ferner wird eine linke Seite mit "%)" abgeschlossen und in interner Darstellung ausgedruckt.

```
<REGELP
  <LAMBDA NIL
    (* "REGELP RUFT FOLGENDE FUNKTIONEN AUF: LINKESEITEP,
        RECHTESEITEP, PRINTL, *COM*")
    (*COM* "START REGELP")
    (COND
      ((AND (PROGN (SETQ AKT-SEITE (PACK (LIST (QUOTE L)
                                               REGELZAEHLER)))
                   (SET AKT-SEITE (CONS (QUOTE %())))
                   (COND
                     ((LINKESEITEP)
                       (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                                              (QUOTE %))))
                       (SETQ LINKESEITEN
                             (NCONC1 LINKESEITEN AKT-SEITE))
                       (PRINTL AKT-SEITE ":" (EVAL AKT-SEITE))
                       T)
                     (T NIL)))
             (RECHTESEITEP))
       (SETQ REGELZAEHLER (ADD1 REGELZAEHLER))
       (*COM* "REGELP OK")
       T)
      (T (PARSERFEHLER "LINKE- ODER RECHTESEITE
                        IN REGEL FEHLERHAFT")
       NIL>>
```

```
<LINKESEITEP
  <LAMBDA NIL
    (* "LINKESEITE RUFT FOLGENDE FUNKTIONEN AUF:
        VERGLEICHP, RGLEICHP, LINKESEITEP,
        RGLEICHP-NS, PRINTL, *COM*")
    (*COM* "START LINKESEITEP")
    (COND
      ((<AND (VERGLEICHP)
             (COND
               ((OR (AND (OR (RGLEICHP (QUOTE UND))
                             (RGLEICHP (QUOTE ODER)))
                         (LINKESEITEP))
                    (RGLEICHP-NS (QUOTE ——%>))))
              T)
             (T (PARSERFEHLER "ODER,UND ODER ——>
                               FEHLT IN LINKESEITE")
        (*COM* "LINKESEITEP OK")
        T)
       (T (*COM* "END LINKESEITEP NIL")
          NIL>>
```

■

RECHTESEITEP: Bem.: Initialisiert die globale Variable Ri als CONS-Zelle mit dem CAR-Verweis "%(" und dem CDR-Verweis NIL und ergänzt die globale Variable RECHTESEITEN um Ri. Ferner wird eine rechte Seite mit "%)" abgeschlossen und ausgedruckt.

```
<RECHTESEITEP
  <LAMBDA NIL
    (* "RECHTESEITE RUFT FOLGENDE FUNKTIONEN AUF:
        FUNKTIONS-AUFRUFP, DT-LISP-OPP, PAR-REK-AUFRP,
        WERTP, PRINTL,*COM*")
    (*COM* "START RECHTESEITEP")
    (SETQ AKT-SEITE (PACK (LIST (QUOTE R)
                                REGELZAEHLER)))
    (SET AKT-SEITE (CONS (QUOTE %())))
    (COND
      ((OR (FUNKTIONS-AUFRUFP)
           (DT-LISP-OPP)
           (PAR-REK-AUFRP)
           (WERTP))
       (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                              (QUOTE %))))
       (SETQ RECHTESEITEN (NCONC1 RECHTESEITEN AKT-SEITE))
       (PRINTL AKT-SEITE ":" (EVAL AKT-SEITE))
       (*COM* "RECHTESEITEP OK")
       T)
      (T (PARSERFEHLER "RECHTESEITE NICHT PAR-REK-AUFRP
                        ODER WERT")
         FUNKTIONS-AUFRUF, DT-LISP-OP,
         PAR-REK-AUFRP ODER WERT")
         NIL>>
```

■

PAR-REK-AUFRUFP: Bem.: Setzt die globale Variable PRA auf den Wert T, falls ein parallel-rekursiver Aufruf des Algorithmus auf der rechten Seite einer Produktionsregel erkannt wird.

```
<PAR-REK-AUFRP
  <LAMBDA NIL
    (* "PAR-REK-AUFR RUFT FOLGENDE FUNKTIONEN AUF:
        RGLEICHP-NS, RGLEICHP, PARAMETERLISTEP,
        PRINTL, *COM*")
    (COND
      ((AND (RGLEICHP-NS (QUOTE PRA))
            (COND
              ((RGLEICHP-NS (QUOTE %())
                T)
              (T (PARSERFEHLER "( FEHLT IN PAR-REK-AUFR")))
            (PARAMETERLISTEP))
       (SETQQ PRA T)
       (*COM* "PAR-REK-AUFRP OK")
       T)
      (T NIL>>
```

■

```
<WERTP
  <LAMBDA NIL
    (* "WERTP RUFT FOLGENDE FUNKTIONEN AUF:
        ALGPARAMP, RELWERTP, PRINTL, *COM*")
    (*COM* "START WERTP")
    (COND
      ((OR (ALGPARAMP)
           (RELWERTP))
       (*COM* "WERTP OK")
       T)
      (T (PARSERFEHLER "WERT NICHT ALGPARAM ODER RELWERT")
         NIL>>
```

■

VERGLEICHP: Bem.: Schließt einen Vergleich in runde Klammern ein.

```
<VERGLEICHP
  <LAMBDA NIL
    (* "VERGLEICHP RUFT FOLGENDE FUNKTIONEN AUF:
        VGLSEITEP, RELP, PRINTL, *COM*")
    (*COM* "START VERGLEICHP")
    (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                           (QUOTE %())))
    (COND
      ((AND (VGLSEITEP)
            (RELP)
            (VGLSEITEP))
       (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
```

```
                                (QUOTE %))))
        (*COM* "VERGLEICHP OK")
        T)
      ((*COM* "END VERGLEICHP NIL")
        T NIL>>
```

■

```
<VGLSEITEP
  <LAMBDA NIL
    (* "VGLSEITEP RUFT FOLGENDE FUNKTIONEN AUF:
        DT-LISP-OPP, RELWERTP, ALGPARAMP,
        PRINTL, *COM*")
    (*COM* "START VGLSEITEP")
    (COND
      ((OR (DT-LISP-OPP)
           (FUNKTIONS-AUFRUF-NRP)
           (RELWERTP)
           (ALGPARAMP)
           (KWOTE-ATOMP)
           (INTEGERP))
       (*COM* "VGLSEITEP OK")
       T)
      (T (PARSERFEHLER "VGLSEITE NICHT
          DT-LISP-OP, RELWERT,
          FUNKTIONS-AUFRUF, ALGPARAM
          ODER INTEGER")
         NIL>>
```

■

RELP: Bem.: Konvertiert das Relationszeichen "=" der Spezifikationssprache in die interne Darstellung "==".

```
<RELP
  <LAMBDA NIL
    (* "RELP RUFT FOLGENDE FUNKTIONEN AUF:
        RGLEICHP, PRINTL, *COM*")
    (*COM* "START RELP")
    (COND
      ((OR (COND
             ((EQ ATOM (QUOTE =))
              (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                                     (QUOTE ==)))
              (*COM* "NEXATOM GELESEN")
              (SETQ REGELMENGE (CDR REGELMENGE))
              (SETQ ATOM (CAR REGELMENGE)
                T))
             (T NIL))
           (RGLEICHP (QUOTE /=))
           (RGLEICHP (QUOTE %<))
           (RGLEICHP (QUOTE %>))
           (RGLEICHP (QUOTE %<=))
           (RGLEICHP (QUOTE %>=)))
       (*COM* "RELP OK")
```



```
        T)
      (T (PARSERFEHLER "REL NICHT =, /=, <, >, <= ODER >=")
         NIL>>
```

■

```
<RELWERTP
  <LAMBDA NIL
    (* "RELWERTP RUFT FOLGENDE FUNKTIONEN AUF:
        RGLEICHP, PRINTL, *COM*")
    (*COM* "START RELWERTP")
    (COND
      ((OR (RGLEICHP (QUOTE UNDEF))
           (RGLEICHP (QUOTE TRUE))
           (RGLEICHP (QUOTE FALSE)))
       (*COM* "RELWERTP OK")
       T)
      (T (*COM* "END RELWERTP NIL")
         NIL>>
```

■

```
<KWOTE-ATOMP
  <LAMBDA NIL
    (* "KWOTE-ATOMP RUFT FOLGENDE FUNKTIONEN AUF:
        NEXTATOM, *COM* ")
    (*COM* "START KWOTE-ATOMP")
    (COND
      ((<AND (EQ (CAR (UNPACK ATOM))
                 (QUOTE %'))
             (ATOM (PACK (CDR (UNPACK ATOM>
       (NEXTATOM)
       (*COM* "KWOTE-ATOMP OK")
       T)
      (T (*COM* "END KWOTE-ATOMP NIL")
         NIL>>
```

■

FUNKTIONS-AUFRUF-NRP: Bem.: Schließt eine Parameterliste mit "%)" ab.

```
<FUNKTIONS-AUFRUF-NRP
  <LAMBDA NIL
    (* "FUNKTIONS-AUFRUF-NRP RUFT FOLGENDE FUNKTIONEN AUF:
       RGLEICHP, HILFSALGORITHMUSP,
       PARAMETERLISTEP, PRINTL,*COM*")
    (*COM* "START FUNKTIONS-AUFRUF-NRP")
    (COND
      ((AND (HILFSALGORITHMUSP)
            (COND
              ((RGLEICHP (QUOTE %())
               T)
              (T (PARSERFEHLER "( FEHLT IN
```

```
                FUNKTIONS-AUFRUF-NRP")))
          (COND
            ((PARAMETERLISTEP)
              (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                                     (QUOTE %))))
             T)
            (T NIL)))
        (*COM* "FUNKTIONS-AUFRUF-NRP OK")
        T)
      (T (*COM* "END FUNKTIONS-AUFRUF-NRP NIL")
       NIL>>
```

FUNKTIONS-AUFRUFP: Bem.: Schließt eine Parameterliste mit "%)" ab.

```
<FUNKTIONS-AUFRUFP
  <LAMBDA NIL
    (* "FUNKTIONS-AUFRUFP FUFT FOLGENDE FUNKTIONEN AUF:
      RGLEICHP, HILFSALGORITHMUSP,
      PARAMETERLISTEP, PRINTL, *COM*")
    (*COM* "START FUNKTIONS-AUFRUFP")
    (COND
      ((AND (OR (RGLEICHP (CAR ALGNAME))
                (HILFSALGORITHMUSP))
            (COND
              ((RGLEICHP (QUOTE %())
               T)
              (T (PARSERFEHLER "( FEHLT IN
                       FUNKTIONS-AUFRUF")))
            (COND
              ((PARAMETERLISTEP)
                (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                                       (QUOTE %))))
               T)
              (T NIL)))
       (*COM* "FUNKTIONS-AUFRUFP OK")
       T)
      (T (*COM* "END FUNKTIONS-AUFRUFP NIL")
       NIL>>
```

PARAMETERLISTEP: Bem.: Schließt jeden Parameter in runde Klammern ein.


```
<PARAMETERLISTEP
  <LAMBDA NIL
    (* "PARAMETERLISTEP RUFT FOLGENDE FUNKTIONEN AUF:
       PARAMETERP, RGLEICHP-NS, PARAMETERLISTEP,
       RGLEICHP, PRINTL, *COM*")
    (*COM* "START PARAMETERLISTEP")
    (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                           (QUOTE %())))
    (COND
      (<AND (PARAMETERP)
            (COND
              ((OR (RGLEICHP (QUOTE %)))
                   (AND (COND
                          ((RGLEICHP-NS (QUOTE ,))
                            (SET AKT-SEITE
                                 (NCONC1 (EVAL AKT-SEITE)
                                         (QUOTE %))))
                          (SET AKT-SEITE
                               (NCONC1 (EVAL AKT-SEITE)
                                       (QUOTE ,)))
                          T)
                        (T (*COM* "IN PARAMETERLISTE:
                            ES FOLGT NOCH EIN
                            PARAMETER")
                           NIL))
                        (PARAMETERLISTEP)))
               T)
              (T (PARSERFEHLER ", ODER )
                    FEHLT IN PARAMETERLISTE">
        T)
      (T (*COM* "END PARAMETERLISTEP NIL")
       NIL>>
```

```
<PARAMETERP
  <LAMBDA NIL
    (* "PARAMETERP RUFT FOLGENDE FUNKTIONEN AUF:
       DT-LISP-OPP, FUNKTIONS-AUFRUFP,
       ATOMP, PRINTL, *COM*")
    (*COM* "START PARAMETERP")
    (COND
      ((OR (DT-LISP-OPP)
           (FUNKTIONS-AUFRUFP)
           (ATOMP))
        (*COM* "PARAMETERP OK")
        T)
      (T (PARSERFEHLER "PARAMETER NICHT
            DT-LISP-OP, FUNKTIONS-AUFRUF
            ODER ATOM")
       NIL>>
```

```
<DT-LISP-OPP
  <LAMBDA NIL
    (* "DT-LISP-OPP RUFT FOLGENDE FUNKTIONEN AUF:
        RGLEICHP, DT-OPERATIONP, LISP-OPERATION,
        PRINTL, *COM*")
    (*COM* "START DT-LISP-OPP")
    (COND
      ((AND (RGLEICHP (QUOTE %())
            (COND
              ((OR (DT-OPERATIONP)
                   (LISP-OPERATIONP))
               T)
              (T (PARSERFEHLER "DT-LISP-OP NICHT
                    DT-OPERATION ODER
                    LISP-OPERATION">
       (*COM* "DT-LISP-OPP OK")
       T)
      (T (*COM* "END DT-LISP-OPP NIL")
         NIL>>
```

```
<DT-OPERATIONP
  <LAMBDA NIL
    (* "DT-OPERATIONP RUFT FOLGENDE FUNKTIONEN AUF:
        INK+OP, RGLEICHP, ARGUMENTLISTEP,
        PRINTL, *COM*")
    (*COM* "START DT-OPERATIONP")
    (COND
      ((AND (INK+OPP)
            (COND
              ((OR (RGLEICHP (QUOTE %)))
                   (ARGUMENTLISTEP))
               T)
              (T (PARSERFEHLER ") FEHLT IN DT-OPERATION">
       (*COM* "DT-OPERATIONP OK")
       T)
      (T (*COM* "END DT-OPERATIONP NIL")
         NIL>>
```

```
<LISP-OPERATIONP
  <LAMBDA NIL
    (* "LISP-OPERATIONP RUFT FOLGENDE FUNKTIONEN AUF:
        LISP-FKTP, RGLEICHP, ARGUMENTLISTEP,
        PRINTL, *COM*")
    (*COM* "START LISP-OPERATIONP")
    (COND
      ((AND (LISP-FKTP)
            (COND
              ((OR (RGLEICHP (QUOTE %)))
                   (ARGUMENTLISTEP))
               T)
```



```
              (T (PARSERFEHLER ") FEHLT IN LISP-OPERATION">
       (*COM* "LISP-OPERATIONP OK")
       T)
      (T (*COM* "END LISP-OPERATIONP NIL")
         NIL>>
```

```
<INK+OPP
  <LAMBDA NIL
    (* "INK+OPP RUFT FOLGENDE FUNKTIONEN AUF:
        ALGPARAMP, LITATOMP, PRINTL, *COM*")
    (*COM* "START INK+OPP")
    (COND
      ((AND (ALGPARAMP)
            (COND
              ((LITATOMP)
               T)
              (T (PARSERFEHLER "OPERATIONSNAME
                    FEHLT IN INK+OP">
       (*COM* "INK+OPP OK")
       T)
      (T (*COM* "END INK+OPP NIL")
         NIL>>
```

```
<ARGUMENTLISTEP
  <LAMBDA NIL
    (* "ARGUMENTLISTEP RUFT FOLGENDE FUNKTIONEN AUF:
        PARAMETERP, ARGUMENTLISTEP, RGLEICHP,
        PRINTL, *COM*")
    (*COM* "START ARGUMENTLISTEP")
    (COND
      ((AND (PARAMETERP)
            (OR (RGLEICHP (QUOTE %)))
                (ARGUMENTLISTEP)))
       (*COM* "ARGUMENTLISTEP OK")
       T)
      (T (*COM* "END ARGUMENTLISTEP NIL")
         NIL>>
```

<RGLEICHP <LAMBDA (ZEICHEN) ... >>

Bem.: RGLEICHP überprüft die Gleichheit des gelesenen Zeichens mit ZEICHEN und schreibt dieses bei Übereinstimmung durch Aufruf der Funktion NEXTATOM in die Ergebnisliste.

```
<RGLEICHP
  <LAMBDA (ZEICHEN)
    (* "RGLEICHP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START RGLEICHP")
    (COND
      ((EQ ATOM ZEICHEN)
       (NEXTATOM)
       (*COM* "RGLEICHP OK")
       T)
      (T NIL>>
```

■

<u><RGLEICHP-NS <LAMBDA (ZEICHEN) ... >></u>

Bem.: RGLEICHP-NS überprüft die Gleichheit des gelesenen Zeichens mit ZEICHEN, schreibt dieses jedoch <u>nicht</u> in die Ergebnisliste.

```
<RGLEICHP-NS
  <LAMBDA (ZEICHEN)
    (*COM* "START RGLEICHP-NS")
    (COND
      ((EQ ATOM ZEICHEN)
       (SETQ REGELMENGE (CDR REGELMENGE))
       (SETQ ATOM (CAR REGELMENGE))
       (*COM* "RGLEICHP-NS OK")
       T)
      (T NIL>>
```

■

```
<ALGPARAMP
  <LAMBDA NIL
    (* "ALGPARAMP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START ALGPARAMP")
    (COND
      ((MEMB ATOM ALGPARAM)
       (NEXTATOM)
       (*COM* "ALGPARAMP OK")
       T)
      (T NIL>>
```

■


```
<HILFSALGORITHMUSP
  <LAMBDA NIL
    (* "HILFSALGORITHMUSP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START HILFSALGORITHMUSP")
    (COND
      ((MEMB ATOM HILFSALGORITHMEN)
       (NEXTATOM)
       (*COM* "HILFSALGORITHMUSP OK")
       T)
      (T NIL>>
```

■

```
<LITATOMP
  <LAMBDA NIL
    (* "LITATOM RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START LITATOMP")
    (COND
      ((AND (NEQ ATOM (QUOTE %)))
            (LITATOM ATOM))
       (NEXTATOM)
       (*COM* "LITATOMP OK")
       T)
      (T NIL>>
```

■

<u>ATOMP</u>: Bem.: Alle Sonderzeichen wie Klammern, Relationszeichen usw. gelten im Sinne der zugrundeliegenden Grammatik nicht als Atome.

```
<ATOMP
  <LAMBDA NIL
    (* "ATOMP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START ATOMP")
    (COND
      ((AND (NEQ ATOM (QUOTE %())
             (NEQ ATOM (QUOTE %)))
             (NEQ ATOM (QUOTE =))
             (NEQ ATOM (QUOTE /=))
             (NEQ ATOM (QUOTE %<))
             (NEQ ATOM (QUOTE %>))
             (NEQ ATOM (QUOTE %<=))
             (NEQ ATOM (QUOTE %>=))
             (NEQ ATOM (QUOTE ——%>))
             (NEQ ATOM (QUOTE ,))
             (NEQ ATOM (QUOTE ;))
             (NEQ ATOM (QUOTE }))
             (ATOM ATOM))
       (NEXTATOM)
       (*COM* "ATOMP OK")
       T)
```

```
      (T NIL>>
```

■

```
<INTEGERP
  <LAMBDA NIL
    (* "INTEGERP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START INTEGERP")
    (COND
      ((OR (TYPEP ATOM 22)
           (TYPEP ATOM 23))
       (NEXTATOM)
       (*COM* "INTEGERP OK")
       T)
      (T NIL>>
```

■

```
<LISP-FKTP
  <LAMBDA NIL
    (* "LISP-FKTP RUFT FOLGENDE FUNKTIONEN AUF:
       NEXTATOM, *COM*")
    (*COM* "START LISP-FKT")
    (COND
      ((EQ (CAR (UNPACK (GETD ATOM)))
           (QUOTE &))
       (NEXTATOM)
       (*COM* "LISP-FKTP OK")
       T)
      (T NIL>>
```

■

NEXTATOM: Bem.: NEXATOM schreibt das aktuelle Atom in die Ergebnisliste und liest das nächste. NEXTATOM ist keine Prädikat-Funktion.
Endemeldung von NEXTATOM ist das neu gelesene Atom.

```
<NEXTATOM
  <LAMBDA NIL
    (*COM* "START NEXTATOM")
    (SET AKT-SEITE (NCONC1 (EVAL AKT-SEITE)
                           ATOM))
    (SETQ REGELMENGE (CDR REGELMENGE))
    (SETQ ATOM (CAR REGELMENGE))
    (*COM* ATOM>>
```

■

<PARSERFEHLER <NLAMBDA (FEHLERTEXT) ... >>

Bem.: PARSERFEHLER gibt FEHLERTEXT unter Angabe der Stel-

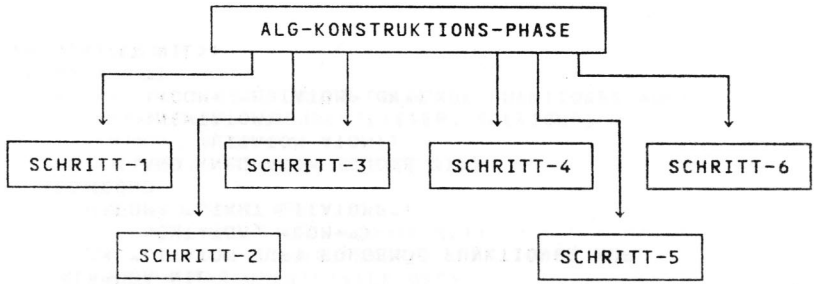
le, an der der Fehler erkannt wurde, als Fehlermeldung aus und bricht das Parsing ab.

```
<PARSERFEHLER
  <NLAMBDA (FEHLERTEXT)
    (PRINTL "***ERROR:" FEHLERTEXT)
    (PRINTL "GELESEN BIS:" ATOM "IN" AKT-SEITE)
    (GO PARSERSTOP>>
```

■

5.2.1.3. Funktionen der Algorithmus-Konstruktionsphase

Die Algorithmus-Konstruktionsphase erzeugt, wie in Abschnitt 4.1.2. beschrieben, aus den an Standardbezeichner gebundenen Eingabedaten einen dem Produktionssystem der Spezifikation entsprechenden COND-Ausdruck und einige Zusatzinformationen für die sich anschließende LISP-Phase.
Der zugrundeliegende Kontrollmechanismus ist das sukzessive Aufrufen der Funktionen SCHRITT-1 bis SCHRITT-6 durch die Steuerfunktion ALG-KONSTRUKTIONS-PHASE:



<ALG-KONSTRUKTION-PHASE <LAMBDA NIL ... >>

Zweck: Sukzessive Anwendung der SCHRITTi-Funktionen.
Aufruf: Durch IMPLEMENTIERE-ALG
Ergebnis: Nach Abarbeitung befindet sich in der Property-List des Atoms ALGNAME unter dem Indikator ALGORITHMUS das mit Hilfe der Regelbasen transformierte Produktionssystem in Form eines COND - Ausdrucks.
Wert: NIL.

```
<ALG-KONSTRUKTIONS-PHASE
 <LAMBDA  NIL
 (* "ALG-KONSTRUKTIONS-PHASE RUFT FOLGENDE FUNKTIONEN AUF:
```

```
    SCHRITT-1 SCHRITT-2 SCHRITT-3
    SCHRITT-4 SCHRITT-5 SCHRITT-6")
  (MAPC (QUOTE (1 2 3 4 5 6))
        (FUNCTION (LAMBDA (KONST-PHASE)
            (APPLY (PACK (LIST (QUOTE SCHRITT-)
                               KONST-PHASE)))
            (COND
              ((EQ GBC-FLAG (QUOTE NO-GBC))
               NIL)
              (T (GBC>>
```

Bemerk.: Je nach Angabe von GBC-FLAG in IMPLEMENTIERE-ALG wird ein garbage-collection durch explizites Anstoßen des Garbage-collectors durchgeführt (bei umfangreichen Spezifikationen von Vorteil).

■

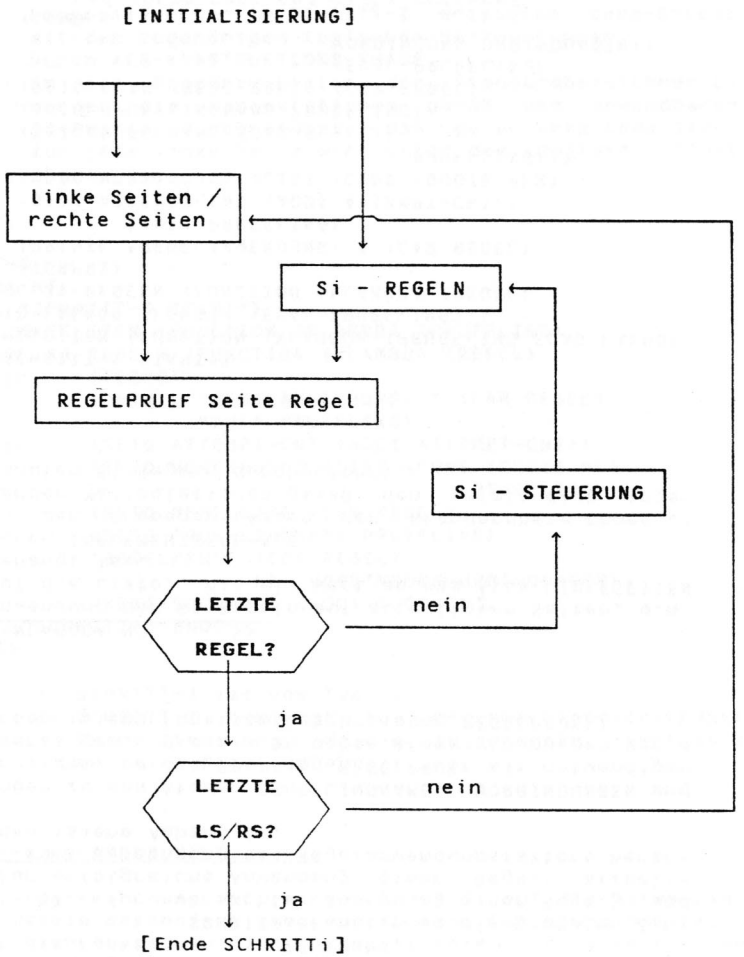
Die SCHRITTi-Funktionen realisieren ein Pattern-Matching mit Hilfe der Funktion REGELPRUEF des ADTCOMP - Subsystems. Die Verwendung von REGELPRUEF in ALGCOMP hatte hauptsächlich pragmatische Bedeutung, da auf ein bereits existierendes, leistungsfähiges PM-System zurückgegriffen werden konnte, welches gewisse Teilaufgaben der Algorithmuskonstruktion lösen konnte. Allerdings waren mit der Übernahme auch eine Reihe von Seiteneffekten verbunden, die - ohne ins Detail zu gehen - größtenteils darin bestanden, die etwas anders gelagerte Problematik in ALGCOMP an die Syntax und die Philosophie des ADTCOMP-Patternmatchings anzugleichen. In vielen Fällen waren dabei Umständlichkeiten in Kauf zu nehmen.

Die SCHRITTi - Funktionen zergliedern die Konstrukte der Produktionenmenge der Spezifikation geeignet, nehmen Manipulationen an den Datenbasen durch Initiierung der Regelanwendung vor, übersetzen "elementare" Konstrukte in ausführbaren LISP-Code und und bauen diese Code-Stücke geeignet zu einem Modul auf, der an die sich anschließende Funktion LISP-PHASE übergeben wird.

Die SCHRITTi - Funktionen haben neben der Kontrollaufgabe in der jeweiligen Stufe der Analyse darüberhinaus die Aufgabe, die ihnen jeweils zugeordneten Steuerfunktionen der Regelanwendung (Si-STEUERUNG) mit essentiellen Informationen zu versorgen bzw. selbst in die Agenda einzugreifen.

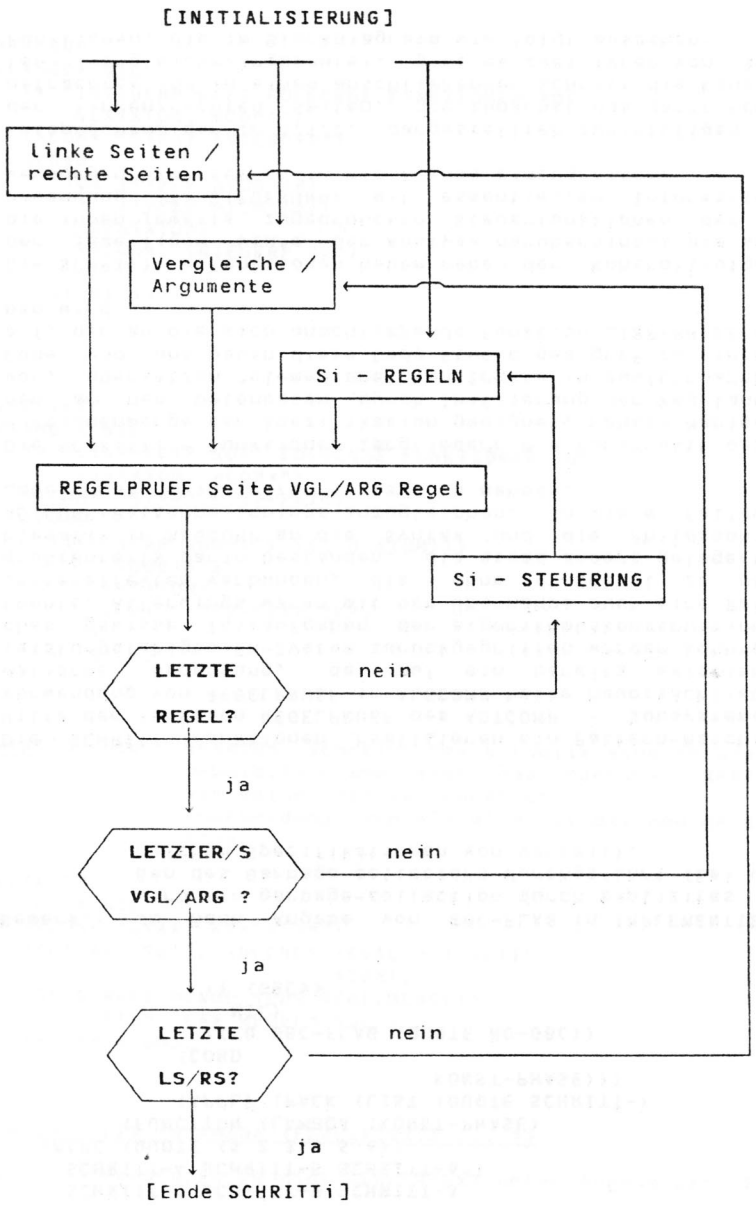
Entsprechend der in 4.1.2. dargestellten zweistufigen Analyse der linken/rechten Seiten, die zunächst das ganze Konstrukt betrachtet und in einem anschließenden Schritt die Konstituenten (Vergleiche/Argumente), gibt es zwei Typen von SCHRITT-Funktionen, die im Blockdiagramm wie folgt aussehen:

1) SCHRITTi, i=1,3,4,6 : zwei Schleifen



(Bei SCHRITT6 folgen noch zusätzliche Aktionen)

2) SCHRITTi, i=2,5 : drei Schleifen



In jeder der SCHRITTi-Funktionen liefert der Aufruf der Funktion REGELPRUEF neben den Seiteneffekten durch erfolgreiche Anwendung einer Regel einen Wahrheitswert, entsprechend einem erfolgreichen oder erfolglosen Versuch. Dieser Wahrheitswert wird an die (SCHRITTi)-lokale Variable FLAG übergeben. Der FLAG-Wert wird von der Si-STEUERUNG benutzt in der Entscheidung über die nächste auf die Datenbasis anzuwendende Regel. Eine ähnliche Bedeutung hat die (SCHRITTi)-lokale Variable LAENGE, deren Wert die Länge (Anzahl der LISP-Atome) des aktuellen Prüflings ist. Die genaue Beschreibung der Regelauswahlmechanismen geschieht in Abschnitt 4.1.3. und im erläuternden Text der Funktionen Si-STEUERUNG.
Außerdem werden in allen SCHRITTi-Funktionen die globalen Zähler ATTEMPT-CNT (für versuchte Anwendung einer Regel) und SUCC-CNT (für erfolgreiche Anwendung einer Regel) aktualisiert, die zur Berechnung des Regelanwendungsfaktors herangezogen werden (siehe Anhang 2).

Die Zuweisungen zu den Atomen PRUEFLINGNAME, VORBINDUNGEN und REGELNAME erfolgen, um die Funktion REGELPRUEF mit notwendigen Daten zu versorgen, die als an diese Atome gebundenen Werte erwartet werden (siehe Funktionsbeschreibung REGELPRUEF).

<SCHRITT-1 <NLAMBDA NIL ... >>
Zweck: Anwendung der S1-Regeln auf alle linken Seiten, d.h. auf die Liste, die als Wert an das Atom LINKESEITEN gehängt ist.
Aufruf: Durch IMPLEMENTIERE-ALG
Ergebnis: An den Property-Listen der Standardbezeichner Li werden Manipulationen gemäß den erfolgreich angewandten S1-Regeln durchgeführt.
Wert: NIL.

```
<SCHRITT-1
 <NLAMBDA NIL
  (PRINTL "SCHRITT-1 START")
  <MAPC LINKESEITEN (FUNCTION (NLAMBDA (PRUEFLING FLAG LAENGE)
        (SETQ LAENGE (LENGTH (EVAL PRUEFLING)))
        (MAPC S1-REGELN <FUNCTION (NLAMBDA (REGEL)
            (TERPRI)
            (PRINTL "REGELANWENDUNG: " (CAR REGEL)
                    "AUF" PRUEFLING)
            (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
            (SETQ VORBINDUNG (LIST (CONS (QUOTE *LS)
                                         PRUEFLING)))
            (SETQ REGELNAME (CAR REGEL))
            (SETQ PRUEFLINGNAME PRUEFLING)
            (SETQ FLAG (REGELPRUEF (CDR REGEL)
                                   (EVAL PRUEFLING)
                                   VORBINDUNG URBINDUNGEN))
            (COND
              (FLAG (SETQ SUCC-CNT (ADD1 SUCC-CNT>
        (FUNCTION S1-STEUERUNG>
    (PRINTL "SCHRITT-1 ENDE")
```

```
    (TERPRI>>
```

Bemerk.: SCHRITT-1 ist vom Typ 1.

■

<SCHRITT-2 <NLAMBDA NIL ... >>

Zweck: Anwendung der S2-Regeln auf die Vergleiche aller linken Seiten, wie sie durch SCHRITT-1 separiert wurden.
Aufruf: Durch ALG-KONSTRUKTIONS-PHASE.
Ergebnis: An den Property-Listen der Standardbezeichner Li werden unter den Indikatoren VGLi, VGLLIST und LOGOP Manipulationen gemäß den anwendbaren S2-Regeln durchgeführt.
Wert: NIL.

```
<SCHRITT-2
 <NLAMBDA NIL
  (PRINTL "SCHRITT-2 START")
  <MAPC LINKESEITEN (FUNCTION (NLAMBDA (PRUEFLING)
        (MAPC (CAR (GETP PRUEFLING (QUOTE VGLLIST)))
          (FUNCTION (NLAMBDA (VGL FLAG LAENGE)
            <SETQ LAENGE (LENGTH (CDDR (ASSOC (QUOTE EXPR)
                                              (GETP PRUEFLING VGL>
                  (MAPC S2-REGELN (FUNCTION <NLAMBDA (REGEL)
                  (TERPRI)
                  (PRINTL "REGELANWENDUNG: " (CAR REGEL)
                    "AUF" PRUEFLING ", VERGLEICH " VGL)
                  (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
                  (SETQ VORBINDUNG (LIST (CONS (QUOTE *LS)
                                               PRUEFLING)
                                         (CONS (QUOTE *VGL)
                                               VGL)))
                  (SETQ REGELNAME (CAR REGEL))
                  (SETQ PRUEFLINGNAME PRUEFLING)
                  (SETQ FLAG (REGELPRUEF (CDR REGEL)
                                         (CDDR (ASSOC (QUOTE EXPR)
                                                      (GETP PRUEFLING VGL)))
                                         VORBINDUNG URBINDUNGEN))
                  <COND
                    (FLAG (SETQ SUCC-CNT (ADD1 SUCC-CNT>
                (FUNCTION S2-STEUERUNG>
    (PRINTL "SCHRITT-2 ENDE")
    (TERPRI>>
```

Bemerk.: SCHRITT-2 ist vom Typ 2.

■

<SCHRITT-3 <NLAMBDA NIL ... > >

Zweck: Komposition der in SCHRITT-2 erzeugten Code-Stücke mit den zugehörigen logischen Verknüpfungen.
Aufruf: Durch ALG-KONSTRUKTIONS-PHASE.
Ergebnis: An den Property-Listen der Standardbezeichner Li werden die Manipulationen gemäß den anwendbaren S3-Regeln durchgeführt. Das generierte Code-Stück für jede linke Seite wird unter dem Indikator FINAL abgelegt.
Wert: NIL.

```
<SCHRITT-3
 <NLAMBDA NIL
 (PRINTL "SCHRITT-3 START")
 <MAPC LINKESEITEN (FUNCTION (NLAMBDA (PRUEFLING)
     (MAPC S3-REGELN (FUNCTION (NLAMBDA (REGEL)
               (TERPRI)
               (PRINTL "REGELANWENDUNG: " (CAR REGEL)
                       "AUF" PRUEFLING)
               (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
               (SETQ VORBINDUNG (LIST (CONS (QUOTE *LS)
                                            PRUEFLING)))
               (SETQ REGELNAME (CAR REGEL))
               (SETQ PRUEFLINGNAME PRUEFLING)
               (REGELPRUEF (CDR REGEL)
                           NIL VORBINDUNG URBINDUNGEN)
               (SETQ SUCC-CNT (ADD1 SUCC-CNT>
 (PRINTL "SCHRITT-3 ENDE")
 (TERPRI>>
```

Bemerk.: 1) SCHRITT-3 ist vom Typ 1.
2) SCHRITT-3 benötigt zum gegenwärtigen Zeitpunkt keine Steuerung, da die zugeordnete Regelbasis nur eine Regel umfaßt.

■

<SCHRITT-4 <NLAMBDA NIL ... > >

Zweck: Untersuchung der rechten Seite Ri auf die Notwendigkeit der Anfertigung von Kopien. Zerlegung der rechten Seite Ri in ihre Konstituenten, falls es sich um einen rekursiven Aufruf des Algorithmus handelt; andernfalls Transformation in LISP-Code.
Aufruf: Durch ALG-KONSTRUKTIONS-PHASE.
Ergebnis: Auf der Property-Liste von Ri werden unter den Indikatoren EVALUIERT, KOPIERTj, ARGj, ARGLIST Manipulationen gemäß den anwendbaren S4-Regeln durchgeführt und Informationen für den SCHRITT-5 über die Struktur der Regelmenge bereitgestellt.
Besteht eine rechte Seite Ri nicht aus einem rekursiven Aufruf des Algorithmus, so wird der LISP-Code dieser Seite unter dem Indikator FINAL an die

```
          Property-Liste gehängt.
Wert:     NIL.

<SCHRITT-4
 <NLAMBDA NIL
 (PRINTL "SCHRITT-4 START")
 <MAPC RECHTESEITEN (FUNCTION (NLAMBDA (PRUEFLING FLAG LAENGE)
         (SETQ LAENGE (LENGTH (EVAL PRUEFLING)))
         (MAPC S4-REGELN (FUNCTION <NLAMBDA (REGEL)
              (TERPRI)
              (PRINTL "REGELANWENDUNG: " (CAR REGEL)
                      "AUF" PRUEFLING)
              (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
              (SETQ VORBINDUNG (LIST (CONS (QUOTE *RS)
                                           PRUEFLING)))
              (SETQ PRUEFLINGNAME PRUEFLING)
              (SETQ REGELNAME (CAR REGEL))
              (SETQ FLAG (REGELPRUEF (CDR REGEL)
                                     (EVAL PRUEFLING)
                                     VORBINDUNG URBINDUNGEN))
              <COND
                (FLAG (SETQ SUCC-CNT (ADD1 SUCC-CNT>
         (FUNCTION S4-STEUERUNG>
  (PRINTL "SCHRITT-4 ENDE")
  (TERPRI>>
```

Bemerk.: SCHRITT-4 ist vom Typ 1.

■

<SCHRITT-5 <NLAMBDA NIL ... > >

Zweck: Anwendung der S5-Regeln auf die einzelnen Argumente der rechten Seite Ri. Dabei Entscheidung über Kopien, Substitution von Kopienbezeichnern und Erzeugung geeigneter Strukturen und Code-Stücke für derartig modifizierte Konstrukte. Transformation von Konstrukten der Spezifikationssprache in LISP-Code. Erzeugung von SETQ-Formen und Bereinigung des aktuellen Codes von Redundanzen.

Aufruf: Durch ALG-KONSTRUKTIONS-PHASE.

Ergebnis: Entweder in der dem Indikator ARGj zugeordneten Assoziationsliste unter dem Schlüssel "form" oder unter dem neu initialisierten Indikator FORM der Property-Liste von Ri wird temporärer Code abgelegt, in dem bereits die formalen Parameter des Algorithmus durch Kopien substituiert sind.

Wert: NIL.


```
<SCHRITT-5
  <NLAMBDA NIL
 (PRINTL "SCHRITT-5 START")
 <MAPC RECHTESEITEN
       (FUNCTION (NLAMBDA (PRUEFLING LISPNAMEN LAENGE)
    (PROG NIL
          (COND
            ((EQ (CAAR (GETP PRUEFLING (QUOTE EVALUIERT)))
                 (QUOTE PARALLEL-REKURSIV))
 Bem. 2       (SETQ MAPX (APPEND (LIST (CAR MAPX))
                                 (CAR (GETP PRUEFLING (QUOTE NEUE-NAMEN)))
                                 (CDR MAPX)))
              (RETURN))
            (T NIL))
          <MAPC (CAR (GETP PRUEFLING (QUOTE ARGLIST)))
                (FUNCTION (NLAMBDA (ARG FLAG)
                  <SETQ REGELN (COND
                     ((EQ (CAAR (GETP PRUEFLING
                                      (QUOTE EVALUIERT)))
       Bem. 3              (QUOTE FUN-ARG-REKURSION))
                      (CAR S5-REGELN))
                     (T (CADR S5-REGELN))>
                  (MAPC REGELN (FUNCTION <NLAMBDA (REGEL)
                      (TERPRI)
                        "AUF" PRUEFLING ", ARGUMENT" ARG)
                      (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
                      (SETQ VORBINDUNG (LIST (CONS (QUOTE *RS)
                                                   PRUEFLING)
                                             (CONS (QUOTE *ARG)
                                                   ARG)))
                      (SETQ REGELNAME (CAR REGEL))
                      (SETQ PRUEFLINGNAME PRUEFLING)
                      (SETQ FLAG (REGELPRUEF (CDR REGEL)
                                             (CDDR (ASSOC (QUOTE EXPR)
                                                   (GETP PRUEFLING ARG)))
                                             VORBINDUNG URBINDUNGEN))
                      <COND
                        (FLAG (SETQ SUCC-CNT (ADD1 SUCC-CNT>
                      <SETQ LAENGE
                            (LENGTH (CDDR (ASSOC (QUOTE EXPR)
                                                 (GETP PRUEFLING ARG>
          (FUNCTION S5-STEUERUNG>
  (TERPRI>>
```

Bemerk.: 1) SCHRITT-5 ist vom Typ 2.

2) Bei der Festlegung der zu bearbeitenden rechten Seiten ist ein Sonderfall zu beachten: evaluiert eine rechte Seite Ri "parallel-rekursiv", d.h. besteht sie aus mehr als einem rekursiven Aufruf des Algorithmus, so müssen diese einzeln bearbeitet werden. Deswegen wird die entsprechende Map-Liste MAPX geeignet modifiziert durch Einfügen der entsprechenden vom System erzeugten Regelnamen Ri-XX für jeden einzelnen dieser Aufrufe. (siehe auch 4.1.2. und Funktionsbeschreibung NEUE-RSEITEN).

3) Bei der Festlegung der Regelmenge muß der unter 2) angegebene Fall berücksichtigt werden: für einen rekursiven Aufruf, der parallel zu einem anderen ausgeführt werden soll, wird anderer Code benötigt als im nicht-parallelen Fall (siehe 4.1.1.), und somit kommt ein anderer Satz von S5-Regeln zur Anwendung.

■

<SCHRITT-6 <NLAMBDA NIL ... > >

Zweck: Aufbau des Codes für alle rechten Seiten Ri aus den Ergebnissen der Schritte 4 und 5. Insbesondere Erzeugung korrekter "DUPLIZIERE" - Formen. Komposition des terminalen COND-Ausdruckes. Berechnung von statistischen Informationen.

Aufruf: Durch ALG-KONSTRUKTIONS-PHASE.

Ergebnis: Zunächst werden nach Anwendung der zulässigen Regeln die endgültigen LISP-Codes der rechten Seiten Ri auf den Property-Listen der zugeordneten Standardbezeichner unter dem Indikator FINAL abgelegt. Danach werden die Werte aller FINAL-Indikatoren aller Standardbezeichner Li und Ri zusammen mit einem Fehler-Ausgang zu einem LISP-COND-Ausdruck komponiert und unter dem Indikator ALGORITHMUS in der Property-Liste des Atoms ALGNAME abgelegt. Abschließend werden die Property-Listen der Standardbezeichner gelöscht.

Wert: NIL.

```
<SCHRITT-6
 <NLAMBDA NIL
  (PRINTL "SCHRITT-6 START")
  <MAPC RECHTESEITEN
        (FUNCTION (NLAMBDA (PRUEFLING FLAG LAENGE)
        (SETQ LAENGE 1)
        (MAPC S6-REGELN <FUNCTION (NLAMBDA (REGEL)
                  (TERPRI)
                  (PRINTL "REGELANWENDUNG: " (CAR REGEL)
                          "AUF" PRUEFLING)
                  (SETQ ATTEMPT-CNT (ADD1 ATTEMPT-CNT))
                  (SETQ VORBINDUNG (LIST (CONS (QUOTE *RS)
                                               PRUEFLING)))
                  (SETQ REGELNAME (CAR REGEL))
                  (SETQ PRUEFLINGNAME PRUEFLING)
                  (SETQ FLAG (REGELPRUEF (CDR REGEL)
                                NIL VORBINDUNG URBINDUNGEN))
                  (COND
                    (FLAG (SETQ SUCC-CNT (ADD1 SUCC-CNT>
              (FUNCTION S6-STEUERUNG>
  (TERPRI)
  (PRINTL "SCHRITT 6 - ENDE")
     (TERPRI)
     (PRINTL "START KOMPOSITION")
     <PUT (CAR ALGNAME)
```



```
     (QUOTE ALGORITHMUS)
       (CONVERT-KWOTE (EKL (APPEND (QUOTE <NLAMBDA NIL>)
                                   (LIST (APPEND (LIST (QUOTE COND))
                                    <MAP2CAR LINKESEITEN RECHTESEITEN
                                      (FUNCTION (NLAMBDA (LS RS)
                                       (APPEND <CAR (EKL
                                                     (GETP LS (QUOTE FINAL>
                                        (CAR (EKL (GETP RS (QUOTE FINAL>
                                        (LIST (QUOTE (T (FEHLER 10>
  (TERPRI)
  (PRINTDEF (GETP (CAR ALGNAME)
                  (QUOTE ALGORITHMUS)))
  (TERPRI)
  (TERPRI)
  (PRINTL "VERWENDET WURDEN DIE REGELN")
  (TERPRI)
  (SETQ VERWENDET (SORT (CAR VERWENDET)))
  (PRINTL VERWENDET)
  (TERPRI)
  (PRINTL "REGELANWENDUNGSFAKTOR: "
          (FQUOTIENT SUCC-CNT ATTEMPT-CNT))
  (SETQ HILFSFUNKTIONEN (REMOVE (QUOTE NIL)
                                HILFSFUNKTIONEN))
  (PRINTL HILFSFUNKTIONEN)
  (TERPRI)
  (PRINTL "SCHRITT-6 ENDE")
  (TERPRI)
  (MAP2C LINKESEITEN RECHTESEITEN (FUNCTION (NLAMBDA (LS RS)
             (RPLACD RS)
             (RPLACD LS>>
```

Bemerk.: SCHRITT-6 ist vom Typ 1.

■

Die folgenden Funktionen sind Verwaltungs-, Hilfs- und Steuerfunktionen des Pattern-Matching-Systems und werden ausschließlich in der Algorithmus-Konstruktionsphase von CALL-Voraussetzungen innerhalb von Regeln aufgerufen. Durch diese Funktionen werden oft Manipulationen ausgelagert, die sonst in komplizierter Form innerhalb der Regeln ausgedrückt werden müßten.

Viele Funktionen modifizieren in Seiteneffekten die Bindeliste BINDE des Pattern-Match-Systems REGELPRUEF. In dieser Liste werden die sogenannten Urbindungen (Bindungen, die über einen ganzen Implementierungslauf Gültigkeit haben), und alle aktuellen Bindungen an "*"-Variable verwaltet. Die zur Anwendung kommenden Regeln greifen auf diese Liste zu (Genaueres in [EKT 79]).

<LISPNAMEN-VERMERK! <NLAMBDA (NAME) .... >>

Zweck: Verwaltungsfunktion für die unter den Bezeichner LISPNAMEN abgelegte Liste von LISP-Funktionsnamen.
Aufruf: Innerhalb der Voraussetzungen der Regel S5-2.
Param.: NAME: Der Name der zu vermerkenden Funktion.
Ergebnis: Der Funktionsname wird der Liste angehängt.
Wert: T.

```
<LISPNAMEN-VERMERK!
  <NLAMBDA (NAME)
    (SETQ LISPNAMEN (APPEND LISPNAMEN (LIST NAME)))
    T>>
```

Bemerk.: Die unter LISPNAMEN abgelegte Liste wird für jede rechte Seite Ri verwaltet.

■

<HILFSFKT-VERMERK! <NLAMBDA (REGEL NAME) ... >>

Zweck: Verwaltungsfunktion für die unter dem Bezeichner HILFSFUNKTIONEN abgelegte Liste der Namen der Hilfsfunktionen, die vom ALGCOMP-System in eine Implementierung eingebaut werden.
Aufruf: Innerhalb der Voraussetzungen der Regeln S2-9 und S6-1.
Param.: REGEL: Der Regelname.
NAME: Lok. Variable.
Ergebnis: Der in Abhängigkeit von der aktuellen Regel ermittelte Hilfsfunktionsname wird, sofern er sich nicht schon dort befindet, der Liste zugefügt, die unter HILFSFUNKTIONEN erreichbar ist.
Wert: T.

```
<HILFSFKT-VERMERK!
  <NLAMBDA (REGEL NAME)
    (COND
      <(EQ REGEL (QUOTE S2-9))
        (SETQ NAME (CDR (ASSOC (QUOTE *RELBIND)
                               BINDE)))
        (COND
          ((MEMB NAME HILFSFUNKTIONEN)
            NIL)
          (T (SETQ HILFSFUNKTIONEN
                   (APPEND HILFSFUNKTIONEN (LIST NAME>
      <(EQ REGEL (QUOTE S6-1))
        (MAPC (ZAHLENLISTE 5)
              (FUNCTION (NLAMBDA (ZAHL)
                (COND
                  ((CDR (ASSOC ZAHL BINDE))
                    (SETQQ NAME DUPLIZIERE)
                    (SETQ MAPX))
                  (T NIL))
                (COND
                  ((MEMB NAME HILFSFUNKTIONEN)
```



```
                    NIL)
                  (T (SETQ HILFSFUNKTIONEN
                       (APPEND HILFSFUNKTIONEN (LIST NAME>
        (T (FEHLER 19)))
    T>>
```

Bemerk.: Die Liste HILFSFUNKTIONEN wird von der Funktion LISP-PHASE weiterverwendet.

■

<LISP-DUPLIKATE <NLAMBDA (RS LISTE LISTE2 WERTE ERG) ... >>

Zweck: Erzeugt für eine rechte Seite eine Folge von DUPLIZIERE-Formen gemäß den in Schritt 5 erkannten Daten.
Aufruf: Innerhalb der Voraussetzungen der Regel S6-1.
Param.: RS: Name der rechten Seite.
LISTE: Lok. Variable. Bezeichnet die sukzessiv aufgebaute Liste der DUPLIZIERE-Formen.
LISTE2: Variable zur Zwischenablage von Daten.
WERTE: s.o.
ERG: Enthält die Ergebnisform.
Ergebnis: In der Bindeliste BINDE wird unter dem Variablenbezeichner *LISPDUPL die Folge der DUPLIZIERE-Formen abgelegt.
Wert: T.

```
<LISP-DUPLIKATE
  <NLAMBDA (RS LISTE LISTE2 WERTE ERG)
    <SETQ LISTE (EKL (GETP RS (QUOTE KOPIERT-LISP>
    <MAPCAR LISTE (FUNCTION (NLAMBDA (LISTE1)
1                 (SETQ LISTE1 (CDR LISTE1))
                  (SETQ LISTE2 (APPEND LISTE2 LISTE1>
    <SETQ ERG (MAPCAR ALGPARAM (FUNCTION (NLAMBDA (ALGP)
                    (SETQ WERTE)
                    <MAPCAR LISTE2 (FUNCTION (NLAMBDA (PAAR)
                      (COND
                        ((EQ (CAR PAAR)
2                           ALGP)
                         (SETQ WERTE (APPEND WERTE (CDR PAAR)))
                         (SETQ LISTE2 (REMOVE PAAR LISTE2)))
                        (T NIL>
                      (COND
                       (WERTE (SETQ WERTE (CONS ALGP WERTE)))
                        (T NIL>
    (SETQ ERG (REMOVE (QUOTE NIL)
                      ERG))
    <SETQ LISTE (MAPCAR ERG (FUNCTION (NLAMBDA (LISTE1)
                    (SETQ LISTE1 (APPEND (LIST (CAR LISTE1))
                                         (REMOVE (CAR LISTE1)
                                                 LISTE1)))
                    (COND
                        ((EQ (CDR LISTE1))
3                         NIL)
                        (T (LIST (QUOTE %()
```

```
                                (QUOTE DUPLIZIERE)
                                (CAR LISTE1)
                                (CDR LISTE1)
                                (QUOTE %)>
   (SETQ ERG)
   <MAPC LISTE (FUNCTION (NLAMBDA (LISTE1)
            (SETQ ERG (APPEND ERG LISTE1>
   <SETQ BINDE (APPEND BINDE (LIST (CONS (QUOTE *LISPDUPL)
                                         ERG>
   T>>
```

Bemerk.: Die Funktion arbeitet folgendermaßen:
In 1 werden alle Paare (FP K) (mit FP formaler Parameter, K Kopiebezeichner) in LISTE2 gesammelt.
In 2 werden für jeden formalen Parameter FP CONS-Zellen (FP . K1 K2 ... Kn) erzeugt, wobei die Ki die Kopienbezeichner sind, die für FP in den entsprechenden Regeln substituiert worden sind. Eine Liste dieser CONS-Zellen wird an ERG übergeben.
In 3 werden zunächst die CONS-Zellen in Listen umgewandelt und in eine DUPLIZIERE-Form eingebunden.

■

<STANDARD-BIND! <NLAMBDA (ZAHL ARG ARG-NR NUMMER A-ATOM) ... >>

Zweck: Wenn in Funktionsaufrufen, die innerhalb von Argumenten eines rekursiven Aufrufs auftreten, Parameter durch Kopien ersetzt werden sollen, erzeugt diese Funktion geeignete neue (Standard-) Bezeichner.
Aufruf: Innerhalb der Voraussetzungen der Regel S5-5.
Param.: ZAHL: Anzahl der zu erzeugenden Bezeichner.
ARG: Name des aktuellen Arguments.
ARG-NR: Aktuelle Argumentposition
NUMMER: Wert ist NIL, falls der aktuell betrachtete Parameter des Funktionsaufrufs numerisch ist.
A-ATOM: Bezeichnet das Atom *Aj der Bindeliste, für das eine Kopie erzeugt werden soll.
Ergebnis: In der Bindeliste BINDE werden unter den Variablenbezeichnern *STDj die Namen der erzeugten Kopienbezeichner (STDi-j) abgelegt.
Wert: T.

```
<STANDARD-BIND!
  <NLAMBDA (ZAHL ARG ARG-NR NUMMER A-ATOM)
    <MAPC (ZAHLENLISTE ZAHL)
          (FUNCTION (NLAMBDA (Z)
              <SETQ ARG-NR (CAR (LAST (UNPACK ARG>
              (SETQ A-ATOM (PACK (LIST (QUOTE *A)
                                       Z)))
              (SETQ NUMMER (COND
                  ((NUMBERP (CDR (ASSOC A-ATOM BINDE)))
                    (CDR (ASSOC A-ATOM BINDE)))
                  (T NIL)))
              (SETQ BINDE (APPEND BINDE
                 ┌  (LIST (CONS (PACK (LIST (QUOTE *STD)
```



```
                 1 └                Z))
                              (PACK (OR
                              (COND
                 2 ┌              (NUMMER (LIST NUMMER))
                                (T NIL))
                              (COND
                              <NOT (PARAMETER-REIHENFOLGE?
                 3 │             (LIST (CDR (ASSOC A-ATOM BINDE))
                                         ARG-NR)))
                               (LIST (CDR (ASSOC A-ATOM BINDE>
                              (T NIL))
                              (LIST (QUOTE STD)
                                    Z
                 4 │                 (QUOTE -)
                                   ARG-NR>
   T>>
```

Bemerk.: Die Funktion arbeitet folgendermaßen:
In 1 wird der Variablenbezeichner *STDj erzeugt, der als Wert
entweder
in 2 eine Zahl Z zugewiesen bekommt (falls *Aj die Zahl Z bezeichnet, braucht man keine Kopie für Z),
oder
in 3 den Wert von *Aj (falls *Aj eine formalen Parameter des Algorithmus bezeichnet, der nicht kopiert werden muß, da sein Auftreten in der rechten Seite vor einer Wertzuweisung an ihn liegt),
oder
in 4 einen geeignet (eindeutig) erzeugten Kopiebezeichner STDi-j (falls *Aj einen formalen Parameter bezeichnet, der kopiert werden muß).

■

<ARG-PARAM-BIND! <NLAMBDA (ARG PARAM VAR NR) ... >>

Zweck: Die Funktion erarbeitet den einer aktuellen Argumentposition zugeordneten formalen Parameter.
Aufruf: Innerhalb der Voraussetzungen der Regeln S5-D1, S5-1, S5-31, S5-32, S5-33, S5-4 und S5-5.
Param.: ARG: Der Name des aktuellen Arguments.
PARAM: In der Regel die Liste ALGPARAM der formalen Parameter des Algorithmus.
VAR: Lok. Variable, die den Namen des gesuchten formalen Parameters aufnimmt.
NR: Lok. Zahlvariable.
Ergebnis: In der Bindeliste BINDE wird unter dem Bezeichner *APB der Name des formalen Parameters abgelegt.
Wert: T.

```
<ARG-PARAM-BIND!
  <LAMBDA (ARG PARAM VAR NR)
          (* EDITED: " 6-JUL-81 16:00:38")
    <SETQ NR (CAR (LAST (UNPACK ARG>
    <COND
      ((EQ NR 1)
        NIL)
      (T (SETQ NR (SUB1 NR))
         (RPT NR (QUOTE (SETQ PARAM (CDR PARAM>
    (COND
      ((CAR PARAM)
        (SETQ VAR (CAR PARAM)))
      (T (FEHLER 13)))
    (SETQ BINDE (APPEND (LIST (CONS (QUOTE *APB)
                                    VAR))
                        BINDE))
    T>>
```

<u><FORM-BIND! <NLAMBDA (RS) ... >></u>

Zweck: Führt Datentransfer zwischen Indikatoren von Property-Listen durch.
Aufruf: Innerhalb der Voraussetzungen der Regel S6-23.
Param.: RS: Der Name der rechten Seite.
Ergebnis: In der Bindeliste BINDE wird unter dem Variablenbezeichner *FB der Wert des FORM-Indikators von Ri abgelegt.
Wert: T.

```
<FORM-BIND!
  <NLAMBDA (RS)
    <SETQ BINDE (APPEND BINDE (LIST (CONS (QUOTE *FB)
                                          (CAR (GETP RS (QUOTE FORM>
    T>>
```

<u><ENTKLAMMERUNG <NLAMBDA (Z) .... >></u>

Zweck: Syntaktische Veranderung von Ausdrücken, die an Variablenbezeichner gebunden sind.
Aufruf: Innerhalb der Voraussetzungen der Regel S2-2.
Param.: Z: Anzahl der zu bearbeitenden Ausdrucke.
Ergebnis: In der Bindeliste BINDE werden unter den Variablenbezeichnern *Aj die um die äußerste Klammerung verringerten Ausdrucke, die Werte der Variablen *AUj sind, abgelegt.
Wert: T.


```
<ENTKLAMMERUNG
  <NLAMBDA (Z L1)
    <MAPC (ZAHLENLISTE Z)
          (FUNCTION (NLAMBDA (ZAHL)
                    (SETQ BINDE (APPEND BINDE
                       (LIST (CONS (PACK (LIST (QUOTE A*)
                                               ZAHL))
                                   (CAR (EKL (CDR (ASSOC
                                   (PACK (LIST (QUOTE *AU)
                                                ZAHL))
                                   BINDE>
    T>>
```

<u><REL-BIND! <NLAMBDA (REL LISTE) ... >></u>

Zweck: Die Funktion verbindet ein Relationssymbol mit einen zugeordneten Relationsnamen.
Aufruf: Innerhalb der Voraussetzungen der Regel S2-9.
Param.: REL: Das Relationssymbol.
LISTE: Variable, die den vereinbarten Relationssatz als Wert besitzt.
Ergebnis: In der Bindeliste BINDE wird an den Variablenbezeichner *RELBIND der zutreffende Relationsname gebunden.
Wert: T.

```
<REL-BIND!
  <NLAMBDA (REL LISTE)
    (SETQ LISTE (CAAR RELATION))
    (COND
      <(ASSOC REL LISTE)
        (SETQ BINDE (APPEND BINDE
                            (LIST (CONS (QUOTE *RELBIND)
                            (CDR (ASSOC REL LISTE>
      (T (PRINTL "****FEHLER IN REL-BIND!!")
         (PRINTL "KEINEN FUNKTIONSNAMEN FUER DAS ATOM
                 " REL " IN " (CAR RELATION) " GEFUNDEN ")))
    T>>
```

<u><ARGNR-OK? <NLAMBDA (NR ARG) ... >></u>

Zweck: Die Funktion soll prüfen, ob eine aktuelle Argumentposition mit einer vorgegebenen Zahl übereinstimmt.
Aufruf: Innerhalb der Voraussetzungen der Regeln S5-1, S5-31, S5-32 und S5-33.
Param.: NR: Vorgegebene Zahl.
ARG: Aktuelles Argument.
Ergebnis: Keine Seiteneffekte.
Wert: T bei Übereinstimmung, sonst NIL.

```
<ARGNR-OK?
  <NLAMBDA (NR ARG)
    (COND
      ((<EQUAL NR (CAR (LAST (UNPACK ARG>
        T)
      (T NIL>>
```

<u>**<PARAMETER-REIHENFOLGE? <LAMBDA (LISTE LISTE1 LISTE2) ... >>**</u>

Zweck: Die Funktion überprüft, ob sich die in der Liste LISTE übergebenen formalen Parameter des Algorithmus in der gleichen Reihenfolge befinden wie in der Parameterliste der Spezifikation.
Aufruf: Innerhalb der Voraussetzungen der Regel S4-54.
Param.: LISTE: Liste der Form (FP1 Pos FP2 Pos ...), die die formalen Parameter FPj und ihre Position Pos in einem aktuellen Aufruf enthält.
LISTE1: Eine Liste der Form ((FP1 . Pos) ... (FPn . Pos)), die die formalen Parameter FPj und ihre Position in der Parameterliste der Spezifikation enthält.
LISTE2: Namen der formalen Parameter, die bezüglich der Parameterliste der Spezifikation in einer höheren Argumentposition stehen.
Ergebnis: Keine Seiteneffekte.
Wert: T, falls LISTE2 mindestens einen formalen Parameter enthält.
NIL sonst.

```
<PARAMETER-REIHENFOLGE?
  <LAMBDA (LISTE LISTE1 LISTE2)
    <MAP2C (ZAHLENLISTE (LENGTH ALGPARAM))
           ALGPARAM
           (FUNCTION (NLAMBDA (Z A)
                (SETQ LISTE1 (APPEND LISTE1 (LIST (CONS A Z>
    (MAPC LISTE <FUNCTION (NLAMBDA (ELEM)
                (COND
                  ((MEMB ELEM ALGPARAM)
                    (COND
                      <(GREATERP (CADR MAPX)
                                 (CDR (ASSOC ELEM LISTE1)))
                        (SETQ LISTE2 (APPEND LISTE2 (LIST ELEM>
                      (T NIL)))
                  (T NIL>
           (FUNCTION CDDR))
    (COND
      (LISTE2 T)
      (T NIL>>
```


<u>**<CONST-BIND! <NLAMBDA (ARG) ... >>**</u>

Zweck: Transformation von Konstanten.
Aufruf: Innerhalb der Voraussetzungen der Regeln S2-1, S2-1, S4-2 und S5-1.
Param.: ARG: Die zu transformierende Konstante.
Ergebnis: Folgende Fälle von Transformationen treten auf:

UNDEF ——> %(QUOTE UNDEF %)
FALSE ——> NIL oder NOFORM
TRUE ——> T
CONST ——> CONST

Wert: T.

```
<CONST-BIND!
  <NLAMBDA (ARG)
    (COND
      ((EQ ARG (QUOTE UNDEF))
        <SETQ BINDE (APPEND BINDE
                    (LIST (CONS (QUOTE *CONSTBIND)
                                (QUOTE (%( QUOTE UNDEF %)>
       T)
      ((EQ ARG (QUOTE FALSE))
        <SETQ BINDE (APPEND BINDE
                    (LIST (CONS (QUOTE *CONSTBIND)
                                (COND
                                  ((EQ REGELNAME (QUOTE S4-2))
                                    (QUOTE NOFORM))
                                  (T NIL>
       T)
      ((EQ ARG (QUOTE TRUE))
        <SETQ BINDE (APPEND BINDE
                    (LIST (CONS (QUOTE *CONSTBIND)
                                (QUOTE T>
       T)
      (ARG <SETQ BINDE (APPEND BINDE
                    (LIST (CONS (QUOTE *CONSTBIND)
                                                ARG>
         T)
      (T (FEHLER 15>>
```

<u>**<IN-BINDE! <NLAMBDA (ATOM IND) >>**</u>

Zweck: Transfer von Daten aus Property-Listen in die Bindeliste BINDE.
Aufruf: Innerhalb der Voraussetzungen der Regeln S2-9, S3-1 und S6-1.
Param.: ATOM: Bezeichnet eine linke oder rechte Seite.
IND: Bezeichnet einen Vergleichsindikator VGLj.
Ergebnis: Ist der IND ≠ NIL, werden aus der VGLj zugeordneten Assoziationsliste die Werte der Schlüssel "rel", "links" und "rechts" kopiert und in der Bindeliste BINDE an die Bezeichner *REL, *L und *R gehängt.
Ist IND = NIL, so werden entsprechend dem Wert von ATOM (Ri oder Li) über die zugeordneten Listen

ARGLIST oder VGLLIST alle bis dahin erzeugten Werte, die unter dem Schlüssel "form" in den ARGj/VGLj zugeordneten Assoziationslisten abgelegt wurden, nach BINDE kopiert. Dabei werden die Variablenbezeichner *Fj verwendet.

Wert: T

```
<IN-BINDE!
  <NLAMBDA (ATOM IND)
    <COND
     ┌ <IND (SETQ BINDE (APPEND BINDE
     │                 (LIST <CONS (QUOTE *REL)
     │                             (CDDR (ASSOC (QUOTE REL)
     │                                          (GETP ATOM IND (QUOTE REL>
1    │                       <CONS (QUOTE *L)
     │                        (CDDR (ASSOC (QUOTE LINKS)
     │                                     (GETP ATOM IND (QUOTE LINKS>
     │                       (CONS (QUOTE *R)
     │                             (CDDR (ASSOC (QUOTE RECHTS)
     │                                          (GETP ATOM IND (QUOTE RECHTS>
     └
      (T <SETQ ARGLISTE (COND
                ((EQ (CAR (UNPACK ATOM))
                     (QUOTE L))
                 (QUOTE VGLLIST))
                ((EQ (CAR (UNPACK ATOM))
                     (QUOTE R))
                 (QUOTE ARGLIST))
                (T (FEHLER 11>
         (SETQ BINDE (APPEND BINDE
           ┌          <MAP2CAR (CAR (GETP ATOM ARGLISTE))
           │           <ZAHLENLISTE (LENGTH (CAR (GETP ATOM ARGLISTE>
           │           (FUNCTION (NLAMBDA (VERGL NR)
           │                      (CONS (PACK (LIST (QUOTE *F)
       2   │                                        NR))
           │                            (CDDR (OR (ASSOC (QUOTE FORM)
           │                                             (GETP ATOM VERGL))
           │                                      (ASSOC (QUOTE EXPR)
           │                                             (GETP ATOM VERGL>
           └
           ┌  (MAPCAR (ZAHLENLISTE (DIFFERENCE
           │                        (LENGTH (CAR (GETP ATOM ARGLISTE)))
           │                                11))
       3   │                  (FUNCTION (NLAMBDA (ZAHL)
           │                            (CONS (PACK (LIST (QUOTE *F)
           │                                              ZAHL>
           └
    T>>
```

Bemerk.: Die Funktion arbeitet folgendermaßen:
In 1 wird der Fall abgearbeitet, daß IND ≠ NIL ist. Im anderen Fall wird in 2 für alle in ARGLIST oder VGLLIST aufgeführten Indikatoren der Variablenbezeichner *Fj mit dem "form"- oder "expr"-Wert der zugeordneten Assoziationsliste verknüpft.
In 3 werden weitere *Fj in BINDE mit dem Wert NIL initialisiert. Der Grund dafür ist die Struktur der Folgerungsmuster der IN-BINDE! aufrufenden Regeln, die aus Praktibilitätsgründen meist 9 *Fj-Terme

besitzen. Existieren jedoch nur i<9 *F-Terme in BINDE (weil nur i Vergleiche/Argumente auftreten), so führt der Auswertungsversuch von *Fj (j>i) innerhalb eines REGELPRUEF-Aufrufs zu einem Fehler und einer Erzeugung falschen Codes. (siehe auch Funktionsbeschreibung ZAHLENLISTE).

■

<IN-LISTE? <NLAMBDA (ATOM . LISTE) >>

Zweck: Überprüfung, ob sich ein Atom ATOM in der Liste LISTE befindet.
Aufruf: Innerhalb der Voraussetzungen der Regeln S4-2, S4-4, S4-53, S4-55, S5-2, S5-32 und S5-33.
Param.: ATOM: Zu suchendes literales Atom.
LISTE: Suchliste.
Ergebnis: Keine Seiteneffekte.
Wert: T, falls ATOM in LISTE.
NIL sonst.

```
<IN-LISTE?
  <NLAMBDA (ATOM . LISTE)
    (EQ ATOM (CAR (MEMB ATOM (EVAL (CAR LISTE>>
```

■

<EQ-KWOTE? <NLAMBDA (A B) >>

Zweck: Überprüfung, ob sich die übergebenen Argumente nur durch ein Kwotezeichen unterscheiden.
Aufruf: Innerhalb der Voraussetzungen der Regel S4-51.
Param.: A: Literales Atom.
B: Literales Atom.
Ergebnis: Keine Seiteneffekte.
Wert: T, falls sich A und B nur um ein Kwotezeichen unterscheiden.
NIL sonst.

```
<EQ-KWOTE?
  <NLAMBDA (A B)
    (SETQ A (COND
       <(EQ (CAR (UNPACK A))
            (QUOTE %'))
        (PACK (CDR (UNPACK A>
       (T A)))
    (SETQ B (COND
       <(EQ (CAR (UNPACK B))
            (QUOTE %'))
        (PACK (CDR (UNPACK B>
       (T B)))
    (EQ A B>>
```

■

<NO-KWOTE <NLAMBDA (NAME WERT) >>

Zweck: Entfernung von Kwotezeichen, mit denen Inkarnationen gekwotet sind.
Aufruf: Innerhalb der Voraussetzungen der Regel S6-1.
Param.: NAME: Name der Regel, in der die Funktion aufgerufen wird.
WERT: Lok. Variable.
Ergebnis: In der Bindeliste BINDE werden an Werten von Variablenbezeichnern, die gekwotete Inkarnationsbezeichner sind, die Kwotezeichen entfernt.
Wert: T.

```
<NO-KWOTE
 <NLAMBDA (NAME WERT)
  (COND
   <(EQ NAME (QUOTE S6-1))
     (MAPC (QUOTE (*P *P0 *P1 *P2 *P4))
      (FUNCTION (NLAMBDA (PAR)
          (COND
            ((CDR (ASSOC PAR BINDE))
              (SETQ WERT (CDR (ASSOC PAR BINDE)))
              (COND
                <(EQ (CAR (UNPACK WERT))
                     (QUOTE %'))
                  (SETQ BINDE (REMOVE (CONS PAR WERT)
                                      BINDE))
                  (SETQ BINDE (APPEND BINDE (LIST
                        (CONS PAR (PACK (CDR (UNPACK WERT>
                  (T NIL)))
            (T NIL>
      (T (FEHLER 44)))
    T>>
```

■

<INDEX-CHECK <NLAMBDA LISTE >>

Zweck: Überprüfung, ob von denen in LISTE übergebenen (Index-) Werten in der Bindeliste BINDE mindestens einer den Wert NIL aufweist.
Aufruf: Innerhalb der Voraussetzungen der Regeln S4-51, S4-52, S5-1, S5-31, S5-32 und S5-33.
Param.: LISTE: Liste von Indizes, die in BINDE gebunden sind.
Ergebnis: keine Seiteneffekte.
Wert: T, falls mindestens ein Index an NIL gebunden ist. NIL sonst.


```
<INDEX-CHECK
  <LAMBDA LISTE
    (PROG NIL
          <MAPC LISTE (FUNCTION (NLAMBDA (INDEX)
                (COND
                  ((CDR (ASSOC INDEX BINDE))
                    NIL)
                  (T (RETURN T>
          (RETURN NIL>>
```

■

<INKARANTION-? <NLAMBDA (INK ERG) ... >>

Zweck: Überprüfung, ob es sich bei dem an INK übergebenen Wert um einen Inkarantionsbezeichner handelt.
Aufruf: Innerhalb der Voraussetzungen der Regel S5-9.
Param.: INK: Bezeichnet ein literales Atom.
ERG: Lok. Variable zur Zwischenablage von Information.
Ergebnis: Falls der Wert von INK eine Inkarnation bezeichnet, wird in der Bindeliste BINDE unter dem Variablenbezeichner *INKBIND die Initialisierungsform für eine Inkarnation dieses Typs bereitgestellt. (Zur Initialisierung siehe .....).
Wert: T, falls Wert von INK eine Inkarnation eines ADTCOMP-Datentyps bezeichnet.
NIL sonst.

```
<INKARNATION-?
  <NLAMBDA (INK ERG)
    (SETQ ERG (CDR (ASSOC INK PARAM-TYP-LISTE)))
    (COND
      ((MEMB ERG (QUOTE (INTEGER BOOL NAT ARRAY LITATOM)))
        NIL)
      (ERG <SETQ BINDE (APPEND BINDE (LIST
                              (CONS (QUOTE *INKBIND)
                                    (LIST (QUOTE %()
                                          ERG
                                          (QUOTE DO)
                                          (QUOTE %)>
        T)
      (T NIL>>
```

■

<NEUE-RSEITEN <NLAMBDA (ZAHL RS) ... >>

Zweck: Wenn eine rechte Seite als parallel-rekursiv erkannt worden ist, muß jeder rekursive Aufruf einzeln bearbeitet werden. Die Funktion organisiert diese Arbeit.
Aufruf: Innerhalb der Voraussetzungen der Regel S4-3.
Param.: ZAHL: Anzahl der zu bearbeitenden rekursiven Aufrufe.
RS: Name der parallel-rekursiven rechten Seite.

Ergebnis: Es werden neue Standardbezeichner Ri-j generiert. In der Bindeliste BINDE werden die Variablenbezeichner *RSj (mit den Namen Ri-j) und *NEUE-NAMEN (mit der Liste der Ri-j) initialisiert. Jedem Ri-j wird als Wert ein rekursiver Aufruf der parallel-rekursiven rechten Seite Ri zugewiesen.
Abschließend wird der Kontrollfluß des 5. Schrittes so modifiziert, daß als nächstes die separierten rekursiven Aufrufe der parallel-rekursiven rechten Seite bearbeitet werden.
Wert: T.

```
<NEUE-RSEITEN
 <NLAMBDA (ZAHL RS)
  (COND
1 ((ASSOC (QUOTE NEUE-NAMEN)
          BINDE)
   T)
2 (T <SETQ NEUE-NAMEN (MAPCAR (ZAHLENLISTE ZAHL)
                              (FUNCTION (NLAMBDA (X)
                                        (PACK (LIST RS (QUOTE -)
                                                    X>
3  <STKEVAL (STKPOS (QUOTE MAPC)
                    2)
            (QUOTE (SETQ MAPX (APPEND (LIST (CAR MAPX))
                                      NEUE-NAMEN
                                     (CDR MAPX>
   <SETQ BINDE (APPEND BINDE (LIST
                     (CONS (QUOTE *NEUE-NAMEN)
                           NEUE-NAMEN))
                     (MAP2CAR (ZAHLENLISTE ZAHL)
4                           NEUE-NAMEN
                            (FUNCTION (NLAMBDA (Z N)
                            (CONS (PACK (LIST (QUOTE *RS)
                                              Z))
                                  N>
   <SETQ V-LISTE (MAPCAR (ZAHLENLISTE ZAHL)
                         (FUNCTION (NLAMBDA (Z)
                                   (PACK (LIST (QUOTE *V)
5                                        Z>
     <MAP2C V-LISTE NEUE-NAMEN (FUNCTION (NLAMBDA (V NN)
                (SET NN (CDR (ASSOC V BINDE>
6    (SETQ MAPX)
       T>>
```

Bemerk.: Die Funktion arbeitet folgendermaßen:
In 1 wird der Fall berücksichtigt, daß bereits ein Funktionsaufruf von NEUE-RSEITEN bearbeitet worden ist (siehe Struktur der Regel S4-3).
In 2 wird die Liste der neuen Namen erzeugt und in 3 in die MAPX-Liste der zu bearbeitenden rechten Seiten eingebaut.
In 4 werden die Variablenbezeichner *RSj und *NEUE-NAMEN initialisiert.
In 5 wird den Bezeichner Ri-j ein Wert zugewiesen

und in 6 die aktuelle MAPX-Liste (der noch anzuwendenden Regeln) gelöscht, so daß zur Laufzeit in der nächsthöheren MAPX-Liste (der zu bearbeitenden rechten Seiten Ri) das nächste Element (die erste der erzeugten rechten Seiten Ri-j) zur Anwendung kommt.

■

<NOT-ALL-NIL? <NLAMBDA LISTE ... >>

Zweck: Überprüfung, ob von den in LISTE übergebenen Indizes mindestens einer an T gebunden ist.
Aufruf: Innerhalb der Voraussetzungen der Regeln S4-51, s4-52, S4-54, S5-31 und S6-23.
Param.: LISTE: Liste von Indizes.
Ergebnis: Keine Seiteneffekte.
Wert: T, falls mindestens ein Index an T gebunden ist.
NIL sonst.

```
<NOT-ALL-NIL?
  <NLAMBDA LISTE
    (PROG NIL
          (MAPC LISTE (FUNCTION (NLAMBDA (X)
                      (COND
                        ((CDR (ASSOC X BINDE))
                          (RETURN T))
                        (T NIL>>
```

■

<PARALLEL-REK-AUFBAU <NLAMBDA (RS) ... >>

Zweck: Die Funktion soll die im 5. Schritt erzeugten Werte der FORM-Indikatoren von rekursiven Aufrufen, die innerhalb einer parallel-rekursiven rechten Seite auftreten, geeignet in den im Abschnitt 4.4.4. beschriebenen Zielcode einbauen.
Aufruf: Innerhalb der Voraussetzungen der Regel S6-22.
Param.: RS: Parallel-rekursive rechte Seite.
Ergebnis: In der Bindeliste BINDE wird der Variablenbezeichner *PARALLEL-REK-AUFBAU mit einer Liste initialisiert, die den LISP-Code für die vorliegende parallel-rekursive rechte Seite enthält.
Wert: T.

```
<PARALLEL-REK-AUFBAU
   <NLAMBDA (RS)
    (SETQ NAMEN (CAR (GETP RS (QUOTE NEUE-NAMEN>
    (SETQ SYSPARAM (MAPCAR (GETP (QUOTE PARAM-TYP-LISTE)
1                          (QUOTE SYSTEM))
                       (FUNCTION CAR)))
2   <SETQ LISTE1 (MAPCAR NAMEN (FUNCTION (NLAMBDA (N)
                          (CAR (GETP N (QUOTE ARGLIST>
    <SETQ LISTE2 (MAP2CAR LISTE1 NAMEN
                  (FUNCTION (NLAMBDA (ARGLIST NAM)
                    (REMOVE (QUOTE NIL)
```

```
                    (MAPCAR ARGLIST (FUNCTION (NLAMBDA (ARG)
                       (CDDR (ASSOC (QUOTE FORM)
                                    (GETP NAM ARG>
3   <SETQ LISTE2 (EKL (MAPCAR LISTE2
                      (FUNCTION (NLAMBDA (LIST ERG)
                          <MAPC LIST (FUNCTION (NLAMBDA (L)
                                (SETQ ERG (APPEND ERG L>
                           (EVAL (QUOTE ERG>
    <SETQ LISTE3 (MAP2CAR LISTE2 (ZAHLENLISTE (LENGTH NAMEN))
                 (FUNCTION (NLAMBDA (L Z)
                    (SUBST L (QUOTE FORM-LISTE)
                         (APPEND (LIST (QUOTE SETQ)
                                       (PACK (LIST (QUOTE F)
4                                                  Z))
                                 (APPEND (LIST (QUOTE PROG)
                                               (QUOTE FORM-LISTE)
                                         (APPEND (LIST (QUOTE RETURN)
                                          (APPEND (LIST (QUOTE FUNCTION)
                                                        (QUOTE APPLY*))
                                                  (LIST (APPEND SYSPARAM>
    <SETQ LISTE4 (MAPCAR (ZAHLENLISTE (LENGTH NAMEN))
                         (FUNCTION (NLAMBDA (Z)
                             (LIST (QUOTE SUBST)
5                                  (QUOTE (GENSYM))
                                   (LIST (QUOTE QUOTE)
                                         (PACK (LIST (QUOTE F)
                                                     Z>
    <SETQ LISTE5 (LIST (QUOTE RETEVAL)
            <QUOTE (STKNTH 1 (STKPOS (QUOTE PROG>
            (LIST (QUOTE QUOTE)
             (APPEND (LIST (QUOTE PROG)
6                          (QUOTE NIL))
                    (MAPCAR (ZAHLENLISTE (LENGTH NAMEN))
                            (FUNCTION (NLAMBDA (Z)
                                (LIST (QUOTE APPLY*)
                                      (PACK (LIST (QUOTE F)
                                                  Z))
                                (CAR ALGNAME>
7   <SETQ LISTE6 (LIST (QUOTE QUOTE)
                       (APPEND LISTE3 (LIST LISTE5>
8   <MAPC LISTE4 (FUNCTION (NLAMBDA (L)
              (SETQ LISTE6 (APPEND L (LIST LISTE6>
9   (SETQ LISTE6 (LIST (QUOTE SETQ)
                       (QUOTE A-FORM)
                       LISTE6))
10  <SETQ LISTE7 (LIST (QUOTE MAPC)
                       (QUOTE A-FORM)
                       (QUOTE (FUNCTION EVAL>
    (SETQ LISTE8 (APPEND (LIST LISTE6)
                         (LIST LISTE7)))
11    <SETQ BINDE (APPEND BINDE (LIST (CONS
                        (QUOTE *PARALLEL-REK-AUFBAU)
                      LISTE8>
    T>>
```


Bemerk.: Die Arbeitsweise der Funktion soll mit einem Beispiel kurz skizziert werden:
Sei R1 parallel-rekursive rechte Seite mit zwei rekursiven Aufrufen des Algorithmus mit je zwei Argumenten.
In 1 werden die Atome NAMEN und SYSPARAM initialisiert:

```
NAMEN ←—— (R1-1 R1-2)
SYSPARAM ←——  (A B)
```

In 2 werden in LISTE1 die Argumentlisten ARGLIST gesammelt:

```
LISTE1 ←—— ((ARG1 ARG2)(ARG1 ARG2))
```

In 3 werden in LISTE2 die "form"-Werte aus den den Indikatoren ARGj zugeordneten Assoziationslisten gesammelt:

```
LISTE2 ←—— ((FORM1 FORM2)(FORM1 FORM2))
```

In 4 wird um jede FORM-Liste ein LISP-FUNARG-Konstrukt gelegt, und das Ergebnis an LISTE3 gehängt.

```
LISTE3 ←—— ((SETQ F1 (PROG ((FORM1)(FORM2))
                           (RETURN
                            (FUNCTION APPLY* (SYSPARAM))))
                           (SETQ F2 .....))
```

In 5 werden Substitutionsformen für die Bezeichner F1 und F2 erzeugt:

```
LISTE4 ←—— ((SUBST (GENSYM) 'F1)

                (SUBST (GENSYM) 'F2))
```

In 6 wird der RETEVAL-Ausdruck generiert:

```
LISTE5 ←—— (RETEVAL (STKNTH 1 (STKPOS 'PROG))
                    (QUOTE (PROG NIL (APPLY* F1 ALGNAME)
                                     (APPLY* F2 ALGNAME))))
```

In 7 werden SETQ- und RETEVAL-Formen verknüpft:

```
LISTE6 ←—— ((SETQ F1 ...)
            (SETQ F2 ...)
            (RETEVAL ...))
```

In 8 werden die SUBST-Terme vorangestellt:

```
LISTE6 ←—— (SUBST ... F1
            (SUBST .... F2
             ((SETQ F1 ...)(SETQ F2 ...)
                   (RETEVAL ...)))
```

In 9 wird eine Zuweisung erzeugt:

```
LISTE6 ⟵ (SETQ A-FORM (SUBST ... (SUBST ... )))
```

In 10 wird eine Aufrufform für die A-FORM Liste generiert:

```
LISTE7 ⟵ (MAPC A-FORM (FUNCTION EVAL))
```

In 11 schließlich die Komposition

```
LISTE8 ⟵ ((SETQ A-FORM ...)(MAPC A-FORM ...))
```

und die Übergabe an die Bindeliste BINDE.

■

<UMBENNENUNG <NLAMBDA (NR ARG1 ARG2) ... >>

Zweck: Die Funktion vereinfacht die Regelmuster in bestimmten Fällen durch Umbenennung von bereits gebundenen Variablen der Bindeliste BINDE.
Aufruf: Innerhalb der Voraussetzungen der Regel S5-31.
Param.: NR: Interner Index, der die Art der Umbennung angibt.
ARG1: Name einer unzubennenden Variablen.
ARG2: wie ARG1.
Ergebnis: In der Bindeliste BINDE werden die durch NR spezifizierten Umbenennungen durchgeführt.
Wert: T.

```
<UMBENENNUNG!
  <NLAMBDA (NR ARG1 ARG2)
        (* EDITED: "10-FEB-81 16:05:39")
    (COND
      ((EQ NR 1)
        <SETQ BINDE (APPEND BINDE (LIST (CONS (QUOTE *P2)
                                              ARG1)))
                            (LIST (CONS (QUOTE *OP2)
                                        ARG2>
        (SETQ BINDE (REMOVE (QUOTE (2 . T))
                            BINDE))
        <SETQ BINDE (APPEND BINDE (LIST (LIST 2>
        T)
      (T (FEHLER 16>>
```

■

<ZAHLENLISTE <LAMBDA (Z ERG) ... >>

Zweck: Dienstfunktion. Erstellt eine Liste der Zahlen 1 bis Z bzw. eine Teilliste davon.
Aufruf: Durch die Funktionen IN-BINDE!, STANDARD-BIND!, ENTKLAMMERUNG, PARAMETER-REIHENFOLGE?, NEUE-RSEITEN und PARALLEL-REK-AUFBAU.
Param.: Z: Gibt die obere Grenze an.
ERG: Lok. Variable.

Ergebnis: Keine Seiteneffekte.
Wert: Für Z>0 wird die Liste (1 ... Z) erzeugt, für Z<0 die Liste (|Z| ... 11), für Z=0 die Liste (0).

```
<ZAHLENLISTE
  <LAMBDA (Z ERG)
  (SETQ ERG (LIST 0))
  (PROG NIL
    LOP (COND
          ((EQ Z 0)
            (RETURN (CDR ERG)))
          ((GREATERP Z 0)
            <SETQ ERG (APPEND ERG (LIST (ADD1 (CAR (LAST ERG>
            (SETQ Z (SUB1 Z))
            (GO LOP))
          ((LESSP Z 0)
            <SETQ ERG (APPEND ERG (LIST (PLUS Z 12>
            (SETQ Z (ADD1 Z))
            (GO LOP))
          (T (FEHLER 11>>
```

Bemerk.: Der Wert eines Aufrufs für Z<0 wird z.B. in IN-BINDE! in dem Fall verwendet, daß insgesamt Z<9 Form-Konstrukte zu berücksichtigen sind und den (9-Z) überflüssigen *Fj-Variablen der Wert NIL zugewiesen werden muß. (Siehe Funktionsbeschreibun IN-BINDE!)

■

Die folgenden Funktionen dienen alle der Steuerung der Regelanwendung in den Schritten der Algorithmus-Konstruktionsphase. Die die Regelauswahl steuernden Funktionen Si-STEUERUNG sind alle von der Struktur

```
<Si-STEUERUNG  ( ... )
               (SETQ FORTS-LISTE ...)
               (SETQ STOP-LISTE ....)
               (STEUERUNGS-RUMPF)>,
```

wobei STEUERUNGS-RUMPF die Funktion ist, die die Angaben in der Fortsetzungsliste FORTS-LISTE und der Stopliste STOP-LISTE mit den Kriterien der Regelauswahl (siehe 4.1.3.) geeignet verbindet und die nächste zur Anwendung kommende Regel bestimmt.

Die Werte der Atome FORT-LISTE und STOP-LISTE bestimmen den in Abschnitt 4.1.3. beschriebenen Regelanwendungspfad:

Sei z.B. die Menge der Regelnamen {1 ..10} und seien die Werte von FORTS-LISTE und STOP-LISTE gegeben durch:

```
FORTS-LISTE ⟵ ((2 . 5)(3 . 6)(6 . 8))
STOP-LISTE  ⟵ (5 7 9 10)
```

Wird dann eine Regel N erfolgreich angewandt und existiert in FORTS-LISTE ein Paar (N . R), so ist R der Name der als nächte zur Anwendung kommenden Regel. Ist N in STOP-LISTE vermerkt, so wird die Bearbeitung beendet.

Beispiel: Regelanwendungspfad, wenn jeweils die "kleinste" Regel der aktuellen Menge erfolgreich ist:
{1 ... 10} , {2 ... 10} , {5 ... 10} , {}
oder wenn nicht jeweils die erste erfolgreich ist:
{1 ... 10} , {2 ... 10} , {3 ... 10} , {6 ... 10} , {7 ... 10} , {}
[Im erfolglosen Fall wird die nächstfolgende Regel gewählt, sofern das Kriterium PATTERNLAENGE es nicht verbietet (siehe 4.1.3.).]

Die folgenden Funktionen werden von den Funktionen SCHRITT-i aufgerufen:

```
<S1-STEUERUNG
  <LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
    (SETQQ STOP-LISTE (1 2))
    (SETQQ FORTS-LISTE NIL)
    (STEUERUNGS-RUMPF>>

<S2-STEUERUNG
  <LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
    (SETQQ STOP-LISTE (9))
    (SETQQ FORTS-LISTE ((1 . 9)
             (2 . 9)))
    (STEUERUNGS-RUMPF>>

<S4-STEUERUNG
  <LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
    (SETQQ STOP-LISTE (2 3 4 8))
    (SETQQ FORTS-LISTE ((1 . 5)
             (5 . 9)))
    (STEUERUNGS-RUMPF>>

<S5-STEUERUNG
  <LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
    (SETQQ STOP-LISTE (5 8 9 D))
    (SETQQ FORTS-LISTE ((1 . 9)
             (3 . 8)
             (4 . 8)
             (2 . 5)))
    (STEUERUNGS-RUMPF>>

<S6-STEUERUNG
  <LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
    (SETQQ STOP-LISTE (1 2))
    (SETQQ FORTS-LISTE NIL)
    (STEUERUNGS-RUMPF>>
```


Bemerk.: Eine S3-STEUERUNG ist zur Zeit nicht nötig, da die entsprechende Basis nur eine Regel umfaßt.

■

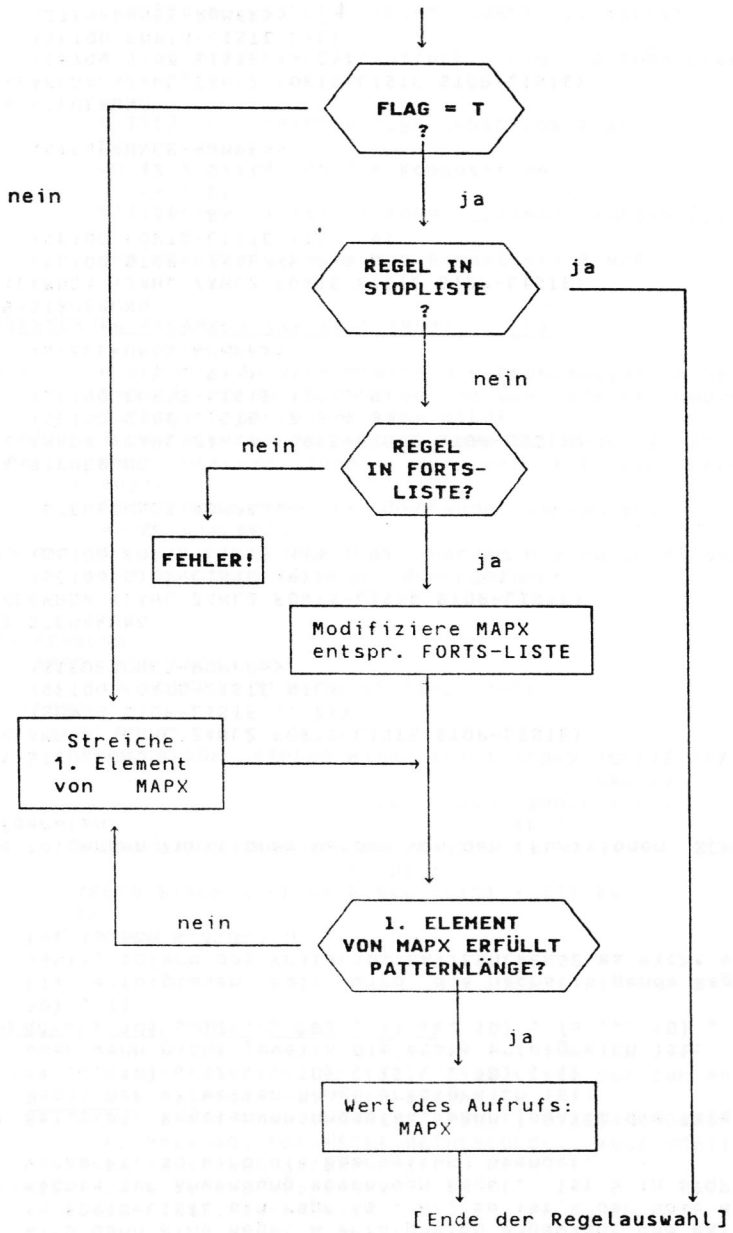
<STEUERUNGS-RUMPF <NLAMBDA NIL ... >>

Zweck: Auswahl der nächsten anzuwendenden Regel.
Aufruf: Durch die Funktionen Si-STEUERUNG.
Ergebnis: In der aktuellen MAPX-Liste der noch anzuwendenden Regeln werden solange die ersten Listenelemente gestrichen, bis die Regelanwendungskriterien für das erste Element zutreffen.
Wert: Die modifizierte Liste der Regeln.

```
<STEUERUNGS-RUMPF
  <NLAMBDA NIL
    <SETQ ZAHL (CADDDR (UNPACK (CAAR MAPX>
    <COND
      ((AND FLAG (MEMB ZAHL STOP-LISTE))
        (SETQ MAPX))
      <(AND FLAG (ASSOC ZAHL FORTS-LISTE))
        (SETQ ZAHL2 (CDR (ASSOC ZAHL FORTS-LISTE)))
        (PROG NIL
          LOP (SETQ MAPX (CDR MAPX))
              (COND
                ((EQ MAPX NIL)
                  (PRINTL "****STEUERUNGS-FEHLER****")
                  (PRINTL "DER IN FORTS-LISTE ANGEGEBENE
                           WERT EXISTIERT NICHT"))
                ((EQ (CADDDR (UNPACK (CAAR MAPX)))
                     ZAHL2)
                  (RETURN MAPX))
                (T (GO LOP>
      (T (SETQ MAPX (CDR MAPX>
    (PROG NIL
      LOP (COND
            ((GREATERP (PATTERNLAENGE (CAAR MAPX))
                       LAENGE)
              (SETQ MAPX (CDR MAPX))
              (GO LOP))
            (T (RETURN MAPX>>
```

Bemerk.: Die Verwendung der Auswahlkriterien, wie sie im Abschnitt 4.1.3. besprochen wurde, spiegelt sich im Blockdiagramm von STEUERUNGS-RUMPF wieder:

Blockdiagramm STEUERUNGS-RUMPF:



<PATTERNLAENGE <LAMBDA (NAME WERT) ... >>

Zweck: Verwaltungsfunktion, die für jede Regel die minimale Musterlänge speichert.
Aufruf: Durch STEUERUNGS-RUMPF.
Param.: NAME: Der Name der Regel.
WERT: Lok.Variable, die die minimale Musterlänge bezeichnet.
Ergebnis: Keine Seiteneffekte.
Wert: Die minimale Musterlänge, falls sie in PATTERNLAENGE gespeichert ist.
Sonst NIL.

```
<PATTERNLAENGE
  <LAMBDA (NAME WERT)
    (COND
       (NAME NIL)
      (T (RETURN)))
    <SETQ WERT (SELECTQ <PACK (LIST (CAR (UNPACK NAME))
                                    (CADR (UNPACK NAME>
                  <S1 (CDR (ASSOC NAME (QUOTE ((S1-1 . 5)
                                               (S1-2 . 5>
                  <S2 (CDR (ASSOC NAME (QUOTE ((S2-1 . 5)
                                               (S2-2 . 5)
                                               (S2-9 . 0>
                  <S4 (CDR (ASSOC NAME (QUOTE ((S4-1 . 5)
                                               (S4-2 . 3)
                                               (S4-3 . 9)
                                               (S4-4 . 6)
                                               (S4-51 . 21)
                                               (S4-52 . 28)
                                               (S4-53 . 23)
                                               (S4-54 . 12)
                                               (S4-55 . 18)
                                               (S4-9 . 8>
                  <S5 (CDR (ASSOC NAME (QUOTE ((S5-D1 . 3)
                                               (S5-1 . 3)
                                               (S5-2 . 5)
                                               (S5-31 . 5)
                                               (S5-32 . 6)
                                               (S5-33 . 10)
                                               (S5-4 . 12)
                                               (S5-5 . 6)
                                               (S5-8 . 8)
                                               (S5-9 . 4>
                  <S6 (CDR (ASSOC NAME (QUOTE ((S6-1 . 0)
                                               (S6-21 . 0)
                                               (S6-22 . 0)
                                               (S&-23 . 0)>>
    (COND
          (WERT WERT)
          (T (FEHLER 40))>>
```

Bemerk.: 1) Falls NAME=NIL, wird der aufrufende Suchprozeß in STEUERUNGS-RUMPF durch RETURN beendet.
2) Solange kein Regelmanipulationpaket existiert, muß jede Veränderung der Regelmenge geeignet in PATTERNLAENGE berücksichtigt werden (per Hand!).

■

<MODIFIZIERE-STEUERUNG <NLAMBDA (ST-NAME STOPLIST FORTSLIST)>>

Zweck: Dienstfunktion, die die bequeme Umdefinition einer Si-STEUERUNG erlaubt.
Aufruf: Vom Benutzer.
Param.: ST-NAME: Name der zu modifizierenden Steuerung.
STOPLIST: Liste der Regelnamen, nach deren erfolgreicher Anwendung keine weitere Regel angewandt werden soll.
FORTSLIST: Liste von dotted-pairs, bei denen der CDR die nächste anwendbare Regel bezeichnet, wenn der CAR der Name der aktuellen und erfolgreich angewandten Regel ist.
Ergebnis: Die entsprechende Steuerungsfunktion erhält eine neue Funktionsdefinitionszelle.
Wert: Meldung an Benutzer.

```
<MODIFIZIERE-STEUERUNG
  <LAMBDA (ST-NAME STOPLIST FORTSLIST)
    <PUTD ST-NAME (DSUBST STOPLIST (QUOTE FORM-ST-LIST)
      (SUBST FORTSLIST (QUOTE FORM-F-LIST)
       (QUOTE (LAMBDA (ZAHL ZAHL2 FORTS-LISTE STOP-LISTE)
                      (SETQQ STOP-LISTE FORM-ST-LIST)
                      (SETQQ FORTS-LISTE FORM-F-LIST)
                      (STEUERUNGS-RUMPF>
  (PRINTL "STEUERUNG " ST-NAME "MODIFIZIERT MIT  ")
  (PRINTL "STOPLISTE: " STOPLIST)
  (PRINTL "FORTSETZUNGSLISTE: " FORTSLIST>>
```

■

5.2.1.4. Funktionen der LISP-Phase

Aus dem Header der Algorithmen-Spezifikation, also den Angaben unter den Schlusselwortern "Parametertypen", "Voraussetzungen", "Hilfsalgorithmen" und "Wert", und aus der Liste der erkannten Hilfsfunktionen sowie den Werten der globalen Variablen PRA, REK und ALGNAME wird schrittweise eine Ablauf-Umgebung für den Algorithmus konstruiert, in der dieser ausgefuhrt werden kann (s.a. 4.1.1. und 4.1.2.3).

<LISP-PHASE <LAMBDA (...) ... >>

Zweck: Konstruktion der Ablauf-Umgebung für den Algorithmus.
Aufruf: durch IMPLEMENTIERE-ALG
Ergebnis: Die komplette Ablauf-Umgebung wird als LAMBDA-Form an die Funktions-Definitions-Zelle des Literalen Atoms ALGNAME gebunden.
Ferner wird das Atom IMPLEMCOMS."ALGNME" definiert, so daß der vollständig implementierte Algorithmus mit Hilfe des MAKEFILE-Kommandos des INTERLISP-Systems bequem auf die Datei IMPLEM."ALGNAME".nn gerettet werden kann.
Die vollständige Ablauf-Umgebung wird nach Konstruktion ausgedruckt.
Wert: NIL
Arbeitsweise: Die Ablauf-Umgebung eines Algorithmus wird schrittweise unter Verwendung der Werte der globalen Variablen ALGNAME, PARAM-TYP-LISTE (Property-List: Indikatoren Benutzer und System), PRA, REK, VORAUSSETZUNGEN, HILFSALGORITHMEN und HILFSFUNKTIONEN als LAMBDA-Form erzeugt. Zunächst werden als innerste Ausdrücke die "RPLACA"- und die "APPLY"-Form konstruiert, wobei berücksichtigt wird, ob es sich um einen nicht-rekursiven, einen rekursiven oder einen parallel-rekursiven Algorithmus handelt. Dann werden die Wertzuweisungs-Statements für System-Parameter programmiert. Es folgen, von innen nach außen geschachtelt, jeweils COND-Ausdrücke für Voraussetzungen, Hilfsalgorithmen, Hilfsfunktionen und die Überprüfung der Benutzer-Parameter. Jeder COND-Ausdruck besteht aus einer Alternative: trifft diese zu, so wird zur Ausführungszeit des Algorithmus mit der Abarbeitung der Ablauf-Umgebung fortgefahren, sonst wird eine Fehlermeldung ausgegeben und das System geht in den Break-Mode. Die COND-Ausdrücke werden mit Hilfe der Unterfunktion SET-ENVIRONMENT konstruiert.
Der äußerste COND-Ausdruck wird in eine LAMBDA-Form mit dem Namen des Algorithmus und der Parameterliste eingebettet.

```
<LISP-PHASE
  <LAMBDA (ENVIRONMENT)
  (* "LISP-PHASE RUFT FOLGENDE FUNKTIONEN AUF:
      SET-ENVIRONMENT, CONVERT-KWOTE,
      DATENTYPP, PRINTL")
(*COM* "KONVERTIEREN GEKWOTETER ATOME IN PARAM-TYP-LISTE,
        INDIKATOR SYSTEM")
     (SET (CAR ALGNAME))
          (QUOTE SYSTEM)
          (CONVERT-KWOTE (GETP (QUOTE PARAM-TYP-LISTE)
                               (QUOTE SYSTEM>
(*COM* "DEFINITION DES ATOMS IMPLEM.<ALGNAME>COMS
```

```
      FUER DAS MAKEFILE-KOMMANDO")
   <RPLACA (PACK (LIST (QUOTE IMPLEM.)
                       (CAR ALGNAME)
                       (QUOTE COMS)))
           (LIST (LIST (QUOTE FNS)
                       (CAR ALGNAME))
                 (LIST (QUOTE PROP)
                       (QUOTE ALL)
                       (CAR ALGNAME>
(*COM* "ANHAENGEN VON RELATION UND KOMMENTAR AN <ALGNAME>")
    (COND
      <RELATION (PUT (CAR ALGNAME)
                     (QUOTE RELATION)
                     (LIST (CAR RELATION)
                           (CAAR RELATION>
      (T NIL))
    (COND
      (KOMMENTAR (PUT (CAR ALGNAME)
                      (QUOTE KOMMENTAR)
                      KOMMENTAR))
      (T NIL))
(*COM* "KONSTRUKTION DER RETEVAL-FORM, UNTERTEILT NACH
        PARALLEL-REKURSIVEN, REKURSIVEN UND NICHT-
        REKURSIVEN ALGORITHMEN")
    <COND
      <PRA (SETQ ENVIRONMENT
             (LIST (QUOTE DSUBST)
                   <LIST (QUOTE QUOTE)
                         (LIST (QUOTE RETEVAL)
                               <LIST (QUOTE STKNTH)
                                     1
                                     (LIST (QUOTE STKPOS)
                                           (QUOTE (QUOTE PROG>
                               (LIST (QUOTE QUOTE)
                                     (LIST (QUOTE APPLY)
                                           (CAR ALGNAME)
                                           NIL>
                   (LIST (QUOTE QUOTE)
                         (APPEND ALGNAME ALGPARAM))
                   (LIST (QUOTE COPY)
                         (LIST (QUOTE GETP)
                               (LIST (QUOTE QUOTE)
                                     (CAR ALGNAME))
                               (LIST (QUOTE QUOTE)
                                     (QUOTE ALGORITHMUS>
      <REK (SETQ ENVIRONMENT
             (LIST (QUOTE DSUBST)
                   <LIST (QUOTE QUOTE)
                         (LIST (QUOTE RETEVAL)
                               (LIST (QUOTE STKPOS)
                                     (LIST (QUOTE QUOTE)
                                           (CAR ALGNAME))
                                     -1)
                               (LIST (QUOTE QUOTE)
                                     (LIST (QUOTE APPLY)
                                           (CAR ALGNAME)
```



```
                                                 NIL>
                   (LIST (QUOTE QUOTE)
                         (APPEND ALGNAME ALGPARAM))
                   (LIST (QUOTE COPY)
                         (LIST (QUOTE GETP)
                               (LIST (QUOTE QUOTE)
                                     (CAR ALGNAME))
                               (LIST (QUOTE QUOTE)
                                     (QUOTE ALGORITHMUS>
      (T (SETQ ENVIRONMENT
           (LIST (QUOTE COPY)
                 (LIST (QUOTE GETP)
                       (LIST (QUOTE QUOTE)
                             (CAR ALGNAME))
                       (LIST (QUOTE QUOTE)
                             (QUOTE ALGORITHMUS>
(*COM* "ERZEUGEN VON DSUBST-FORMEN FUER
        BENUTZER-PARAMETER")
    <MAPC (GETP (QUOTE PARAM-TYP-LISTE)
                (QUOTE BENUTZER))
          (FUNCTION (LAMBDA (PARAM)
              (COND
                ((DATENTYPP (CDR PARAM))
                  (SETQ ENVIRONMENT
                    (LIST (QUOTE DSUBST)
                          (CAR PARAM)
                          (LIST (QUOTE QUOTE)
                                (CAR PARAM))
                          ENVIRONMENT)))
                (T NIL>
(*COM* "KONSTRUKTION DER APPLY-FORM, GETRENNT FUER
        PARALLEL-REKURSIVE %(EXPLIZITER RETURN-WERT%)
        UND ANDERE ALGORITHMEN")
    <COND
      <PRA (SETQ ENVIRONMENT
             (LIST (LIST (QUOTE RPLACA)
                         (LIST (QUOTE QUOTE)
                               (CAR ALGNAME))
                         ENVIRONMENT)
                   (LIST (QUOTE PROG)
                         NIL
                         (LIST (QUOTE APPLY)
                               (CAR ALGNAME)
                               NIL)
                         (LIST (QUOTE RETURN)
                               (LIST (QUOTE EVAL)
                                     (CAAR ALGORITHMUS-WERT>
      (T (SETQ ENVIRONMENT
           (LIST (LIST (QUOTE RPLACA)
                       (LIST (QUOTE QUOTE)
                             (CAR ALGNAME))
                       ENVIRONMENT)
                 (LIST (QUOTE APPLY)
                       (CAR ALGNAME)
                       NIL>
(*COM* "ERZEUGEN DER FORMEN, DIE DIE SYSTEM-PARAMETER
```

```
      MIT WERTEN BELEGEN")
   (COND
     ((GETP (QUOTE PARAM-TYP-LISTE)
            (QUOTE SYSTEM))
      (SETQ ENVIRONMENT
        (APPEND
         <MAPCAR (GETP (QUOTE PARAM-TYP-LISTE)
                       (QUOTE SYSTEM))
                 (FUNCTION (LAMBDA (PARAM)
                  (COND
                    <<LISTP (CADDR PARAM))
                      (COND
                        ((EQ (CAR (CADDR PARAM))
                             (QUOTE INIT))
                          (CADDR PARAM))
                        (T (LIST (QUOTE SETQ)
                                 (CAR PARAM)
                                 (CADDR PARAM>
                    (T (LIST (QUOTE SETQ)
                             (CAR PARAM)
                             (CADDR PARAM>
          ENVIRONMENT)))
      (T NIL))
(*COM* "ERZEUGEN DER FORMEN, DIE DIE VORAUSETZUNGEN TESTEN.
        AUFRUF DER UNTERFUNKTION SET-ENVIRONMENT: ES WIRD
        NUR DIE FORM UND DIE FEHLERMELDUNG ALS PARAMETER
        UEBERGEBEN")
    (COND
      <VORAUSSETZUNGEN
         (MAPC VORAUSSETZUNGEN
               (FUNCTION (LAMBDA (VORAUSSETZUNG)
                (SET-ENVIRONMENT VORAUSSETZUNG
                    (LIST (QUOTE PRINTL)
                          "***ERROR***
                           VORAUSSETZUNG"
                          <LIST (QUOTE QUOTE)
                                (MAPCAR
                                  VORAUSSETZUNG
                                 (FUNCTION (LAMBDA (ELEM)
                                   <<AND (LISTP ELEM)
                                         (EQ (CAR ELEM)
                                             (QUOTE EVAL)))
                                     (CADAR (LAST (CADR ELEM>
                                   (T ELEM>
                          "NICHT ERFUELLT">
      (T NIL))
(*COM* "KONSTRUKTION DER COND-AUSDRUECKE, DIE DIE
        DEFINITION VON HILFSALGORITHMEN UND
        HILFSFUNKTIONEN UEBERPRUEFEN")
    (COND
      <HILFSALGORITHMEN
          (MAPC HILFSALGORITHMEN
                (FUNCTION (LAMBDA (HILFSALG)
                  (SET-ENVIRONMENT
                      (LIST (QUOTE GETD)
                            (LIST (QUOTE QUOTE)
```



```
                                  HILFSALG))
                          (LIST (QUOTE PRINTL)
                                "***ERROR***
                                HILFSALGORITHMUS"
                                (LIST (QUOTE QUOTE)
                                      HILFSALG)
                                "UNDEFINIERT">
   (COND
     <HILFSFUNKTIONEN
        (MAPC <DREMOVE (QUOTE EQ)
              (DREMOVE (QUOTE NEQ)
               (DREMOVE (QUOTE LESSP)
                (REMOVE (QUOTE GREATERP)
                  HILFSFUNKTIONEN>
              (FUNCTION (LAMBDA (HILFSFKT)
                (SET-ENVIRONMENT
                    (LIST (QUOTE GETD)
                          (LIST (QUOTE QUOTE)
                                HILFSFKT))
                          (LIST (QUOTE PRINTL)
                                "***ERROR***
                                 HILFSFUNKTION"
                                (LIST (QUOTE QUOTE)
                                      HILFSFKT)
                                "UNDEFINIERT">
      (T NIL))
(*COM* "ERZEUGEN DER FORM; DIE DEN DATENTYP DER WERTE
        VON BENUTZER-PARAMETERN UEBERPRUEFT.
        BEI VERAENDERTEN INTERLISP-DATENTYPEN MUSS AUCH
        DIESER TEIL VON LISP-PHASE MODIFIZIERT WERDEN")
    <MAPC (REVERSE (GETP (QUOTE PARAM-TYP-LISTE)
                         (QUOTE BENUTZER)))
          (FUNCTION (LAMBDA (PARAM)
            (SELECTQ (CDR PARAM)
              (LISPATOM (SET-ENVIRONMENT
                          (LIST (QUOTE OR)
                                (LIST (QUOTE TYPEP)
                                      (CAR PARAM)
                                      30)
                                (LIST (QUOTE TYPEP)
                                      (CAR PARAM)
                                      22)
                                (LIST (QUOTE TYPEP)
                                      (CAR PARAM)
                                      23))
                          (LIST (QUOTE PRINTL)
                                "***ERROR***"
                                (CAR PARAM)
                                "NICHT VOM TYP LISPATOM")))
              (LISPZELLE (SET-ENVIRONMENT
                           (LIST (QUOTE TYPEP)
                                 (CAR PARAM)
                                 21)
                           (LIST (QUOTE PRINTL)
                                 "***ERROR***"
                                 (CAR PARAM)
```

```
                    "NICHT VOM TYP
                     LISP-ZELLE")))
(INTEGER (SET-ENVIRONMENT
           (LIST (QUOTE OR)
                 (LIST (QUOTE TYPEP)
                       (CAR PARAM)
                       22)
                 (LIST (QUOTE TYPEP)
                       (CAR PARAM)
                       23))
                 (LIST (QUOTE PRINTL)
                       "***ERROR***"
                       (CAR PARAM)
                       "NICHT VOM TYP
                        INTEGER")))
(REAL (SET-ENVIRONMENT
        (LIST (QUOTE OR)
              (LIST (QUOTE TYPEP)
                    (CAR PARAM)
                    26)
              (LIST (QUOTE TYPEP)
                    (CAR PARAM)
                    27))
              (LIST (QUOTE PRINTL)
                    "***ERROR***"
                    (CAR PARAM)
                    "NICHT VOM TYP REAL")))
 (LITATOM (SET-ENVIRONMENT
            (LIST (QUOTE TYPEP)
                  (CAR PARAM)
                  30)
            (LIST (QUOTE PRINTL)
                  "***ERROR***"
                  (CAR PARAM)
                  "NICHT VOM TYP LITATOM")))
 (BOOL (SET-ENVIRONMENT
         (LIST (QUOTE OR)
               (LIST (QUOTE EQ)
                     (CAR PARAM)
                     (QUOTE T))
               (LIST (QUOTE EQ)
                     (CAR PARAM)
                     NIL))
         (LIST (QUOTE PRINTL)
               "***ERROR***"
               (CAR PARAM)
               "NICHT VOM TYP BOOL")))
 (ARRAY (SET-ENVIRONMENT
          (LIST (QUOTE TYPEP)
                (CAR PARAM)
                31)
          (LIST (QUOTE PRINTL)
                "***ERROR***"
                (CAR PARAM)
                "NICHT VOM TYP ARRAY")))
 (STRING (SET-ENVIRONMENT
```



```
                    (LIST (QUOTE TYPEP)
                          (CAR PARAM)
                          35)
                    (LIST (QUOTE PRINTL)
                          "***ERROR***"
                          (CAR PARAM)
                          "NICHT VOM TYP STRING")))
          (SET-ENVIRONMENT
            <LIST (QUOTE EQ)
                  (LIST (QUOTE QUOTE)
                        (CDR PARAM))
                  (LIST (QUOTE GETP)
                        (CAR PARAM)
                        (LIST (QUOTE QUOTE)
                              (COND
                                ((MEMB (QUOTE -)
                                       (UNPACK
                                         (CDR PARAM)))
                                 (QUOTE
                                   PASSIVSTRUKTUR)))
                                (T (QUOTE STRUKTUR>
          (LIST (QUOTE PRINTL)
                "***ERROR***"
                (CAR PARAM)
                "NICHT VOM TYP"
                (LIST (QUOTE QUOTE)
                      (CDR PARAM>
(*COM* "ERZEUGEN DER AEUSSEREN LAMBDA-FORM")
    (PUTD (CAR ALGNAME)
          (APPEND (QUOTE <LAMBDA>)
                  (LIST ALGPARAM)
                  (LIST ENVIRONMENT)))
    (TERPRI)
    (PRINTL "ERGEBNIS DER LISP-PHASE:")
    (PRETTYPRINT ALGNAME)
    (REMPROPS (LIST (QUOTE PARAM-TYP-LISTE>>
```

■

<u><SET-ENVIRONMENT <LAMBDA (FORM FORM-T) ... >></u>

Zweck: Erzeugen der COND-Ausdrücke in der Ablauf-Umgebung des Algorithmus.

Aufruf: durch LISP-PHASE

Param.: FORM evaluiert zu einer Liste, die als Prädikat der ersten Klausel des zu erzeugenden COND-Ausdruck benutzt wird;
FORM-T evaluiert zu einer (PRINTL "Fehlermeldung")-Form, die als T-Klausel des COND-Ausdrucks verwendet wird.

Ergebnis: Der Wert der LISP-PHASE-lokalen Variablen ENVIRONMENT wird neu definiert:
Der alte Wert dieser Variablen wird als Aktions-Teil der ersten Klausel in den neu konstruierten COND-Ausdruck eingebettet, wobei das Prädikat durch FORM

bestimmt wird. Die T-Klausel besteht aus der durch FORM-T übergebenen Druckanweisung für eine Fehlermeldung und einem Break-Aufruf.

Wert: NIL

```
<SET-ENVIRONMENT
  <LAMBDA (FORM FORM-T)
    (COND
      <(EQ (CAR ENVIRONMENT)
           (QUOTE COND))
        (SETQ ENVIRONMENT (LIST (QUOTE COND)
                                (LIST FORM ENVIRONMENT)
                                (LIST (QUOTE T)
                                      FORM-T
                                      (LIST (QUOTE BREAK1)
                                            NIL
                                            (QUOTE T))
                                      NIL>
      (T (SETQ ENVIRONMENT (LIST (QUOTE COND)
                                 (APPEND (LIST FORM)
                                         ENVIRONMENT)
                                 (LIST (QUOTE T)
                                       FORM-T
                                       (LIST (QUOTE BREAK1)
                                             NIL
                                             (QUOTE T))
                                       NIL)))
         NIL>>
```

<CONVERT-KWOTE <LAMBDA (LISTE) ... >>

Zweck: Konvertiert gekwotete Atome 'Atom in beliebigen Listen in die Form (QUOTE Atom).

Bem.: Besprechnung von CONVERT-KWOTE in Abschnitt 5.4. (Seite 516)

<DATENTYPP <LAMBDA (TYP PRINT? ...) ... >>

Zweck: Überprüfung von Datentypen auf Zulässigkeit.

Bem.: Beschreibung von DATENTYPP in Abschnitt 5.2.1.2.2. (Seite 411).

5.2.2. Regeln des ALGCOMP-Systems

Die Regeln, die zur Transformation einer Eingabespezifikation benötigt werden, sind auf sechs Regelbasen verteilt, die in Anlehnung an die zugeordneten Schritte unter den Bezeichnern S1-REGELN, S2-REGELN, ... , S6-REGELN zugreifbar sind. Die Manipulation der Regelmengen und der einzelnen Regel (Veränderung, Erzeugung, Vernichtung) muß vom Regelschreiber "per Hand" vorgenommen werden! Bei dieser Arbeit sind auch eine Reihe von Seiteneffekten zu berücksichtigen, wie etwa die Gewährleistung der Ordnungsstruktur innerhalb einer Regelmenge oder die Hinzufügung von Informationen über neue Regeln zu den Steuerfunktionen der Regelanwendung (siehe dazu 4.1.3.). Ein spezielles Regelmanipulationspaket für ALGCOMP, analog dem des ADTCOMP-Systems, wird bei steigender Anzahl von Regeln notwendig werden.

Da die meisten Regeln "Kondensate" aus mehreren anderen (alten) Regeln darstellen (die Regelmenge umfaßte zeitweilig 75 Regeln gegenüber 31 jetzt), ging durch den Verdichtungsprozeß ein Teil der Übersichtlichkeit verloren. Deswegen sind den meisten Regeln kurze Anmerkungen zugefügt, die Struktur und Arbeitsweise erläutern.

Die Syntax und die Semantik der Regeln sind gegenüber dem ADTCOMP-System unverändert geblieben (siehe [EKT 79]). Die Benennungen der in den V-Voraussetzungen definierten Variablen sind oft so gewählt, daß sie bei ihrer Verwendung im Muster der Regeln auf die an dieser Stelle erwartete Klasse von Objekten hinweisen (*OP für Operationen, *IND für Indizes). Allerdings ist nicht zwingend nach einem Matching an eine *const - Variable auch eine Konstante (z.B. UNDEF od. TRUE) gebunden, d.h. es wird nicht überprüft, ob es sich um ein Objekt der durch den Variablennamen angedeuteten Art handelt (siehe dazu den Bindungsmechanismus des ADTCOMP-Pattermatchers REGELPRUEF in [EKT 79]). Diese Tatsache ist eine häufige Ursache für eine fehlerhafte Behandlung neuer, bisher nicht betrachteter Konstrukte.

Zu den stichwortartigen Angaben, die für jede Regel unter "Ergebnis" gemacht werden, sei auf ausführlicheren Text im Abschnitt 4.1.2 über die einzelnen Phasen der Implementierung hingewiesen.

S1-REGELN

Die Muster der S1-Regeln sind jeweils zum Matching für eine ganze linke Seite (*LS oder Li) konzipiert.

S1-1

Ziel: Die Regel soll erkennen, daß Li eine Konjunktion von Vergleichen ist und die Vergleiche separieren.

Ergebnis: Die Indikatoren VGLj, LOGOP und VGLLIST werden auf

der Property-Liste von Li entsprechend dem Erkannten initialisiert.

```
(S1-1 (V *V1 AUSDRUCK)
      (V *V2 AUSDRUCK)
      (V *V3 AUSDRUCK)
      (V *V4 AUSDRUCK)
      (V *V5 AUSDRUCK)
      (A %( *V1 (? 1 UND *V2)
         (? 2 UND *V3)
         (? 3 UND *V4)
         (? 4 UND *V5)
         %))
      (V 9 NICHT 1)
      =%>
      (*LS VGL1 EXPR %. *V1)
      ((? 1 *LS VGL2 EXPR %. *V2))
      ((? 2 *LS VGL3 EXPR %. *V3))
      ((? 3 *LS VGL4 EXPR %. *V4))
      ((? 4 *LS VGL5 EXPR %. *V5))
      (*LS LOGOP (? 9 NONE)
           (? 1 AND))
      (*LS VGLLIST VGL1 (? 1 VGL2)
           (? 2 VGL3)
           (? 3 VGL4)
           (? 4 VGL5)))
```

Bemerk.:
1) Die aktuelle Implementierung der Regel erkennt maximal 5 Vergleiche.
2) Das kürzeste matchbare Muster ist %(*V1 %). Die minimale Musterlänge ML (siehe 4.1.3.) ist 5, da für *V1 die Vereinbarung "Ausdruck" getroffen wurde, und ein "Ausdruck" aus mindestens 3 Atomen besteht (%(*X %)).

■

S1-2

Ziel: Die Regel soll erkennen, daß Li eine Disjunktion von Vergleichen ist und die Vergleiche separieren.

Ergebnis: Die Indikatoren VGLj, LOGOP und VGLLIST werden auf der Property-Liste von Li entsprechend dem Erkannten initialisiert.

```
(S1-2 (V *V1 AUSDRUCK)
      (V *V2 AUSDRUCK)
      (V *V3 AUSDRUCK)
      (A %( *V1 (? 1 ODER *V2)
         (? 2 ODER *V3)
         %))
      (V 4 NICHT 1)
      =%>
      (*LS VGL1 EXPR %. *V1)
      ((? 1 *LS VGL2 EXPR %. *V2))
      ((? 2 *LS VGL3 EXPR %. *V3))
```



```
      (*LS LOGOP (? 4 NONE)
           (? 1 OR))
      (*LS VGLLIST VGL1 (? 1 VGL2)
           (? 2 VGL3>
```

Bemerk.:
1) Die z.Zt. implementierte Form der Regel erkennt maximal 3 Vergleiche.
2) Die minimale Musterlänge ist 5.

■

Für Mischformen von Konjunktionen und Disjunktionen oder für andere logische Verknüpfungen brauchten keine Regeln geschrieben werden, da diese Fälle in den bearbeiteten Spezifikationen nicht auftauchten. Sie lassen sich in entsprechender Weise jedoch leicht erzeugen.

S2-REGELN

Die Muster der S2-Regeln sind jeweils zum Matching für einen einzelnen Vergleich konzipiert.

S2-1

Ziel: Die Regel soll das vorliegende Relationszeichen des Vergleichs erkennen und die rechts und links davon stehenden Terme separieren.

Ergebnis: Auf der dem Indikator VGLj als Wert zugeordneten Assoziationsliste werden Assoc-Paare mit den Schlüsseln "links", "rechts" und "rel" definiert und mit Werten, die die nach LISP transformierten Teilstücke des Vergleichs darstellen, initialisiert.

```
(S2-1 (V *REL (ODER %>= %> %< %<= == /=))
      (V *P1 ATOM)
      (V *CONST ATOM)
      (V *P2 ATOM)
      (V *OP1 ATOM)
      (V *OP2 ATOM)
      (V *IND1 ATOM)
      (V *IND2 ATOM)
      (V *LISP-OD-INK1 ATOM)
      (V *LISP-OD-INK2 ATOM)
      (A %( (? 0 *P1 *REL *CONST %))
         (? 1 %( *P1 *OP1)
         (? 2 *IND1)
         (? 3 %( *LISP-OD-INK1 *IND1 %))
         (? 1 %) *REL)
         (? 8 *NAME %( %( *P1 %) %))
         (? 7 *CONST)
         (? 4 %( *P2 *OP2)
         (? 5 *IND2)
         (? 6 %( *LISP-OD-INK2 *IND2 %))
         (? 4 %))
```

```
   %))
(CALL (? 7 (CONST-BIND! *CONST)))
=%>
(*LS *VGL LINKS %. (? 1 %( *P1 *OP1)
     (? 2 *IND1)
     (? 3 %( *LISP-OD-INK1 *IND1 %))
     (? 1 %)))
(*LS *VGL RECHTS %. (? 4 %( *P2 *OP2)
     (? 5 *IND2)
     (? 6 %( *LISP-OD-INK2 *IND2 %))
     (? 4 %))
     (? 7 *CONSTBIND)
     (? 8 %( *NAME *P1 %)))
(*LS *VGL REL %. *REL))
```

Bemerk:. 1) Die in der CALL-Voraussetzung aufgerufene Funktion CONST-BIND! transformiert Konstanten wie UNDEF oder TRUE in NIL oder T und legt das Ergebnis in der Bindeliste BINDE unter der Variablen *CONSTBIND ab. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von CONST-BIND!)
2) Die minimale Musterlänge beträgt 5.

■

S2-2

Ziel: Die Regel soll das vorliegende Relationszeichen des Vergleichs erkennen und die rechts und links davon stehenden Terme separieren.

Ergebnis: Auf der dem Indikator VGLj als Wert zugeordneten Assoziationsliste werden Assoc-Paare mit den Schlüsseln "links", "rechts" und "rel" definiert und mit Werten, die die nach LISP transformierten Teilstücke des Vergleichs darstellen, initialisiert.

```
(S2-2 (V *REL (ODER %>= %<= %> %< == /=))
      (V *P1 ATOM)
      (V *CONST ATOM)
      (V *AU1 AUSDRUCK)
      (V *AU2 AUSDRUCK)
      (V *AU3 AUSDRUCK)
      (A %( (? 0 *P1 *REL *CONST)
         (? 9 *P1 %( *AU1 , *AU2 , *AU3 %) *REL *CONST)
         (? 8 *P1 %( *AU1 , *AU2 %) *REL *CONST)
         %))
      (CALL (CONST-BIND! *CONST))
      (CALL (? 9 (ENTKLAMMERUNG 3)))
      (CALL (? 8 (ENTKLAMMERUNG 2)))
      =%>
      (*LS *VGL LINKS %. (? 0 *P1)
           (? 9 %( *P1 *A1 *A2 *A3 %))
           (? 8 %( *P1 *A1 *A2 %)))
```



```
      (*LS *VGL RECHTS %. *CONSTBIND)
      (*LS *VGL REL %. *REL))
```

Bemerk:. 1) Die in zwei CALL-Voraussetzungen aufgerufene Funktion ENTKLAMMERUNG nimmt eine syntaktische Veränderung (Entfernung von für LISP überflüssigen Klammern) an den Werten vor, die an die Variablen *AUi gebunden sind. Die so modifizierten Ausdrücke werden in der Bindeliste BINDE unter den Variablenbezeichnern *Ai abgelegt. Diese Variablen werden in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von ENTKLAMMERUNG)
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion CONST-BIND!: Siehe Bem. 1) von Regel S2-1.
3) Die Regel erkennt in der z.Zt. implementierten Form 2 oder 3 Ausdrücke innerhalb der linken Seite eines Vergleichs.
4) Die minimale Patternlänge ist 5.

■

S2-9

Ziel: Die Regel soll aufbauend auf den bisherigen Extraktionen den Vergleich neu rekonstruieren nach LISP-Konventionen.

Ergebnis: Unter dem Indikator FORM wird auf der Property-Liste von Li LISP-ausführbarer Code für diesen Vergleich abgelegt.

```
(S2-9 (CALL (IN-BINDE! *LS *VGL))
      (CALL (REL-BIND! *REL))
      (CALL (HILFSFKT-VERMERK! S2-9))
      =%>
      (*LS *VGL FORM %. %( *RELBIND *L *R %)>
```

Bemerk:. 1) Die in einer CALL-Voraussetzung aufgerufene Funktion IN-BINDE! kopiert aus den den Indikatoren VGLj zugeordneten Assoziationslisten die Werte der Schlüssel "links", "rechts" und "rel" und legt sie in der Bindeliste BINDE unter den Variablenbezeichnern *L, *R und *REL ab. Diese Variablen werden von der Funktion REL-BIND! und von der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von IN-BINDE!)
2) Die in einer CALL-Voraussetzung aufgerufene Funktion REL-BIND! sucht für das in S2-1 oder S2-2 erkannte Relationszeichen in dem in der Spezifikation angegebenen Relationssatz nach dem zugeordneten Namen und legt ihn in der Bindeliste BINDE unter der Variablen *RELBIND ab. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von REL-

BIND!)
3) Die in der CALL-Voraussetzung aufgerufene Funktion HILFSFKT-VERMERK! sammelt die Namen der in den erzeugten Code durch das System eingebauten Hilfsfunktionen in einer Liste, die Wert des Standardbezeichners HILFSFUNKTIONEN ist. In der vorliegenden Regel wird der aktuelle Relationsname in die Liste übertragen. Die Liste der Hilfsfunktionen wird in der LISP-Phase weiterverwendet. (Siehe auch Funktionsbeschreibung von HILFSFKT-VERMERK!)
4) Die minimale Musterlänge ist 0.

S3-REGELN

Die S3-Regeln besitzen auf Grund ihres zusammenfassenden Charakters keine Muster.

S3-1

Ziel: Die Regel soll alle im 2. Schritt in LISP-Code umgeformten Vergleiche mit der im 1. Schritt erkannten logischen Verknüpfung verbinden.
Ergebnis: Der Indikator FINAL wird auf der Property-Liste von Li initialisiert. Sein Wert ist der Li entsprechende LISP-Code.

```
(S3-1 (E (? 1 *LS LOGOP NONE))
      (E (? 2 *LS LOGOP AND))
      (E (? 3 *LS LOGOP OR))
      (V 4 NICHT 1)
      (CALL (IN-BINDE! *LS))
      =%>
      (*LS FINAL (? 4 (? 2 %( AND))
                 (? 4 (? 3 %( OR))
                 *F1
                 (? 4 *F2 *F3 *F4 *F5 *F6)
                 (? 4 %)))
```

Bemerk:. 1) Die in der CALL-Voraussetzung ohne 2. Parameter aufgerufene Funktion IN-BINDE! kopiert mit Hilfe der Liste, die Wert des Indikators VGLLIST der Property-Liste von Li ist, den für jeden Vergleich VGLj erzeugten Code in die Bindeliste BINDE und legt sie dort unter den Variablenbezeichnern *Fj ab. Diese Variablen werden in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von IN-BINDE!)
2) In der z.Zt. implementierten Form kann die Regel maximal 6 Vergleiche mit der zugehörigen logischen Verknüpfung komponieren.
3) Eine minimale Musterlänge ist nicht angebbar.

Auf Grund der oben gemachten Bemerkung über die Struktur der linken Seiten (reine Konjunktionen oder Disjunktionen) kommt das System im Schritt 3 mit dieser Regel aus. Für weitere Varianten von logischen Verknüpfungen können leicht entsprechende Regeln geschrieben werden.

S4-REGELN

Die Muster der S4-Regeln sind jeweils zum Matching für eine rechte Seite konzipiert.

S4-1

Ziel: Die Regel soll erkennen, ob es sich um einen rekursiven Aufruf des Algorithmus handelt.
Ergebnis: Die Indikator EVALUIERT wird auf der Property-Liste von Ri mit dem Wert "rekursiv" initialisiert, und der globalen Variablen REK, die später von der Funktion LISP-PHASE aufgegriffen wird, weist die Regel den Wert T zu.

```
(S4-1 (A %( *ALGNAME %( ... %) %))
      (CALL (SETQ REK T))
      =%>
      (*RS EVALUIERT REKURSIV))
```

Bemerk:. 1) Die in der CALL-Voraussetzung aufgerufene LISP-Funktion SETQ weist der Globalen Variablen REK den Wert T zu.
2) Die minimale Musterlänge ist 5.

■

S4-2

Ziel: Die Regel soll erkennen, ob es sich um die Übergabe eines formalen Parameters oder um die Übergabe einer Konstanten (z.B. UNDEF) handelt.
Ergebnis: Es wird ein Indikator Form auf der Property-Liste von Ri initialisiert, dessen Wert die erkannte und ggf. leicht modifizierte Struktur ist.
Fallweise wird auch ein Indikator EVALUIERT mit dem Wert "parampass" initialisiert.
Die Bearbeitung der vorliegenden rechten Seite ist damit beendet.

```
(S4-2 (V *ATOM ATOM)
      (A %( *ATOM %))
      (CALL (? 1 (IN-LISTE? *ATOM ALGPARAM)))
      (V 2 NICHT 1)
      (CALL (? 2 (CONST-BIND! *ATOM)))
      =%>
      (*RS FORM (? 1 *ATOM)
```

```
        (? 2 *CONSTBIND))
    ((? 1 *RS EVALUIERT PARAMPASS)))
```

Bemerk:. 1) Die in einer CALL-Voraussetzung aufgerufene Funktion IN-LISTE? überprüft, ob sich das erste übergebene Argument in der Liste befindet, die durch das zweite übergebene Argument bezeichnet ist. Der so ermittelte Wahrheitswert wird an den Index übergeben. (Siehe auch Funktionsbeschreibung von IN-LISTE?)
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion CONST-BIND!: Siehe Bem. 1) von Regel S2-1.
3) Die minimale Musterlänge ist 3.

■

S4-3

Ziel: Die Regel soll erkennen, daß es sich um einen parallel-rekursiven Aufruf handelt.

Ergebnis: Es werden auf der Property-Liste von Ri die Indikatoren EVALUIERT (mit dem Wert "parallel-rekursiv") und NEUE-NAMEN (Wert: Liste der neuen Namen) initialisiert. Da jeder neue Name ein LISP-Atom bezeichnet, werden weiterhin auf den so erreichbaren Property-Listen die Indikatoren EVALUIERT mit dem Wert "fun-arg-rekursion" initialisiert.

```
(S4-3 (V *V1 AUSDRUCK)
      (V *V2 AUSDRUCK)
      (V *V3 AUSDRUCK)
      (V *V4 AUSDRUCK)
      (V *V5 AUSDRUCK)
      (V *V6 AUSDRUCK)
      (A %( *V1 , *V2 (? 3 , *V3)
           (? 4 , *V4)
           (? 5 , *V5)
           (? 6 , *V6)
         %))
      (CALL (? 6 (NEUE-RSEITEN 6 *RS)))
      (CALL (? 5 (NEUE-RSEITEN 5 *RS)))
      (CALL (? 4 (NEUE-RSEITEN 4 *RS)))
      (CALL (? 3 (NEUE-RSEITEN 3 *RS)))
      (CALL (NEUE-RSEITEN 2 *RS))
      =%>
      (*RS EVALUIERT PARALLEL-REKURSIV)
      (*RS NEUE-NAMEN *NEUE-NAMEN)
      (*RS1 EVALUIERT FUN-ARG-REKURSION)
      (*RS2 EVALUIERT FUN-ARG-REKURSION)
      ((? 3 *RS3 EVALUIERT FUN-ARG-REKURSION))
      ((? 4 *RS4 EVALUIERT FUN-ARG-REKURSION))
      ((? 5 *RS5 EVALUIERT FUN-ARG-REKURSION))
      ((? 6 *RS6 EVALUIERT FUN-ARG-REKURSION)))
```

Bemerk:. 1) Die in mehreren CALL-Voraussetzungen aufgerufene

Funktion NEUE-RSEITEN erzeugt gemäß der Anzahl der parallel-rekursiven Aufrufe neue Standardbezeichner, die als Wert die jeweils erkannte Interndarstellung des rekursiven Aufrufs zugewiesen bekommen. Die Liste der neuen Standardbezeichner wird unter dem Variablenbezeichner *NEUE-NAMEN in der Bindeliste BINDE abgelegt.
Der Name des i-ten neuen Standardbezeichners wird unter dem Variablenbezeichner *RSi in der Bindeliste BINDE abgelegt.
Auf alle so definierten Variablenbezeichner wird in der Folgerung der Regel zugegriffen. (Siehe auch Funktionsbeschreibung von NEUE-RSEITEN)
2) In der z.Zt. implementierten Form der Regel werden maximal 6 parallel-rekursive Aufrufe erkannt.
3) Die minimale Musterlänge ist 8.

■

S4-4

Ziel: Die Regel soll erkennen, daß es sich wahlweise um einen (Hilfs)-Algorithmus-Aufruf, einen Inkarnations-Aufruf oder einen LISP-Funktionsaufruf handelt.

Ergebnis: Auf der Property-Liste werden die Indikatoren EVALUIERT (mit einem der Werte "alg-aufruf", "ink-aufruf" oder "lisp-aufruf") und FORM (Wert: die erkannte, nach LISP transformierte Struktur) initialisiert.
Die Bearbeitung dieser linken Seite ist damit abgeschlossen.

```
(S4-4 (V *ATOM ATOM)
      (V *EIG1 ATOM)
      (V *NAME ATOM)
      (V *P ATOM)
      (V *P1 ATOM)
      (V *P2 ATOM)
      (V *OP ATOM)
      (V *EIG ATOM)
      (V *LISP ATOM)
      (V *IND ATOM)
      (A (? 1 %( *NAME %( %( *P %))
         (? 0 , %( *P1 %) , %( *P2 %))
         (? 10 , %( *P1 %))
         (? 1 %) %))
         (? 6 %( %( *ATOM *IND %( *P *OP *EIG %) %) %))
         (? 2 %( %( *P *OP)
         (? 4 *EIG)
         (? 3 *ATOM)
         (? 5 %( *LISP *IND)
         (? 7 *EIG1)
         (? 5 %))
         (? 2 %) %)))
```

```
(CALL (? 1 (IN-LISTE? *NAME HILFSALGORITHMEN)))
(CALL (? 2 (IN-LISTE? *P ALGPARAM)))
=%>
(*RS EVALUIERT (? 1 ALG-AUFRUF)
     (? 2 INK-AUFRUF)
     (? 6 LISP-AUFRUF))
(*RS FORM (? 2 %( *P *OP)
     (? 4 *EIG)
     (? 3 *ATOM)
     (? 5 %( *LISP *IND)
     (? 7 *EIG1)
     (? 5 %))
     (? 2 %))
     (? 1 %( *NAME *P)
     (? 0 *P1 *P2)
     (? 10 *P1)
     (? 1 %))
     (? 6 %( *ATOM *IND %( *P *OP *EIG %) %))))
```

1) Wirkung der in mehreren CALL-Voraussetzungen aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Die minimale Musterlänge ist 6.

■

S4-51

Ziel: Die Regel soll mit den vorgegebenen Mustern erkennen, daß ein formaler Parameter mehrmals auftritt und kopiert werden muß.

Ergebnis: Auf der Property-Liste von Ri werden (fallweise) die Indikatoren KOPIERT2 oder KOPIERT3 initialisiert, deren Werte die Namen der zu kopierenden formalen Parameter sind.

```
(S4-51 (V *PAR ATOM)
       (V *QPAR ATOM)
       (A %( *ALGNAME %( (? 1 %( %( *PAR ... %) %))
          ... , (? 3 %( %( *PAR ... %) %))
          (? 4 %( %( ... %( *PAR ... %) %) %))
          (? 5 %( %( ... %( %( *QPAR ... %) %) %) %))
          (? 6 ... , %( %( ... %( *PAR ... %) %) %))
          ... %) %))
       (CALL (? 71 (EQ-KWOTE? *PAR *QPAR)))
       <CALL (? 7 (NOT (INDEX-CHECK 1 71 5>
       <CALL (? 8 (NOT (INDEX-CHECK 1 6>
       <CALL (? 9 (NOT (INDEX-CHECK 3 6>
       <CALL (? 10 (NOT (INDEX-CHECK 1 4>
       (CALL (NOT-ALL-NIL? 7 8 9 10))
       =%>
       ((? 7 *RS KOPIERT2 *QPAR))
       ((? 8 *RS KOPIERT3 *PAR))
       ((? 9 *RS KOPIERT3 *PAR))
       ((? 10 *RS KOPIERT2 *PAR)))
```


Bemerk:.
1) Die in einer CALL-Voraussetzung aufgerufene Funktion EQ-KWOTE überprüft, ob es sich bei *PAR und *QPAR um den gleichen Parameter handelt, nur im zweiten Fall gekwotet. Der so ermittelte Wert wird an den entsprechenden Index als Wert übergeben. (Siehe auch Funktionsbeschreibung von EQ-KWOTE)
2) Die in mehreren CALL-Voraussetzungen aufgerufene Funktion INDEX-CHECK überprüft für die ihr als Argument übergebene Liste von Indices, ob von den daran in der Bindeliste BINDE gebundenen Werten mindestens einer ≠ NIL ist. Der Wert der Funktion ist dann T, andernfalls NIL, und er wird an den zugeordneten Index übergeben. (Siehe auch Funktionsbeschreibung von INDEX-CHECK.)
3) Die in einer CALL-Voraussetzung aufgerufene Funktion NOT-ALL-NIL? überprüft für die ihr als Argument übergebene Liste von Indices, ob mindestens einer der an diese Indices gebundenen Werte T ist. Der Wert der Funktion ist dann T, andernfalls NIL, und er wird an die aufrufende Funktion übergeben. (Siehe auch Funktionsbeschreibung von NOT-ALL-NIL?)
4) Die minimale Musterlänge ist 21.

■

S4-52

Ziel: Die Regel soll mit den vorgegebenen Mustern erkennen, daß ein formaler Parameter mehrmals auftritt und kopiert werden muß.

Ergebnis: Auf der Property-Liste werden (fallweise) die Indikatoren KOPIERT1 oder KOPIERT3 initialisiert, deren Werte die Namen der zu kopierenden formalen Parameter sind.

```
(S4-52 (V *PAR ATOM)
       (V *A ATOM)
       (V *B ATOM)
       (V *C ATOM)
       (A %( *ALGNAME %(
          (? 1 %( *A %( %( ... %) ,
               %( %( *PAR ... %) %) %) %))
          ... , (? 2 %( %( *C *PAR %) %))
          ... , (? 3 %( %( ... %( *PAR ... %) %) %))
          (? 4 %( %( *B)
          (? 7 *D)
          (? 4 *PAR %) %))
          ... %) %))
       <CALL (? 5 (NOT (INDEX-CHECK 1 2>
       <CALL (? 6 (NOT (INDEX-CHECK 1 3>
       (CALL (NOT-ALL-NIL? 5 6))
       =%>
       ((? 5 *RS KOPIERT1 *PAR))
       ((? 6 *RS KOPIERT3 *PAR)))
```

Bemerk:. 1) Wirkung der in mehreren CALL-Voraussetzungen aufgerufenen Funktion INDEX-CHECK: Siehe Bem. 2) von Regel S4-51.
2) Die minimale Musterlänge ist 28.

■

S4-53

Ziel: Die Regel soll mit Hilfe der vorgegebenen Muster die Vorentscheidung treffen, daß ein formaler Parameter eventuell kopiert werden muß.

Ergebnis: Auf der Property-Liste von Ri wird der Indikator KOPIERTj-EVTL initialisiert, dessen Wert der Name des betreffenden formalen Parameters ist. Diese Information greift Regel S5-1 auf.

```
(S4-53 (V *A ATOM)
       (V *B ATOM)
       (V *C ATOM)
       (V *PAR ATOM)
       (V *NAME ATOM)
       (A %( *ALGNAME %( ... , ... , ... , %( %( *NAME
         %( *A *B *PAR %) *C %) %) , %( *PAR %) %) %))
       (CALL (IN-LISTE? *PAR ALGPARAM))
       =%>
       (*RS KOPIERT5-EVTL *PAR))
```

Bemerk.: 1) Wirkung der in der CALL-Voraussetzung aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Die minimale Musterlänge ist 23.

■

S4-54

Ziel: Die Regel soll überprüfen, ob die Reihenfolge von formalen Parametern in der vorliegenden rechten Seite vertauscht ist und ob deswegen entsprechend kopiert werden muß.

Ergebnis: Auf der Property-Liste von Ri wird (fallweise) der Indikator KOPIERT2 oder KOPIERT3 initialisiert, dessen Wert der Name des zu kopierenden formalen Parameters ist.

```
(S4-54 (V *A ATOM)
       (V *B ATOM)
       (V *C ATOM)
       (A %( *ALGNAME %(
          (? 1 %( ... %) , %( *B %) , %( *C %) , %( ... %))
          (? 2 %( *A %) , %( *B %) , %( ... %))
          %) %))
       <CALL (? 2 (PARAMETER-REIHENFOLGE?
```



```
                    (QUOTE (*A 1 *B 2>
       <CALL (? 1 (PARAMETER-REIHENFOLGE?
                    (QUOTE (*B 2 *C 3>
       (CALL (NOT-ALL-NIL? 1 2))
       =%>
       ((? 1 *RS KOPIERT3 *C))
       ((? 2 *RS KOPIERT2 *B)))
```

Bemerk:. 1) Die in mehreren CALL-Voraussetzungen aufgerufene Funktion PARAMETER-REIHEFOLGE? prüft für die ihr als Argument übergebene Liste, ob die dort vermerkten formalen Parameter mit ihren aktuellen Argumentpositionen in einer anderen Reihenfolge stehen als in der Parameterliste der Spezifikation (siehe dazu auch 4.1.3.). Falls die Reihenfolge für einen formalen Parameter vertauscht ist, liefert die Funktion den Wahrheitswert T, anderfalls NIL, und übergibt ihn an den entsprechenden Index als Wert. (Siehe auch Funktionsbeschreibung von PARAMETER-REIHENFOLGE?)
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion NOT-ALL-NIL?: Siehe Bem. 3) von Regel S4-51.
3) Die minimale Musterlänge ist 15.

■

S4-55

Ziel: Die Regel soll mit den vorgegebenen Mustern erkennen, daß ein formaler Parameter mehrmals auftritt und kopiert werden muß.

Ergebnis: Auf der Property-Liste werden die Indikatoren KOPIERT3 und (fallweise) KOPIERT4 initialisiert, deren Werte die Namen der zu kopierenden formalen Parameter sind.

```
(S4-55 (V *A1 ATOM)
       (V *P1 ATOM)
       (A %( *ALGNAME %( ... , %( %( *A1 *P1 %) %) ,
            %( %( ... *P1 %) %) (? 1 , %( *P1 %))
          %) %))
       (CALL (IN-LISTE? *P1 ALGPARAM))
       =%>
       (*RS KOPIERT3 *P1)
       ((? 1 *RS KOPIERT4 *P1)))
```

Bemerk.: 1) Wirkung der in der CALL-Voraussetzung aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Die minimale Musterlänge ist 18.

■

S4-9

Ziel: Die Regel soll die Argumente eines rekursiven Auf-

rufs extrahieren.

Ergebnis: Es werden auf der Property-Liste von Ri die Indikatoren ARGj für jedes erkannte Argument initialisiert. Werte sind Assoziationslisten, die unter dem Schlüssel "Expr" die Interndarstellung des jeweiligen Arguments enthalten.
Weiter wird ein Indikator ARGLIST initialisiert, dessen Wert eine Liste der den Argumenten zugeordneten Indikatoren ARGj ist.

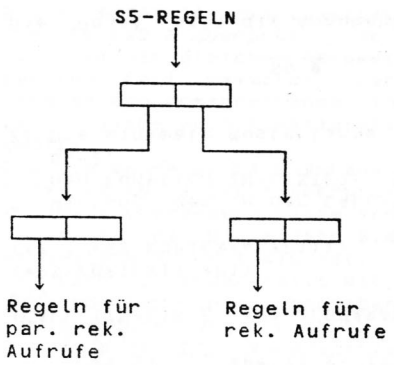
```
(S4-9 (V *A1 AUSDRUCK)
      (V *A2 AUSDRUCK)
      (V *A3 AUSDRUCK)
      (V *A4 AUSDRUCK)
      (V *A5 AUSDRUCK)
      (V *A6 AUSDRUCK)
      (V *A7 AUSDRUCK)
      (A %( *ALGNAME %( *A1 (? 1 , *A2)
         (? 2 , *A3)
         (? 3 , *A4)
         (? 4 , *A5)
         (? 5 , *A6)
         %) %))
      =%>
      (*RS ARG1 EXPR %. *A1)
      ((? 1 *RS ARG2 EXPR %. *A2))
      ((? 2 *RS ARG3 EXPR %. *A3))
      ((? 3 *RS ARG4 EXPR %. *A4))
      ((? 4 *RS ARG5 EXPR %. *A5))
      ((? 5 *RS ARG6 EXPR %. *A6))
      ((? 6 *RS ARG7 EXPR %. *A7))
      (*RS ARGLIST ARG1 (? 1 ARG2)
           (? 2 ARG3)
           (? 3 ARG4)
           (? 4 ARG5)
           (? 5 ARG6)
           (? 6 ARG7>
```

Bemerk.: 1) In der z.Zt. implementierten Form erkennt die Regel maximal 7 Argumente eines rekursiven Aufrufs.
2) Die minimale Musterlänge ist 8.

■

S5-REGELN

Die S5-Regelbasis ist in einer anderen Listenstruktur organisiert als die übrigen Basen. Unterschieden wird der Teil der Regeln, der für die Bearbeitung der Argumente (einfach)- rekursiver Aufrufe verwendet wird, von dem Teil, der zur Bearbeitung von Argumenten rekursiver Aufrufe verwendet wird, die innerhalb einer parallel-rekursiven rechten Seite auftreten. Diese Aufteilung ist in der unterschiedlichen Zielstruktur für parallel-rekursive und nicht-parallel-rekursive rechte Seiten begründet (siehe 4.1.1. und 4.4.4.). Die realisierte Listenstruktur ist folgende:



Die richtige Auswahl der anwendbaren Regelmenge wird durch die Funktion SCHRITT-5 geleistet (siehe Funktionsbeschreibung SCHRITT-5).

Die Muster der S5-Regeln sind jeweils zum Matching eines Arguments eines rekursiven Aufrufs konzipiert.

S5-D1

Zweck: Die Regel soll die Argumente eines rekursiven Aufrufs, der innerhalb einer parallel-rekursiven rechten Seite auftritt, geeignet modifizieren.

Ergebnis: Auf der Property-Liste von Ri wird in der dem Indikator ARGj zugeordneten Assoziationsliste der Schlüssel "Form" definiert. Sein Wert ist das geeignet (bzgl. der angestrebten Zielstruktur) umgeformte Argument in Interndarstellung.

```
(S5-D1 (V *A ATOM)
       (V *B ATOM)
       (A %( (? 1 *A)
          (? 2 %( *A *B %))
          %))
       (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                              ALGPARAM))
       (CALL (? 4 (GLEICH *APB *A)))
       (V 5 NICHT 4)
       (CALL (? 6 (GLEICH *A FALSE)))
       (V 3 NICHT 2)
       (V 7 NICHT 6)
       =%>
       (*RS *ARG FORM %. (? 2 %( *APB %( *A *B %) %))
            (? 4 (? 3 NIL))
            (? 5 (? 3 (? 7 %( *APB *A %))))
            (? 6 (? 3 *APB))))
```

Bemerk:. 1) Die in einer CALL-Voraussetzung aufgerufene Funktion ARG-PARAM-BIND! bewirkt, daß der der aktuel-

len Argumentposition zugeordnete formale Parameter in der Bindeliste BINDE unter dem Variablenbezeichner *APB abgelegt wird. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von ARG-PARAM-BIND!)

2) Die in mehreren CALL-Voraussetzungen aufgerufene Funktion GLEICH führt eine Überprüfung der übergebenen Argumente auf Gleichheit durch. Der so ermittelte Wahrheitswert wird als Wert an den entsprechenden Index übergeben.
   GLEICH ist eine Funktion des ADTCOMP - Systems. (Siehe auch Funktionsbeschreibung von GLEICH)
3) Die minimale Musterlänge ist 14.

■

S5-1

Zweck: Die Funktion soll eine SETQ-Form für ein atomares Argument erzeugen, das eventuell durch eine Kopie substituiert wird.

Ergebnis: Es sind 5 Fälle zu unterscheiden:
Das atomare Argument *CONST erfüllt (exklusiv)

a) *RS KOPIERT5-EVTL *CONST
b) *RS KOPIERT2 *CONST
c) *RS KOPIERT3 *CONST
d) *RS KOPIERT4 *CONST
e) weder a), b), c) noch d).

a: Die Vorentscheidung über eine Kopie wird bearbeitet. Wenn *APB (der der aktuellen Argumentposition zugeordnete formale Parameter / siehe Funktion ARG-PARAM-BIND!) gleich *CONST ist, d.h. eine Form der Art (SETQ A A) entsteht, wird nicht kopiert. Diese redundante Struktur wird später entfernt.
   Falls *APB ungleich *CONST ist, also eine Form der Art (SETQ A B) erzeugt werden soll, wird B durch den Standardbezeichner X5 substituiert.

b: Die Entscheidung, *CONST durch eine Kopie zu ersetzen, wird nur dann in eine entsprechende Substitution umgewandelt, wenn die aktuelle Argumentposition 2 ist (das 2. Argument des Aufrufs vorliegt).
   Andernfalls keine Kopiensubstitution.

c: Die Entscheidung, *CONST durch eine Kopie zu ersetzen, wird nur dann in eine entsprechende Substitution umgewandelt, wenn die aktuelle Argumentposition 3 ist.
   Andernfalls keine Kopiensubstitution.

d: Die Entscheidung, *CONST durch eine Kopie zu ersetzen, wird nur dann in eine entsprechende Substitution umgewandelt, wenn die aktuelle Argumentposition 4 ist.
   Andernfalls keine Kopiensubstitution.

e: *CONST wird lediglich der Behandlung durch die

Funktion CONST-BIND! unterzogen.

In allen Fällen wird auf der dem Indikator ARGj zugeordneten Assoziationsliste der Wert des Schlüssels "Expr" neu definiert durch die in der Regelfolgerung angegebene Form.
Zusätzlich wird bei Vorliegen des Falles a) der Indikator KOPIERT5 auf der Property-Liste von Ri mit dem Wert *CONST initialisiert.

```
(S5-1 (V *CONST ATOM)
      (A %( *CONST %))
   a  (E (? 1 *RS KOPIERT5-EVTL *CONST))
      (V 2 NICHT 1)
      (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                             ALGPARAM))
      <CALL (? 3 (NOT (GLEICH *APB *CONST>
      (CALL (? 2 (CONST-BIND! *CONST)))
      (V 5 NICHT 3)
   b  (E (? 6 *RS KOPIERT2 *CONST))
      (CALL (? 8 (ARGNR-OK? 2 *ARG)))
      <CALL (? 9 (NOT (INDEX-CHECK 6 8>
      (V 7 NICHT 9)
   c  (E (? 10 *RS KOPIERT3 *CONST))
      (CALL (? 11 (ARGNR-OK? 3 *ARG)))
      <CALL (? 12 (NOT (INDEX-CHECK 10 11>
      (V 13 NICHT 12)
   d  (E (? 14 *RS KOPIERT4 *CONST))
      (CALL (? 15 (ARGNR-OK? 4 *ARG)))
      <CALL (? 16 (NOT (INDEX-CHECK 14 15>
      (V 17 NICHT 16)
      =%>
      ((? 1 (? 3 *RS KOPIERT5 *CONST)))
      (*RS *ARG EXPR %. %( SETQ *APB (? 1 (? 3 X5))
           (? 5 (? 1 *CONST))
           <? 2 (? 7 (? 13 (? 17 *CONSTBIND>
           (? 9 X2)
           (? 12 X3)
          (? 16 X4)
          %)))
```

Bemerk:. 1) Die in mehreren CALL-Voraussetzungen aufgerufene Funktion ARGNR-OK? überprüft, ob die aktuelle Argumentposition gleich der ihr als Argument übergebenen Zahl ist. In diesem Fall ist der Wert T, anderfalls NIL, und dieser Wahrheitswert wird an den entsprechenden Index als Wert übergeben. (Siehe auch Funktionsbeschreibung von ARGNR-OK?)
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion CONST-BIND!: Siehe Bem. 1) von Regel S2-1.
3) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
4) Wirkung der in einer CALL-Voraussetzung aufge-

rufenen Funktion GLEICH: Siehe Bem. 2) von Regel S5-D1.
5) Wirkung der in mehreren CALL-Voraussetzungen aufgerufenen Funktion INDEX-CHECK: Siehe Bem. 2) von Regel S4-51.
6) Die minimale Musterlänge ist 3.

■

S5-2

Ziel: Die Regel soll erkennen, ob es sich bei dem betrachteten Argument um einen LISP-Funktionsaufruf handelt und ggf. eine Kopienentscheidung treffen.

Ergebnis: Auf der dem Indikator ARGj zugeordneten Assoziationsliste wird unter dem Schlüssel "lisp-aufruf" eine Liste abgelegt, die den erkannten LISP-Funktionsaufruf in interner Darstellung enthält. Fallweise wird noch ein Schlüssel "kopiert-lisp" initialisiert.

```
(S5-2 (V *LISP ATOM)
      (A %( %( *LISP ... %) %))
      (CALL (NOT (IN-LISTE? *LISP ALGPARAM)))
      (CALL (LISPNAMEN-VERMERK! *LISP))
      (CALL (? 1 (GREATERP (LENGTH LISPNAMEN)
                           1)))
      =%>
      (*RS *ARG LISP-AUFRUF *LISP)
      ((? 1 *RS *ARG KOPIERT-LISP)))
```

Bemerk.:
1) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Die in einer CALL-Voraussetzung aufgerufene Funktion LISPNAMEN-VERMERK! vermerkt den erkannten Funktionsnamen in einer Liste, die unter dem Bezeichner LISPNAMEN referierbar ist.
   Falls mehr als ein LISP-Funktionsaufruf in einer rechten Seite auftritt, werden Kopienentscheidungen getroffen. (Siehe auch Funktionsbeschreibung von LISPNAMEN-VERMERK!)
3) Die minimale Musterlänge ist 5.

■

S5-31

Ziel: Die Regel soll eine bestimmte Klasse von Konstrukten erkennen, formale Parmeter ggf. durch Kopien ersetzen und eine SETQ-Form aufbauen.

Ergebnis: Auf der dem Indikator ARGj zugeordneten Assoziationsliste wird der Wert des Schlüssels "Expr" neu definiert als SETQ-Form mit eventuell substituierten formalen Parametern und Syntaxmodifikationen.
Dabei sind 4 Fälle zu unterscheiden:

a) Ein formaler Parameter (Inkarnation) soll im 2. Argument substituiert werden.
b) Ein formaler Parameter (Inkarnation) soll im 3. Argument substituiert werden.
c) Ein formaler Parameter (Index) soll im 3. Argument substituiert werden.
d) Weder a), b) noch c).

a: Der Indikator KOPIERT2 wird betrachtet und sein Wert mit dem an *P2 gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 2, wird in der SETQ-Form *P2 durch X2 ersetzt.
b: Der Indikator KOPIERT3 wird betrachtet und sein Wert mit dem an *P3 gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 3, wird in der SETQ-Form *P3 durch X3 ersetzt.
c: Der Indikator KOPIERT3 wird betrachtet und sein Wert mit dem an *IND1 gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 3, wird in der SETQ-Form *IND1 durch X3 ersetzt.
d: Es werden keine Substitutionen durchgeführt.

```
(S5-31 (V *P1 ATOM)
       (V *P2 ATOM)
       (V *PAR ATOM)
       (V *OP1 ATOM)
       (V *OP2 ATOM)
       (V *LISP-OD-INK ATOM)
       (V *IND2 ATOM)
       (V *IND-OD-OP ATOM)
       (V *IND1 ATOM)
       (V *LISP2 ATOM)
       (A %( (? 1 %( *P1 *OP1)
          (? 2 *IND1)
          (? 3 %( *LISP-OD-INK *IND-OD-OP %))
          (? 4 %( *P2 *OP2)
          (? 5 *IND2)
          (? 6 %( *LISP2 *IND2 %))
          (? 4 %))
          (? 1 %))
          %))
       (V 0 NICHT 4)
       <CALL (? 0 (? 3 (UMBENENNUNG! 1
                        *LISP-OD-INK *IND-OD-OP>
   a   [ (E (? 7 *RS KOPIERT2 *P2))
       [ (CALL (? 8 (ARGNR-OK? 2 *ARG)))
       [ (CALL (? 11 (INDEX-CHECK 7 8)))
   b   [ (E (? 9 *RS KOPIERT3 *P2))
       [ (CALL (? 10 (ARGNR-OK? 3 *ARG)))
       [ (CALL (? 12 (INDEX-CHECK 9 10)))
       [ (E (? 91 *RS KOPIERT3 *IND1))
       [ (CALL (? 92 (ARGNR-OK? 3 *ARG)))
```

```
c | <CALL (? 93 (NOT (INDEX-CHECK 91 92>
  | (V 94 NICHT 93)
  | (CALL (? 13 (NOT-ALL-NIL? 3 4 5 6)))
  | (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
  |                        ALGPARAM))
  | =%>
  | (*RS *ARG EXPR %. %( SETQ *APB (? 1 %( *P1 *OP1)
  |       (? 2 (? 94 *IND1))
  |       (? 2 (? 93 X3))
  |       (? 3 (? 4 %( *LISP-OD-INK *IND-OD-OP %)))
  |       (? 7 (? 8 %( X2))
  |       (? 9 (? 10 (? 93 %( X3)))
  |       (? 9 (? 10 (? 94 %( X3)))
  |       (? 11 (? 12 (? 13 %( *P2)))
  |       (? 0 (? 13 *OP2))
  |       (? 4 *OP2)
  |       (? 5 *IND2)
  |       (? 6 %( *LISP2 *IND2 %))
  |       (? 4 %))
  |       (? 0 (? 13 %)))
  |       (? 1 %))
  |       %)))
```

Bemerk:.

1) Die in der CALL-Voraussetzung aufgerufene Funktion UMBENENNUNG dient zur Manipulation von Variablen der Bindeliste BINDE.
   Falls das Muster mit dem Index 4 nicht gebunden ist, wohl aber das Muster mit Index 3, werden die Werte der Bezeichner *LISP-OD-INK und *IND-OD-OP an die Bezeichner *P2 und *OP2 in BINDE gehängt. Das erspart im weiteren Ablauf ein Betrachten und Bearbeiten der Werte der zuerst genannten Variablenbezeichner. (Siehe auch Funktionsbeschreibung von UMBENENNUNG)
2) Wirkung der in mehreren CALL-Voraussetzungen aufgerufenen Funktion ARGNR-OK?: Siehe Bem. 1) von Regel S5-1.
3) Wirkung der in mehreren CALL-Voraussetzungen aufgerufenen Funktion INDEX-CHECK: Siehe Bem. 2) von Regel S4-51.
4) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion NOT-ALL-NIL?: Siehe Bem. 3) von Regel S4-51.
5) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
6) Die minimale Musterlänge ist 5.

■

S5-32

Ziel: Die Regel soll die durch das Muster angegebene Klasse von Konstrukten erkennen, gegebenenfalls eine Substitution durchführen und eine SETQ-Form aufbauen.

Ergebnis: Es sind 2 Falle zu unterscheiden:

a) Der formale Parameter *IND soll im 4. Argument kopiert werden.
b) Nicht a).

a: Der Indikator KOPIERT3 wird betrachtet und sein Wert mit dem an *IND gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 3, wird in der SETQ-Form *IND durch X3 ersetzt.
b: Es wird keine Substitution durchgeführt.

In beiden Fällen wird auf der dem Indikator ARGj zugeordneten Assoziationsliste der Schlüssel "Expr" neu definiert als SETQ-Form mit dem erkannten und modifizierten Argument.

```
(S5-32 (V *A1 ATOM)
       (V *A2 ATOM)
       (V *IND ATOM)
       (A %( %( *A1 *A2 (? 1 *IND)
          %) %))
       (E (? 2 *RS KOPIERT3 *IND))
       (CALL (IN-LISTE? *A1 ALGPARAM))
       (CALL (? 3 (ARGNR-OK? 3 *ARG)))
       <CALL (? 4 (NOT (GLEICH *A1 SETQ>
       (CALL (? 6 (INDEX-CHECK 2 3)))
       (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                              ALGPARAM))
       (V 7 NICHT 4)
       =%>
       (*RS *ARG EXPR %. (? 4 %( SETQ *APB)
            (? 7 %()
            %( *A1 *A2 (? 1 (? 2 (? 3 X3)))
            (? 1 (? 6 *IND))
            %) %)))
```

Bemerk.:

1) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARGNR-OK?: Siehe Bem. 1) von Regel S5-1.
3) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion GLEICH: Siehe Bem. 2) von Regel S5-D1.
4) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion INDEX-CHECK: Siehe Bem. 2) von Regel S4-51.
5) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
6) Die minimale Musterlänge ist 6.

■

S5-33

Ziel: Die Regel soll die durch das Muster angegebene Klasse von Konstrukten erkennen und ggf. Substitutionen durchführen.

Ergebnis: Es sind 2 Fälle zu unterscheiden:

a) Der formale Parameter *PAR soll im 2. Argument kopiert werden.
b) Nicht a).

a: Der Indikator KOPIERT2 wird betrachtet und sein Wert mit dem an *IND gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 2, wird in der SETQ-Form *PAR durch X2 ersetzt.
b: Es wird keine Substitution durchgeführt.

In beiden Fällen wird auf der dem Indikator ARGj zugeordneten Assoziationsliste der Schlüssel "Expr" neu definiert als SETQ-Form mit dem erkannten und modifizierten Argument.

```
(S5-33 (V *P ATOM)
       (V *OP ATOM)
       (V *IND ATOM)
       (V *ALG ATOM)
       (V *PAR ATOM)
       (A %( %( *P *OP (? 8 *ALG %()
          (? 9 *IND *ALG %()
          %( *PAR %) %) %) %))
       (CALL (IN-LISTE? *ALG HILFSALGORITHMEN))
       (CALL (? 1 *RS KOPIERT2 *PAR))
       (CALL (? 2 (ARGNR-OK? 2 *ARG)))
       <CALL (? 3 (NOT (INDEX-CHECK 1 2>
       (V 4 NICHT 3)
       (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                              ALGPARAM))
       =%>
       (*RS *ARG EXPR %. %( SETQ *APB
                %( *P *OP (? 9 *IND %( *ALG)
            (? 8 %( *ALG)
            (? 3 %'X2)
            (? 4 *PAR)
            %) %) %)))
```

Bemerk.:
1) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion IN-LISTE?: Siehe Bem. 1) von Regel S4-2.
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARGNR-OK?: Siehe Bem. 1) von Regel S5-1.
3) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion INDEX-CHECK: Siehe Bem. 2) von Regel S4-51.
4) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
5) Die minimale Musterlänge ist 10

S5-4

Ziel: Die Regel soll die durch das Muster angegebene Klasse von Konstrukten erkennen und ggf. Substitutionen durchführen.

Ergebnis: Es sind 2 Fälle zu unterscheiden:

a) Der formale Parameter *IND soll im 1. Argument kopiert werden.
b) Nicht a).

a: Der Indikator KOPIERT1 wird betrachtet und sein Wert mit dem an *IND gebundenen verglichen. Stimmen sie überein und ist die aktuelle Argumentposition 1, wird in der SETQ-Form *IND durch X1 ersetzt.
b: Es wird keine Substitution durchgeführt.

In beiden Fällen wird auf der dem Indikator ARGj zugeordneten Assoziationsliste der Schlüssel "Expr" neu definiert als SETQ-Form mit dem erkannten und modifizierten Argument.

```
(S5-4  (V *NAME ATOM)
       (V *P ATOM)
       (V *P1 ATOM)
       (V *P2 ATOM)
       (V *OP ATOM)
       (V *IND ATOM)
       (V *LISP ATOM)
       (V *IND1 ATOM)
       (V *IND2 ATOM)
       (V *PO ATOM)
       (V *OPO ATOM)
       (V *INDO ATOM)
       (V *OPOO ATOM)
       (A %( (? 0 %( *PO *OPO *INDO)
          *NAME %( (? 7 %( *P %))
          (? 6 %( %( *P *OP2 *IND2 %) %))
          (? 1 , %( *IND %))
          (? 2 , %( %( *P1 *OP *IND %) %))
          (? 3 , %( %( *LISP *IND1 %) %))
          (? 9 , %( *OPOO %))
          (? 0 %))
          %) %))
       (E (? 4 *RS KOPIERT1 *IND))
       (V 5 NICHT 4)
       (CALL (NULL (GLEICH *NAME *ALGNAME)))
       (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                              ALGPARAM))
       =%>
```

```
(*RS *ARG EXPR %. %( SETQ *APB %(
    (? 0 *P0 *OP0 *IND0 %()
    *NAME
    (? 7 *P)
    (? 6 %( *P *OP2 *IND2 %))
    (? 1 (? 4 X1))
    (? 1 (? 5 *IND))
    (? 2 %( *P1 *OP *IND %))
    (? 3 %( *LISP *IND1 %))
    (? 9 *OP00)
     (? 0 %))
    %) %)))
```

Bemerk.: 1) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion GLEICH: Siehe Bem. 2) von Regel S5-D1.
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
3) Die minimale Musterlänge ist 12.

■

S5-5

Ziel: Die Regel soll LISP-Funktionsaufrufe bearbeiten, ggf. Kopien für formale Parameter substituieren und eine SETQ-Form aufbauen.

Ergebnis: Zwei Fälle sind zu unterscheiden:

a) Es existiert auf der dem Indikator ARGj zugeordneten Assoziationsliste ein Schlüssel "kopiert-lisp".
b) Nicht a).

a: Es werden alle formalen Parameter des Algorithmus, die in dem Funktionsaufruf auftreten, durch Kopien ersetzt.
b: Der LISP-Funktionsaufruf wird, ggf. syntaxmodifiziert, ohne Kopien übernommen.

In beiden Fällen wird auf der ARGj zugeordneten Assoziationsliste der Wert des Schlüssels "Expr" neu definiert durch eine SETQ-Form mit dem bearbeiteten LISP-Funktionsaufruf.

Falls kopiert wird, wird auf der Property-Liste von R1 der Indikator KOPIERT-LISP initialisiert. Sein Wert ist eine Assoziationsliste der Form ((Schlüssel . Liste) ... (Schlüssel . Liste)), wobei "Schlüssel" ein Indikator ARGj ist und "Liste" eine Liste von Paaren (FP K), die die durch Kopien K substituierten formalen Parameter FP bezeichnen.

```
<S5-5 (E (? 0 *RS *ARG KOPIERT-LISP))
      (V 9 NICHT 0)
```



```
      (V *LISP1 ATOM)
      (V *LISP3 ATOM)
      (V *LISP2 ATOM)
      (V *A1 ATOM)
      (V *A2 ATOM)
      (V *A3 ATOM)
      (V *A4 ATOM)
      (A %( %( *LISP1 (? 1 *A1)
         (? 2 %( *LISP2 *A2 *A3 %))
         (? 4 %( *LISP2 *A2 %( *LISP3 *A3 %) %))
         (? 5 %( *LISP2 %( *LISP3 *A3 %) *A2 %))
         (? 3 *A4)
        %) %))
      (CALL (? 0 (STANDARD-BIND! 4 *ARG)))
      (CALL (ARG-PARAM-BIND! (QUOTE *ARG)
                             ALGPARAM))
      =%>
      (*RS *ARG FORM %. %( SETQ *APB %( *LISP1
         (? 1 (? 0 *STD1))
         (? 1 (? 9 *A1))
         (? 2 (? 0 %( *LISP2 *STD2 *STD3 %)))
         (? 2 (? 9 %( *LISP2 *A2 *A3 %)))
         (? 4 (? 0 %( *LISP2 *STD2 %( *LISP3 *STD3 %) %)))
         (? 4 (? 9 %( *LISP2 *A2 %( *LISP3 *A3 %) %)))
         (? 5 (? 9 %( *LISP2 %( *LISP3 *A3 %) *A2 %)))
         (? 5 (? 0 %( *LISP2 %( *LISP3 *STD3 %) *STD2 %)))
         (? 3 (? 0 *STD4))
         (? 3 (? 9 *A4))
              %) %))
      ((? 0 *RS KOPIERT-LISP *ARG (? 1 %( *A1 *STD1 %))
          (? 2 %( *A2 *STD2 %) %( *A3 *STD3 %))
          (? 3 %( *A4 *STD4 %))
          (? 4 %( *A3 *STD3 %) %( *A2 *STD2 %))
          (? 5 %( *A3 *STD3 %) %( *A2 *STD2 %)>
```

Bemerk:. 1) Die in einer CALL-Voraussetzung aufgerufene Funktion STANDARD-BIND! erzeugt im Falle von zu kopierenden formalen Parametern FPj neue Standardbezeichner, die in der Bindeliste BINDE unter der Variablen *STDj abgelegt werden. Diese Variablen werden in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von STANDARD-BIND!)
2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion ARG-PARAM-BIND!: Siehe Bem. 1) von Regel S5-D1.
3) Die minimale Musterlänge ist 6.

■

S5-8

Ziel: Die Regel soll den bisher erzeugten Code bereinigen, indem Zuweisungen an einen Inkarnationsbezeichner wieder rückgangig gemacht werden, da ein Inkarnationsaufruf eine derartige Zuweisung bereits beinhaltet.

Ergebnis: Auf der ARGj zugeordneten Assoziationsliste wird der Schlüssel "form" initialisiert. Sein Wert ist der so bereinigte Code.

```
(S5-8 (V *P ATOM)
      (V *L LISTE)
      (A %( SETQ *P %( *P *L %) %))
      =%>
      (*RS *ARG FORM %. %( *P *L %)))
```

Bemerk.: 1) Die minimale Musterlänge ist 8.

■

S5-9

Ziel: Die Regel soll bisher erzeugten Code bereinigen von Zuweisungen des Wertes NIL, von Zuweisungen des bisherigen Wertes, und soll Initialisierungszuweisungen für Inkarnationen (SETQ A LEERES-EXEMPLAR) geeignet in LISP-Code transformieren.

Ergebnis: Auf der dem Indikator ARGj zugeordneten Assoziationsliste wird der Schlüssel "form" initialisiert. Sein Wert ist der so modifizierte Code.

```
(S5-9 (V *P ATOM)
      (A %( SETQ *P (? 1 NIL %))
         (? 2 *P %))
         (? 5 LEERES-EXEMPLAR %)))
      (CALL (? 3 (INKARNATION-? *P)))
      =%>
      (*RS *ARG FORM %. (? 1 %( SETQ *P)
           (? 3 (? 5 %( SETQ *P *INKBIND %)))
           (? 1 %)>
```

Bemerk:. 1) Die in der CALL-Voraussetzung aufgerufene Funktion INKARNATION-? überprüft, ob es sich bei dem ihr übergebenen Argument um eine Inkarnation eines vom ADTCOMP-System implementierten Datentyps handelt. Falls ja, so wird unter dem Variablenbezeichner *INKBIND in der Bindeliste BINDE ein Codestück der Form (DT DO) abgelegt, wobei DT den zugrundeliegenden Datentyp und DO die Initialisierungsfunktion bezeichnen. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von INKARNATION-?)

2) Die minimale Musterlänge ist 4.

■

S6-REGELN

Auf Grund ihres zusammenfassenden Charakters haben S6-Regeln kein Muster.

S6-1

Ziel: Die Regel soll DUPLIZIERE-Formen erzeugen und sie mit dem bisher erzeugten Code für eine rechte Seite komponieren.

Ergebnis: In Abhängigkeit der Werte der Indikatoren KOPIERTj und KOPIERT-LISP werden DUPLIZIERE-Formen, bereits existierender Code und ein Zwischen-Code für den rekursiven Aufruf unter dem Indikator FINAL der Property-Liste von Ri abgelegt.

```
(S6-1 (V *P0 ATOM)
      (V *P ATOM)
      (V *P1 ATOM)
      (V *P2 ATOM)
      (V *P4 ATOM)
      (E *RS EVALUIERT REKURSIV)
      (E (? 5 *RS KOPIERT5 *P0))
      (E (? 6 *RS KOPIERT4 *P4))
      (E (? 1 *RS KOPIERT3 *P))
      (E (? 2 *RS KOPIERT2 *P1))
      (E (? 3 *RS KOPIERT1 *P2))
      (E (? 4 *RS KOPIERT-LISP))
      (CALL (? 4 (LISP-DUPLIKATE *RS)))
      (CALL (HILFSFKT-VERMERK! S6-1))
      (CALL (NO-KWOTE S6-1))
      (CALL (IN-BINDE! *RS))
      =%>
      (*RS FINAL (? 5 %( DUPLIZIERE *P0 X5 %))
           (? 6 %( DUPLIZIERE *P4 X4 %))
           (? 1 %( DUPLIZIERE *P X3 %))
           (? 2 %( DUPLIZIERE *P1 X2 %))
           (? 3 %( DUPLIZIERE *P2 X1 %))
           (? 4 *LISPDUPL)
           *F1 *F2 *F3 *F4 *F5 *F6 *F7 *F8 *F9
              %( *ALGNAME *PARLIST %)))
```

Bemerk:. 1) Die in einer CALL-Voraussetzung aufgerufene Funktion LISP-DUPLIKATE verarbeitet die Information, die unter dem Indikator KOPIERT-LISP abgelegt ist, zu einer Folge von LISP-Formen, die unter dem Variablenbezeichner *LISPDUPL in der Bindeliste BINDE abgelegt wird. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von LISP-DUPLIKATE)

2) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion HILFSFKT-VERMERK!: Siehe Bem. 3) von Regel S2-9.

3) Die in einer CALL-Voraussetzung aufgerufene Funktion NO-KWOTE prüft, ob in der Bindeliste BINDE einer der Variablenbezeichner einen Wert %'atom aufweist und wandelt ihn ggf. um in %(QUOTE ATOM %), da die erste Darstellung nicht der intendierte, LISP-ausführbare Code ist. (Siehe auch Funktionsbeschreibung von NO-KWOTE)

4) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion IN-BINDE!: Siehe Bem. 1) von Regel S3-1.
5) Die z.Zt. implementierte Form der Regel verarbeitet maximal 9 Argumente rekursiver Aufrufe.
6) Die minimale Musterlänge wird als 0 definiert.

■

S6-21

Ziel: Die Regel soll bereits im 4. Schritt vollständig bearbeitete rechte Seiten geeignet transformieren bzw. kopieren.

Ergebnis: Der Indikator FINAL wird auf der Property-Liste von Ri initialisiert. Sein Wert ist entweder 'NIL oder der in Schritt 4 erkannte und unter dem Indikator FORM abgelegte Code.

```
(S6-21 (E *RS FORM *F1)
       (CALL (? 1 (GLEICH *F1 NOFORM)))
       (V 2 NICHT 1)
       =%>
       ((? 2 *RS FINAL *F1))
       ((? 1 *RS FINAL %'NIL)))
```

Bemerk.:
1) Wirkung der in der CALL-Voraussetzung aufgerufenen Funktion GLEICH: Siehe Bem. 2) von Regel S5-D1.
2) Die minimale Musterlänge ist als 0 definiert.

■

S6-22

Ziel: Die Regel soll den Code für eine parallel-rekursive rechte Seite Ri aufbauen.

Ergebnis: Der Indikator FINAL der Property-Liste von Ri wird initialisiert. Sein Wert ist der LISP-Code der transformierten rechten Seite.

```
(S6-22 (E *RS EVALUIERT PARALLEL-REKURSIV)
       (CALL (PARALLEL-REK-AUFBAU *RS))
       =%>
       (*RS FINAL *PARALLEL-REK-AUFBAU))
```

Bemerk:.
1) Die in der CALL-Voraussetzung aufgerufene Funktion PARALLEL-REK-AUFBAU erzeugt mit Hilfe des Wertes des Indikators NEUE-NAMEN und den damit assoziierten rekursiven Aufrufen ein Code-Stück, das die parallel-rekursive rechte Seite durch LISP-FUNARG-Konstrukte innerhalb eines LISP-PROGs realisiert. Dieses Code-Stück wird unter dem Variablenbezeichner *PARALLEL-REK-AUFBAU in der Bindeliste BINDE abgelegt . Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch

Funktionsbeschreibung von PARALLEL-REK-AUFBAU)
2) Die minimale Musterlänge wird definiert als 0.

■

S6-23

Ziel: Die Regel soll für die Konstruktklassen "Alg-Aufruf", "Ink-Aufruf" und "Lisp-Aufruf" den geeigneten Code erzeugen.

Ergebnis: Auf der Property-Liste von Ri wird der Indikator FINAL initialisiert, dessen Wert der entsprechende LISP-Code von Ri ist.

```
(S6-23 (E (? 1 *RS EVALUIERT ALG-AUFRUF))
       (E (? 2 *RS EVALUIERT INK-AUFRUF))
       (E (? 3 *RS EVALUIERT LISP-AUFRUF))
       (CALL (NOT-ALL-NIL? 1 2 3))
       (CALL (FORM-BIND! *RS))
       =%>
       (*RS FINAL *FB>
```

Bemerk.:
1) Wirkung der in einer CALL-Voraussetzung aufgerufenen Funktion NOT-ALL-NIL?: Siehe Bem. 3) von Regel S4-51.
2) Die in einer CALL-Voraussetzung aufgerufene Funktion FORM-BIND! kopiert den Wert des Indikators FORM in die Bindeliste BINDE unter dem Variablenbezeichner *FB. Diese Variable wird in der Folgerung der Regel verwendet. (Siehe auch Funktionsbeschreibung von FORM-BIND!)
3) Die minimale Musterlänge ist als 0 definiert.

5.3. Anwendungsfunktionen

Um mit den durch APE implementierten Datentypen bzw. Algorithmen bequem arbeiten zu können, sind einige Anwendungsfunktionen definiert, die z.B. dazu dienen, konkrete Inkarnationen von einem implementierten Datentyp (formale Struktur) zu erzeugen (s.a. 3.1.) oder die bei einem Algorithmus benutzten Relationen zu realisieren.
Die Anwendungsfunktionen für Datentypen werden in Abschnitt 5.3.1 beschrieben und sind in der Datei LISP.DATA.ADT-EX.nn enthalten; die Anwendungsfunktionen für Algorithmen werden in Abschnitt 5.3.2 beschrieben und sind in der Datei LISP.DATA.ALG-EX.nn enthalten.
Im allgemeinen wird man beim Arbeiten mit von ADTCOMP implementierten Datentypen mit den in der Datei LISP.DATA.ADT-EX.nn enthaltenen Anwendungsfunktionen auskommen. Will man jedoch die durch ALGCOMP implementierten Algorithmen benutzen, so sind in der Regel beide Dateien, nämlich sowohl LISP.DATA.ALG-EX.nn wie auch LISP.DATA.ADT-EX.nn zu laden, da die Algorithmen ja nur in Ausnahmefällen nicht auf den selbstimplementierten Datentypen definiert sind.

5.3.1. Anwendungsfunktionen für Datentypimplementierungen

<INIT <NLAMBDA (INKNAME FORMSTRUKNAME) ... >>

Zweck: Erzeugen einer konkreten Inkarnation mit Namen INKNAME von einem Datentyp mit der formalen Struktur FORMSTRUKNAME.
Aufruf: Benutzer
Param.: INKNAME ist der Name der Datentyp-Inkarnation, die von INIT erzeugt wird;
FORMSTRUKNAME ist der Name der Formalen Struktur, von der eine Inkarnation erzeugt werden soll.
Ergebnis: Nach Ausführung der Funktion INIT steht dem Benutzer eine konkrete Inkarnation mit Namen INKNAME der formalen Struktur FORMSTRUKNAME im System zur Verfügung. Diese Inkarnation wird als Literales Atom mit den in 2.1 bzw. 2.2 beschriebenen Bindungen realisiert.
Wert: Wert von INIT ist eine Meldung, daß die Inkarnation im System zur weiteren Verarbeitung zur Verfügung steht.

```
<INIT
 <NLAMBDA (INKNAME FORMSTRUKNAME)
 (REMPROPS (QUOTE (INKNAME)))
 (PUT INKNAME (QUOTE PASSIVSTRUKTUR)
      FORMSTRUKNAME)
 (PUT INKNAME (QUOTE PARAMETER)
      (GETP FORMSTRUKNAME (QUOTE PARAMETER)))
 (PUT INKNAME (QUOTE STRUKTUR)
```



```
      (GETP FORMSTRUKNAME (QUOTE STRUKTUR)))
<PUTD INKNAME (SUBST INKNAME (QUOTE INKNAME)
                (SUBST FORMSTRUKNAME
                       (QUOTE FORMSTRUKNAME)
                       (QUOTE (NLAMBDA (OP . ARGLIST)
                        <SETQ FORM
                              (APPEND
                               (LIST (QUOTE
                                      FORMSTRUKNAME)
                                     OP
                                     (QUOTE INKNAME))
                               (MAPCAR ARGLIST
                                 (FUNCTION (NLAMBDA (ARG)
                                   (COND
                                     ((LISTP ARG)
                                       (EVAL ARG))
                                     ((GETP ARG
                                         (QUOTE STRUKTUR))
                                        ARG)
                                     (T (EVAL ARG>
                    (APPLY (CAR FORM)
                           (CDR FORM>
 (SET INKNAME (APPLY* (GETD (EVAL (QUOTE FORMSTRUKNAME)))
                      (QUOTE DO)))
 (CONCAT INKNAME
         "ALS" FORMSTRUKNAME "-INKARNATION VERFUEGBAR">>
```

<INSERT <NLAMBDA (NEUPARAM DUMMY FORMSTRUKNAME) ... >>

Zweck: Erzeugen einer neuen formalen Struktur, d.h. eines neuen implementierten Datentyps, mit demselben Haupttyp wie FORMSTRUKNAME, jedoch mit neuen Parametern.
Beispiel: Sei NEUPARAM=Integer, DUMMY=in und FORMSTRUKNAME=Keller-Schlange; ferner sei die formale Struktur "Keller-Schlange", eines mit Schlangen parametrisierten Kellers, dem System bekannt. Dann existiert nach dem Aufruf (INSERT INTEGER IN KELLER-SCHLANGE) eine formale Struktur mit Namen "Keller-Integer" als Realisierung des abstrakten Integer-Kellers im System.
Aufruf: Benutzer
Param.: NEUPARAM bezeichnet den (die) in die formale Struktur FORMSTRUKNAME einzusetzenden Parameter;
DUMMY ist eine unbenutzer Parameter, der mit 'IN' angegeben werden sollte;
FORMSTRUKNAME bezeichnet diejenige formale Struktur, die neu parametrisiert werden soll.
Ergebnis: ist eine neue formale Struktur mit gleichem Haupttyp wie FORMSTRUKNAME und Parameter(n) NEUPARAM.
Wert: NIL
Bem. Die formale Struktur FORMSTRUKNAME bleibt unverändert erhalten.

Ist FORMSTRUKNAME unparametrisiert, erfolgt eine Fehlermeldung, da unparametrisierte Datentypen wegen Strukturverschiedenheiten auf diesem Wege nicht parametrisiert werden können.
Ist die zu erzeugende formale Struktur bereits im System vorhanden, erfolgt ebenfalls eine Meldung und INSERT wird beendet.

```
<INSERT
 (COND
  <<EQ (GETD NEUSTRUKNAME)
       NIL)
    (COND
      ((NEQ (GETP FORMSTRUKNAME (QUOTE PARAMETER))
            NIL)
       (RPLACA NEUSTRUKNAME (SUBST NEUSTRUKNAME
                                   FORMSTRUKNAME
                                   (CAR FORMSTRUKNAME)))
       (RPLACD NEUSTRUKNAME)
       (PUT NEUSTRUKNAME (QUOTE STRUKTUR)
            (GETP FORMSTRUKNAME (QUOTE STRUKTUR)))
       (PUT NEUSTRUKNAME (QUOTE PARAMETER)
            NEUPARAM)
       (PUTD NEUSTRUKNAME (SUBST NEUSTRUKNAME FORMSTRUKNAME
                                 (GETD FORMSTRUKNAME)))
       (PRINTL NEUSTRUKNAME
               "ALS FORMALE STRUKTUR VERFUEGBAR!"))
      (T (PRINTL FORMSTRUKNAME "IST NICHT PARAMETRISIERT">
  (T (PRINTL NEUSTRUKNAME "IST SCHON IM SYSTEM VORHANDEN">>
```

■

<TEST <NLAMBDA (NR . ARGLIST) ... >>

Zweck: Die Schreiboperatoren parametrisierter Datentypen überprüfen vor dem Schreiben, ob ihr(e) Argument(e) den vom Parameter des Datentyps geforderten Typ besitzen (vgl. 3.1.). Dieser Überprüfung dient die Funktion TEST.

Aufruf: durch die Schreiboperatoren parametrisierter Datentypen.

Param.: NR gibt die Art des Tests an (für eventuelle Erweiterungen);
ARGLIST gibt den Datentyp an, für den der Test durchgeführt werden soll, also den Typ, zu dem der Schreiboperator gehört. (Für eventuelle Erweiterungen als Liste programmiert; half-spread-form von TEST.)

Wert: TEST ist eine Prädikat-Funktion, die genau dann T liefert, wenn der Argumenttyp des Schreiboperators mit dem Parameter des Datentyps übereinstimmt, NIL sonst.

Bem.: Liefert TEST als Wert NIL, so wird über die allgemeine Fehlerfunktion FEHLER (siehe 5.4.) eine Fehlermeldung ausgegeben und die Schreiboperation

abgebrochen; sonst wird die Schreiboperation ausgeführt.

```
<TEST
 <NLAMBDA (NR . ARGLIST)
 (COND
   ((EQUAL NR 1)
     (COND
       (<OR (EQUAL (GETP (CAR (LAST ARGUMENTE))
                         (QUOTE PASSIVSTRUKTUR))
                   (GETP (CAR ARGLIST)
                         (QUOTE PARAMETER)))
            (AND (EQUAL (GETP (CAR ARGLIST)
                              (QUOTE PARAMETER))
                        (QUOTE INTEGER))
                 (OR (TYPEP (CAR (LAST ARGUMENTE))
                            22)
                     (TYPEP (CAR (LAST ARGUMENTE))
                            23)))
            (AND (EQUAL (GETP (CAR ARGLIST)
                              (QUOTE PARAMETER))
                        (QUOTE LISPATOM))
                 (OR (TYPEP (CAR (LAST ARGUMENTE))
                            22)
                     (TYPEP (CAR (LAST ARGUMENTE))
                            23)
                     (TYPEP (CAR (LAST ARGUMENTE))
                            30>
        T)
       (T NIL>>
```

■

<ANF <LAMBDA (A) ... >>

Zweck: Hier: Hilfsfunktion für die Operatoren des Datentyps BINAERERBAUM. (s.a. [EKT 79])

Param.: A muß ein Atom sein.

Wert: Liefert das erste Zeichen eines Atoms.

```
<ANF
 <LAMBDA (A)
 (CAR (UNPACK A>>
```

■

<REST <LAMBDA (A) ... >>

Zweck: Hier: Hilfsfunktion für die Operatoren des Datentyps BINAERERBAUM. (s.a. [EKT 79])

Param.: A muß ein Atom sein.

Wert: Liefert ein Atom, das aus A durch abschneiden des ersten Zeichens entsteht.

```
<REST
 <LAMBDA (A)
 (PACK (CDR (UNPACK A>>
```

■

<GED-ZWISCHEN <NLAMBDA (OP AL) ... >>

Zweck: Hier: Hilfsfunktion für Gedächtnis-Strukturen (GRENZKELLER, SCHLUESSELLISTE). (s.a. [EKT 79])
Beschreibung siehe 5.1.1.2.4, Seite 330.

<LESSEQP <LAMBDA (A B) ... >>

Zweck: Hier: Hilfsfunktion für den Löschoperator den Datentyp FLEXREIHE. (s.a. [EKT 79])
Beschreibung siehe 5.3.2, Seite 510.

5.3.2. Anwendungsfunktionen für Algorithmus-Implementierungen

<DUPLIZIERE <NLAMBDA (ORIGINAL DOPPEL) ... >>

Zweck: Einfaches oder mehrfaches Kopieren von Datentypen. DUPLIZIERE unterscheidet, ob ORIGINAL einen durch ADTCOMP erzeugten Datentyp oder einen INTERLISP-Typ bezeichnet. Im ersten Fall werden alle drei Verweise des Atoms <ORIGINAL>, also CAR- und CDR-Verweis sowie die Funktions-Definitions-Zelle, kopiert; im zweiten Fall wird nur der Wert, also der CAR-Verweis kopiert.
Bei vielen rekursiven bzw. parallel-rekursiven Algorithmen treten in der Parameterliste eines rekursiven Aufrufs dieselben Datenstrukturen als Parameter mehrfach auf. Um bei der Abarbeitung eines solchen Aufrufs, d.h. inbesondere bei der Berechnung der aktuellen Parameter, von der Reihenfolge der Parameter unabhängig zu sein, werden mehrfach auftretende Parameter zur Ausführungszeit des Algorithmus dupliziert.
Ob ein Parameter an einer bestimmten Stelle kopiert werden muß oder nicht, wird durch die Regeln des vierten und fünften Schrittes bestimmt.

Aufruf: durch von ALGCOMP implementierte Algorithmen

Param.: ORIGINAL verweist auf die zu kopierende Datenstruktur;
DOPPEL kann als Atom oder als Liste von Atomen angegeben werden.
Im ersten Fall wird eine Kopie von ORIGINAL erstellt und an DOPPEL gebunden; ist DOPPEL eine Liste von Atomen, so wird für jedes Atom dieser Liste eine

Kopie von ORIGINAL erstellt und an <Atom> gebunden.

Ergebnis: Ergebnis von DUPLIZIERE ist (sind) die von ORIGINAL erzeugte(n) Kopie(n), zugreifbar über das (die) durch DOPPEL angegebene(n) Atom(e).

Wert: NIL

Bem.: Da bei "leeren Datentypen", die von ADTCOMP erzeugt wurden, der CAR-Verweis der ersten CONS-Zelle auf sich selber zeigt und so eine unendliche Klammerungstiefe entsteht, können solche Strukturen nicht mit DUPLIZIERE kopiert werden. Tritt ein solcher Fall auf, wird eine Fehlermeldung gedruckt und DUPLIZIERE geht in den Break-Mode.

```
<DUPLIZIERE
  <NLAMBDA (ORIGINAL DOPPEL)
    <COND
      ((LISTP DOPPEL)
        NIL)
      (T (SETQ DOPPEL (LIST DOPPEL>
    <MAPC DOPPEL (FUNCTION (LAMBDA (D)
              (COND
                <(GETP ORIGINAL (QUOTE STRUKTUR))
                  (COND
                    ((CDR (EVAL ORIGINAL))
                      (RPLACA D (COPY (EVAL ORIGINAL)))
                      (RPLACD D (COPY (CDR ORIGINAL)))
                      (PUTD D (SUBST D ORIGINAL
                                     (GETD ORIGINAL)))
                      D)
                    (T (PRINTL "***ERROR*** LEERER
                        DATENTYP" ORIGINAL "KANN
                        NICHT KOPIERT WERDEN")
                       (BREAK1 NIL T>
                (T (RPLACA D (COPY (EVAL ORIGINAL>
    NIL>>
```

■

<FIBO-ZAHL <LAMBDA (Z B ...) ... >>

Zweck: Hilfsfunktion für den Algorithmus Fibonaccian-Search.

Aufruf: durch den Algorithmus Fibonaccian-Search.

Param.: Z: Gibt an, ob die 1.,2. oder 3. FIBINACCI-Zahl berechnet werden soll, die kleinergleich B ist.
B: Gibt Grenze für A an.

Ergebnis: Keine Seiteneffekte.

Wert: Die entsprechende FIBONACCI-Zahl.

```
<FIBO-ZAHL
  <LAMBDA (Z B FN FN-1 FN-2)
    (SETQ FN-2 0)
    (SETQ FN-1 1)
    (SETQ FN 1)
    (COND
      ((OR (LESSP B Z)
           (LESSP B 3))
        (PRINTL "****FEHLER**** DAS INTERVALL MUSS
                 GROESSER ALS (0 , 3) SEIN!")
        (RESET))
      (T NIL))
    (PROG (X)
      LOP (SETQ X (PLUS FN FN-1))
          (SETQ FN-2 FN-1)
          (SETQ FN-1 FN)
          (SETQ FN X)
          (COND
            <(GREATEREQP (PLUS X FN-1)
                         B)
              (COND
                ((EQ Z 1)
                  (RETURN FN))
                ((EQ Z 2)
                  (RETURN FN-1))
                ((EQ Z 3)
                  (RETURN FN-2))
                ((EQ Z 4)
                  (RETURN (IDIFFERENCE
                            (IDIFFERNECE (PLUS X FN-1)
                                         B)
                            1)))
                (T (RETURN (QUOTE ***FEHLER***>
            (T (GO LOP>>
```

■

<ATT <LAMBDA (A X) ... >>

Zweck: Ist eigentlich Anwendungsfunktion für FLEXVERBUND. Dient im PVS-Beispiel 4.6. dazu, zulässige Indikatoren zu definieren.
Aufruf: Durch FLEXVERBUND-Inkarnationen.
Param.: A: Attribut für FLEXVERBUND.
Ergebnis: Keine Seiteneffekte.
Wert: Wahrheitswert T, falls A zulässig, sonst NIL.


```
<ATT
  <LAMBDA (A X)
    (COND
      ((MEMB A (QUOTE (NAME VORNAME GEWICHT GROESSE
                       FAMILIENSTAND TEL-NR LOHN-GR
                       GEB-TAG WOHNORT STRASSE PNR)))
       T)
      (T NIL>>
```

■

<LISTSIZE <LAMBDA (LIST) ... >>

Zweck: Hilfsfunktion für die Algorithmen Quick-Sort und Binary-Search.
Aufruf: Durch die implementierten Algorithmen Quick-Sort und Binary-Search.
Param.: LIST ist der Wert eines von ADTCOMP implementierten Datentyps, also eine Liste der Form:
((x) x x ... x),
wobei x für verschiedene Elemente der Liste steht.
Ergebnis: ist die tatsächliche Anzahl der Elemente der Liste bzw. des Datentyps, also ohne die Unterliste (erstes Element).
Wegen dieser Unterliste kann hier auch nicht direkt mit der INTERLISP-Funktion LENGTH gearbeitet werden.
Wert: siehe Ergebnis.

```
<LISTSIZE
  <LAMBDA (LIST)
  (* "HILFSFUNKTION FUER QUICK-SORT UND BINARY-SEARCH")
    (COND
      <(NOT (LISTP LIST))
        (SUB1 (LENGTH (EVAL LIST>
      (T (SUB1 (LENGHT LIST>>
```

■

<TEST <NLAMBDA (NR . ARGLIST) ... >>

Zweck: Überprüfen des Argumenttyps bei Schreiboperatoren parametrisierter Datentypen.
Beschreibung siehe 5.3.1., Seite 503.

Prädikat-Funktionen für Relationssätze

Es folgen einige Prädikat-Funktionen, die die Vergleiche, die auf den linken Seiten der Produktionregeln in den Spezifikationen auftreten, realisieren.
Die Syntax der Produktionsregelmenge erlaubt nur die sechs Relationszeichen = , /= , < , > , <= und >= , um Vergleiche auszudrücken. Offen bleibt jedoch, auf welchen Datentypen diese Relationen arbeiten. Betrachtet man z.B. die <-Relation auf Integern, so ist die Bedeutung des Ausdrucks a<b, a,b ganze Zahlen, unmittelbar klar. Die <-Relation läßt sich in diesem Fall durch die INTERLISP-Funktion LESSP realisieren. Sind a und b jedoch z.B. literale Atome, so ist nicht in jedem Fall sofort einzusehen, welche Bedeutung dem Ausdruck a<b zukommen soll (man denke z.B. an die Behandlung von Sonderzeichen o.ä.).
Demselben Relationszeichen kommt also in Abhängigkeit von dem Datentyp, auf den es angewendet wird, unterschiedliche Bedeutung zu, d.h. es muß durch verschiedene Funktionen realisiert werden.
Aufgrund dessen sind die folgenden Prädikat-Funktionen in "Relationssätzen" zu je sechs Funktionen für jeweils einen Datentyp zusammengefaßt. Diese "Relationssätze", d.h. die Zuordnung von obigen Relationszeichen zu Namen LISP-ausführbarer Funktionen, benutzt ALGCOMP in der Algorithmen-Konstruktions-Phase, um in den LISP-Code die Namen der entsprechenden Prädikat-Funktionen einzusetzen.
Die Relationssätze können bequem mit Hilfe der Funktion MAKE-REL erzeugt werden (Beschreibung siehe 5.4.,Seite 518).
Im Header der Algorithmen-Spezifikation kann unter dem Schlüsselwort 'Relation' der Name des Relationssatzes angegeben werden, mit dem ALGCOMP arbeiten soll. Wird kein Relationssatz spezifiziert, so wird der Relationssatz STANDARD verwendet.
Zur Zeit sind sechs Relationssätze und die dazugehörigen Prädikat-Funktionen definiert. Sie unterscheiden sich in erster Linie durch die Datentypen, auf denen diese Funktionen arbeiten.
Im folgenden werden die einzelnen Relationssätze und die sie realisierenden Funktionen besprochen:

Relationssatz: STANDARD

```
(( = . EQ)                    (/= . NEQ)
 ( < . LESSP)                 ( > . GREATERP)
 (<= . LESSEQP)               (>= . GREATEREQP))
```

Dieser Relationssatz ist für Normalvergleiche zwischen Integern definiert worden. Es können jedoch auch andere Zahlen und andere Datentypen benutzt werden (Näheres über Definitionsbereich und Arbeitsweise dieser INTERLISP-Funktionen siehe z.B. [EK 77,pp. 7-17] oder [GK 78]).
Die Funktionen EQ, NEQ, LESSP und GREATERP werden durch das INTERLISP-System zur Verfugung gestellt. (Funktionsweise siehe [GK 78,pp. 5-9 u. 13-7])

LESSEQP und GREATEREQP sind wie folgt realisiert:

```
<LESSEQP
  <LAMBDA (A B)
    (OR (LESSP A B)
        (EQ A B>>                                    ■


<GREATEREQP
  <LAMBDA (A B)
    (OR (GREATER A B)
        (EQ A B>>                                    ■
```

Relationssatz: UNDEF-KL-INTEGER

```
(( = . GLEICHP)               (/= . UNGLEICHP)
 ( < . KLEINERP)              ( > . GRROESSERP)
 (<= . KLEINERGLEICHP)        (>= . GROESSERGLEICHP)
```

Dieser Relationssatz ist auch in erster Linie für den Datentyp Integer definiert worden; jedoch wird UNDEF hier als ein Element angesehen, daß kleiner als jede ganze Zahl ist. Damit lassen sich Algorithmen oft wesentlich einfacher formulieren (siehe z.B. 4.4.1,MERGE-SORT).

```
<GLEICHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (EQ A B>>                                        ■


<UNGLEICHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (NEQ A B>>                                       ■


<KLEINERP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (COND
      ((EQ B (QUOTE UNDEF))
        NIL)
      ((EQ A (QUOTE UNDEF))
        T)
      (T (LESSP A B>>                                ■


<GROESSERP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (COND
      ((EQ A (QUOTE UNDEF))
        NIL)
```

```
    ((EQ B (QUOTE UNDEF))
      T)
    (T (GREATERP A B>>                                    ■


<KLEINERGLEICHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (OR (KLEINERP A B)
        (GLEICHP A B>>                                    ■


<GROESSERGLEICHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-INTEGER")
    (OR (GROESSERP A B)
        (GLEICHP A B>>                                    ■
```

Relationssatz: ALPHORDER

```
    (( = . EQ)                    (/= . NEQ)
     ( < . LESS-ALPHP)            ( > . GREATER-ALPHP)
     (<= . LESSEQ-ALPHP)          (>= . GREATEREQ-ALPHP))
```

Die Funktionen dieses Relationssatzes vergleichen ihre Argumente mit Hilfe der INTERLISP-Funktion ALPHORDER, die eine lexikographische Ordnung auf Literalen Atomen und Zahlen herstellt.

```
<LESS-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: ALPHORDER")
    (ALPHORDER B A>>                                      ■


<GREATER-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: ALPHORDER")
    (ALPHORDER A B>>                                      ■


<LESSEQ-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: ALPHORDER")
    (OR (EQ A B)
        (ALPHORDER B A>>                                  ■


<GREATEREQ-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: ALPHORDER")
    (OR (EQ A B)
        (ALPHORDER A B>>                                  ■
```


Relationssatz: UNDEF-KL-ALPHORDER

```
    (( = . GLEICH-ALPHP)          (/= . UNGLEICH-ALPHP)
     ( < . KLEINER-ALPHP)         ( > . GROESSER-ALPHP)
     (<= . KLEINERGLEICH-ALPHP)   (>= . GROESSERGLEICH-ALPHP))
```

Auch dieser Relationssatz berücksichtigt, wie UNDEF-KL-INTEGER ein ausgezeichnetes Element UNDEF, das kleiner als alle anderen ist. Ansonsten gilt das für UNDEF-KL-INTEGER und ALPHORDER gesagte.

```
<GLEICH-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (EQ A B>>                                             ■


<UNGLEICH-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (NEQ A B>>                                            ■


<KLEINER-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (COND
      ((EQ B (QUOTE UNDEF))
        NIL)
      ((EQ A (QUOTE UNDEF))
        T)
      (T (AND (NEQ A B)
              (ALPHORDER B A>>                            ■


<GROESSER-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (COND
      ((EQ A (QUOTE UNDEF))
        NIL)
      ((EQ B (QUOTE UNDEF))
        T)
      (T (AND (NEQ A B)
              (ALPHORDER A B>>                            ■


<KLEINERGLEICH-ALPHP
  <LAMBDA (A B)
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (OR (KLEINER-ALPHP)
        (GLEICH-ALPHP>>                                   ■


<GROESSERGLEICH-ALPHP
  <LAMBDA (A B)
```

```
  (* "REL FUER REL-SATZ: UNDEF-KL-ALPHORDER")
    (OR (GROESSER-ALPHP)
        (GLEICH-ALPHP>>                                    ■
```

Relationssatz: PVSREL-PNR

```
    (( = . GLEICH-PNRP)          (/= . UNGLEICH-PNRP)
     ( < . KLEINER-PNRP)         ( > . GROESSER-PNRP)
     (<= . KLEINERGLEICH-PNRP)   (>= . GROESSERGLEICH-PNRP))
```

Dieser Relationssatz wird im PVS-System in 4.6. verwendet. Seine Relationen berücksichtigen die Fälle, wo ein Argument UNDEF ist, und sie stellen den notwendigen Zugriff auf die PNR-Werte der beteiligten FLEXVERBUNDE her. Diese Relationen erleichtern die Spezifikation von Abfragen und Tests.

```
<GLEICH-PNRP
  <LAMBDA (A B)
    (COND
      ((EQ A (QUOTE UNDEF))
        (EQ B (QUOTE UNDEF)))
      ((EQ B (QUOTE UNDEF))
        (EQ A (QUOTE UNDEF)))
      ((EQ A T)
        (EQ B T))
      ((EQ A)
        (EQ B))
      (T (EVAL (DSUBST A (QUOTE A)
                 (SUBST B (QUOTE B)
                   (QUOTE (GLEICHP (A LIES (QUOTE PNR))
                                   (B LIES (QUOTE PNR>>    ■


<UNGLEICH-PNRP
  <LAMBDA (A B)
    (COND
      <(EQ A (QUOTE UNDEF))
        (NOT (EQ B (QUOTE UNDEF>
      <(EQ B (QUOTE UNDEF))
        (NOT (EQ A (QUOTE UNDEF>
      ((EQ A T)
        (NOT (EQ B T)))
      ((EQ A)
        (NOT (EQ B)))
      (EVAL (DSUBST A (QUOTE A)
              (SUBST B (QUOTE B)
                (QUOTE (UNGLEICHP (A LIES (QUOTE PNR))
                                  (B LIES (QUOTE PNR>>    ■


<KLEINER-PNRP
  <LAMBDA (A B)
    (COND
```



```
      ((EQ A (QUOTE UNDEF))
        T)
      ((EQ B (QUOTE UNDEF))
        NIL)
      (T (EVAL (DSUBST A (QUOTE A)
                 (SUBST B (QUOTE B)
                   (QUOTE (KLEINERP (A LIES (QUOTE PNR))
                                    (B LIES (QUOTE PNR>>    ■


<GROESSER-PNRP
  <LAMBDA (A B)
    (COND
      ((EQ A (QUOTE UNDEF))
        NIL)
      ((EQ B (QUOTE UNDEF))
        T)
      (T (EVAL (DSUBST A (QUOTE A)
                 (SUBST B (QUOTE B)
                   (QUOTE (GROESSERP (A LIES (QUOTE PNR))
                                     (B LIES (QUOTE PNR>>    ■


<KLEINERGLEICH-PNRP
  <LAMBDA (A B)
    (OR (KLEINER-PNRP A B)
        (GLEICH-PNRP A B>>                                 ■


<GROESSERGLEICH-PNRP
  <LAMBDA (A B)
    (OR (GROESSER-PNRP A B)
        (GLEICH-PNRP A B>>                                 ■
```

Relationssatz: PVSREL-NAM

```
    (( = . GLEICH-NAMP)          (/= . UNGLEICH-NAMP)
     ( < . KLEINER-NAMP)         ( > . GROESSER-NAMP)
     (<= . KLEINERGLEICH-NAMP)   (>= . GROESSERGLEICH-NAMP))
```

Dieser Relationssatz wird im PVS-System in 4.6. verwendet. Seine Relationen berücksichtigen die Fälle, wo ein Argument UNDEF ist, und sie stellen den notwendigen Zugriff auf die NAME-Werte der beteiligten FLEXVERBUNDE her. Diese Relationen erleichtern die Spezifikation von Abfragen und Tests.

```
<GLEICH-NAMP
  <LAMBDA (A B)
    (COND
      ((EQ A (QUOTE UNDEF))
        (EQ B (QUOTE UNDEF)))
      ((EQ B (QUOTE UNDEF))
        (EQ A (QUOTE UNDEF)))
      ((EQ A T)
        (EQ B T))
      ((EQ A)
        (EQ B))
      (T (EVAL (DSUBST A (QUOTE A)
                       (SUBST B (QUOTE B)
                         (QUOTE (GLEICH-ALPHP
                                  (A LIES (QUOTE NAME))
                                  (B LIES (QUOTE NAME>>
```

■

```
<UNGLEICH-NAMP
  <LAMBDA (A B)
    (COND
      <(EQ A (QUOTE UNDEF))
        (NOT (EQ B (QUOTE UNDEF>
      <(EQ B (QUOTE UNDEF))
        (NOT (EQ A (QUOTE UNDEF>
      (EVAL (DSUBST A (QUOTE A)
                    (SUBST B (QUOTE B)
                      (QUOTE (UNGLEICH-ALPHP
                               (A LIES (QUOTE NAME))
                               (B LIES (QUOTE NAME>>
```

■

```
<KLEINER-NAMP
  <LAMBDA (A B)
    (COND
      ((EQ A (QUOTE UNDEF))
        (NEQ B (QUOTE UNDEF)))
      ((EQ B (QUOTE UNDEF))
        NIL)
      (T (EVAL (DSUBST A (QUOTE A)
                       (SUBST B (QUOTE B)
                         (QUOTE (KLEINER-ALPHP
                                  (A LIES (QUOTE NAME))
                                  (B LIES (QUOTE NAME>>
```

■

```
<GROESSER-NAMP
  <LAMBDA (A B)
    (COND
      ((EQ A (QUOTE UNDEF))
        NIL)
      ((EQ B (QUOTE UNDEF))
        T)
      (T (EVAL (DSUBST A (QUOTE A)
                       (SUBST B (QUOTE B)
                         (QUOTE (GROESSER-ALPHP
```



```
                                  (A LIES (QUOTE NAME))
                                  (B LIES (QUOTE NAME>>
```

■

```
<KLEINERGLEICH-NAMP
  <LAMBDA (A B)
    (OR (KLEINER-NAMP A B)
        (GLEICH-NAMP A B>>
```

■

```
<GROESSERGLEICH-NAMP
  <LAMBDA (A B)
    (OR (GROESSER-NAMP A B)
        (GLEICH-NAMP A B>>
```

■

5.4. Allgemein verwendbare Funktionen

In diesem Abschnitt werden einige Funktionen besprochen, die von verschiedenen Teilen des APE-Systems benutzt werden, aber meist auch direkt vom Benutzer zu eigenen Zwecken aufgerufen werden können.

<CONVERT-KWOTE <LAMBDA (LISTE) ... >>

Zweck: Konvertierung gekwoteter Atome 'Atom, die auch in beliebigen Listen auftreten können, in die Form (QUOTE Atom).
Um in den Algorithmen-Spezifikationen auch gekwotete Atome, wie sie z.B. in Datentyp-Operations-Aufrufen auftreten, kurz und bequem ausdrücken zu können und auch als Atome zu erkennen, wurde diese Funktion geschrieben. Würde man gekwotete Atome in der normalen LISP-Form (QUOTE Atom) spezifizieren, hätte man die Syntax der Spezifikationssprache erweitern müssen. So wird auch ein gekwotetes Atom der Form 'Atom als Atom erkannt und durch CONVERT-KWOTE später intern, in der AK-Phase und in der LISP-Phase, in die LISP-ausführbare Form (QUOTE Atom) umgewandelt.
Aufruf: durch SCHRITT-6, LISP-PHASE und Benutzer
Param: LISTE ist eine beliebige Liste, in der alle Atome der Form 'Atom in die Form (QUOTE Atom) umgewandelt werden. Ist LISTE atomar, so wird dieses Atom konvertiert.
Wert: ist das umgewandelte Atom bzw. die konvertierte Liste.

```
<CONVERT-KWOTE
  <LAMBDA (LISTE)
    (* "MACHT AUS 'ATOM -> (QUOTE ATOM),
       AUCH IN BELIEBIGEN LISTEN")
    (COND
      <<(LISTP LISTE)
        (MAPCAR LISTE (FUNCTION (LAMBDA (ELEM)
                  (COND
                    ((LISTP ELEM)
                      (CONVERT-KWOTE ELEM))
                    (T (COND
                         <<(AND (EQ (CAR (UNPACK ELEM))
                                    (QUOTE %'))
                                (NEQ (PACK
                                       (CDR (UNPACK ELEM)))
                                     NIL))
                           (LIST (QUOTE QUOTE)
                                 (PACK (CDR (UNPACK ELEM>
                         ((AND (EQ (CAR (UNPACK ELEM))
                                   (QUOTE %'))
                               (EQ (PACK
                                     (CDR (UNPACK ELEM)))
                                   NIL))
                          NIL)
```



```
                                   (T ELEM>
      (T (COND
           <<(AND (EQ (CAR (UNPACK LISTE))
                      (QUOTE %'))
                  (NEQ (PACK (CDR (UNPACK LISTE)))
                       NIL))
             (LIST (QUOTE QUOTE)
                   (PACK (CDR (UNPACK LISTE>
           ((AND (EQ (CAR (UNPACK LISTE))
                     (QUOTE %'))
                 (EQ (PACK (CDR (UNPACK LISTE)))
                     NIL))
            NIL)
           (T LISTE>>
```

■

<MAKE-REL <NLAMBDA (NAME) ... >>

Zweck: MAKE-REL unterstützt das Erzeugen von Assoziationslisten mit Paaren der Form (Relationszeichen . Funktionsname), wie sie von ALGCOMP zum Ersetzen der Relationszeichen in den linken Seiten der Produktionsregeln durch Funktionsnamen in der Algorithmus-Konstruktions-Phase benutzt werden.
Außerdem wird die Liste der Relationssätze in PM-ALGCOMS um NAME erweitert, so daß der neue Satz mit dem MAKEFILE-Kommando automatisch mit PM-ALG auf die Datei LISP.DATA.PM-ALG.nn gerettet wird.
Bem.: Selbstverständlich müssen die Funktionen, die die Relationen realisieren, unabhängig von MAKE-REL definiert werden und auf der Datei LISP.DATA.ALG-EX.nn den Algorithmen zur Ausführungszeit zur Verfügung gestellt werden.
Param.: NAME ist der Name des neuen Relationssatzes.
Ergebnis: An das Atom NAME wird die Assoziationsliste mit den Relationszeichen und den neuen Funktionsnamen als Wert gebunden. PM-ALGCOMS wird unter dem Indikator VARS um NAME erweitert.
Wert: NIL

```
<MAKE-REL
  <NLAMBDA (NAME)
    (TERPRI)
    (RPLACA NAME)
    (PRINTL "EINGABE BEENDEN DURCH: STOP")
    (PRINTL "EINGABE:  <REL-ZEICHEN>  <REL-NAME>: ")
    (PRINTL "BREAKS, Z.B. KLAMMERN,
             MIT PROZENT-ZEICHEN EINGEBEN!")
    (TERPRI)
    (SETBRK (QUOTE (76 110))
            0)
    <PROG (ATOM)
      LOOP(SETQ ATOM (RATOM))
          (COND
```

```
          ((EQ ATOM (QUOTE STOP))
            (PRINTL "MAKE-REL BEENDET. ERGEBNIS:")
            (PRINTL NAME (QUOTE :)
                    (CAR NAME))
            (RETURN))
          (T <RPLACA NAME
                     (APPEND (CAR NAME)
                             (LIST (CONS ATOM (RATOM>
             (GO LOOP>
    (SETBRK (QUOTE (76 110))
            1)
    (COND
      ((MEMB NAME (CADR PM-ALGCOMS))
        NIL)
      (T (NCONC1 (CADR PM-ALGCOMS)
                 NAME)
         (PRINTL "PM-ALGCOMS: VARS UM" NAME "ERWEITERT")))
    NIL>>
```

■

<FEHLER <NLAMBDA (NR . ARGLIST) ... >>

Zweck: Allgemeine Fehlerfunktion, die in allen Modulen des Systems aufgerufen werden kann.
Aufruf: Verschiedene APE-Funktionen
Param.: NR: Eine Fehlernummer
ARGLIST: Eine für diese Nummer evtl. notwendige Argumentliste.
Ergebnis: Das System geht in den Break-Mode.
Wert: NIL

```
<FEHLER
  <NLAMBDA (NR . ARGLIST)
    (COND
      ((EQUAL NR 1)
        (PRINTL "STRUKTURFEHLER: DAS ARGUMENT"
                 (CAR (LAST ARGUMENTE))
                " HAT KEINE  "
                (GETP (CAR ARGLIST)
                      (QUOTE PARAMETER))
                "STRUKTUR, WIE SIE VON" DATSTRUK
                "GEFORDERT WIRD!"))
      ((EQ NR 11)
        (PRINTL "****FEHLER IN IN-BINDE!****")
        (BREAK1 NIL T))
      ((EQ NR 13)
        (PRINTL "****FEHLER IN ARG-PARAM-BIND!****")
        (PRINTL "FKT NUR BIS MAX. 5 PARAMETER DEFINIERT")
        (BREAK1 NIL T))
      ((EQ NR 12)
        (PRINTL "****FEHLER IN ZAHLENLISTE****")
        (BREAK1 NIL T))
      ((EQ NR 10)
        (PRINTL "****SCHLECHTE SPEZIFIKATION !****")
```



```
        (PRINTL "KEINE DER ALTERNATIVEN TRIFFT ZU ! ")
        (BREAK1 NIL T))
      ((EQ NR 15)
        (PRINTL "****FEHLER IN CONST-BIND!****")
        (BREAK1 NIL T))
      ((EQ NR 16)
        (PRINTL "****FEHLER IN UMBENENNUNG!****")
        (BREAK1 NIL T))
      ((EQ NR 18)
        (PRINTL "****FEHLER IN HILFSFKT-VERMERK!****")
        (PRINTL "HILFSFUNKTION ——
                 INDEXZUORDNUNG FEHLERHAFT")
        (BREAK1 NIL T))
      ((EQ NR 19)
        (PRINTL "****FEHLER IN HILFSFKT-VERMERK!****")
        (PRINTL "FUER DIESE REGEL EXISTIERT KEINE ZUORDNUNG")
        (BREAK1 NIL T))
      ((EQ NR 30)
        (PRINTL "****FEHLER IN RELATION****")
        (PRINTL "EINE RELATION DES SATZES
         UNDEF-KL-ALPHORDER EVALUIERT ZUR
         GLEICHHEIT DER ARGUMENTE, OBWOHL
         OBWOHL EINE 'SCHARFE' UNGLEICHHEIT
         GEFORDERT IST.")
        (BREAK1 NIL T))
      (T (PRINTL "****FEHLER -- BISHER KEIN TEXT****")
         (BREAK1 NIL T>>
```

■

<TRACEDRUCK <NLAMBDA (FORM) ... >>

Zweck: TRACEDRUCK evaluiert FORM und schreibt ein Protokoll der Ausführung von FORM auf die Datei PM.PROTOKOLL.name . Hierbei werden auch Angaben über die benötigte Zeit und die verbrauchten CONS-Zellen gemacht.
Zweckmäßig ist TRACEDRUCK bei APE-Implementierungsläufen zu verwenden.
Aufruf: durch den Benutzer in der Form
(TRACEDRUCK (IMPLEMENTIERE adt-eingabedatei)) oder
(TRACEDRUCK (PARAMETRISIERE adt-eingabedatei parameter)) oder
(TRACEDRUCK (IMPLEMENTIERE-ALG alg-eingabedatei))
Param.: FORM gibt den zu evaluierenden LISP-Ausdruck an.
Ergebnis: Auf der Datei PM.PROTOKOLL.name steht ein Protokoll der Evaluierung von FORM zur Verfügung.
Wert: Angaben über die benötigte Zeit und die verbrauchten CONS-Zellen.

```
<TRACEDRUCK
  <NLAMBDA (FORM)
    <PRINTL "PROTOKOLLAUSGABE AUF DATEI:"
            (PACK (LIST (QUOTE PM.PROTOKOLL.)
                        (CADADR FORM>
    (LINELENGTH 80)
    <OUTFILE (PACK (LIST (QUOTE PM.PROTOKOLL.)
                         (CADADR FORM>
    (PRINTL "PM-PROTOKOLL VOM" (DATE))
    (TERPRI)
    (PROG1 (TIME (EVAL FORM)
                 1 0)
           (CLOSEF (PACK (LIST (QUOTE PM.PROTOKOLL.)
                               (CADADR FORM>>
```

■

<u><PRINTL <LAMBDA PRINTLIST ... >></u>

Zweck: Allgemeine Druckfunktion von LAMBDA-spread-Typ, d.h. beliebig viele Argumente von PRINTL werden evaluiert und gedruckt.
Aufruf: Verschiedene APE-Funktionen und direkt von Benutzer.
Param.: Alle an PRINTL übergebenen Argumente werden evaluiert und zu einer Liste zusammengefaßt, die Wert von PRINTLIST ist.
Ergebnis: Der Ausdruck von PRINTLIST.
Wert: NIL

```
<PRINTL
  <LAMBDA PRINTLIST
  (* "EVALUIERT UND DRUCKT BELIEBIG VIELE ARGUMENTE")
    <MAPC PRINTLIST (FUNCTION (NLAMBDA (LISTELEM)
                (PRIN1 LISTELEM)
                (SPACES 1>
    (TERPRI>>
```

■

<u><*COM* <NLAMBDA ... ... >></u>

Bem.: Kommentarfunktion; im Teststadium für ausführlichen Kommentar definiert wie PRINTL, sonst als <NLAMBDA NIL T>.
Die Definition von *COM* wird von den Funktionen COMMENT und NOCOMMENT jeweils umgesetzt.

<u><COMMENT <LAMBDA NIL ... >></u>

Zweck: Einschalten eines zusätzlichen, ausführlichen Kommentars bei APE-Implementierungsläufen, der im LISP-Code der Funktionen durch '*COM*' gekennzeichnet ist.
Aufruf: durch den Benutzer
Ergebnis: definiert die Funktion *COM* als PRINTL, (genauer als *RESERVE). Auch der durch '*COM*' gekennzeichnete Kommentar wird gedruckt.
Wert: NIL

```
<COMMENT
  <LAMBDA NIL
          (* "UMDEF: * => *COM*")
    (PUTD (QUOTE *COM*)
          (GETD (QUOTE *RESERVE)))
    (REMPROP (QUOTE *COM*)
             (QUOTE EXPR>>
```

■

<u><NOCOMMENT <LAMBDA NIL ... >></u>

Zweck: Ausschalten des durch COMMENT aktivierten zusätzlichen Kommentars.
Aufruf: durch den Benutzer
Ergebnis: definiert die Funktion *COM* als (NLAMBDA NIL T). Zusätzlicher Kommentar wird ausgeschaltet.
Wert: NIL

```
<NOCOMMENT
  <LAMBDA NIL
          (* "UMDEF: * => *COM*")
    (DF *COM* NIL T)
    (REMPROP (QUOTE *COM*)
             (QUOTE EXPR>>
```

■

Bem.: COMMENT bzw. NOCOMMENT müssen <u>vor</u> dem Start eines Implementierungslaufes aufgerufen werden, um zusätzlichen Kommentar ein- bzw. auszuschalten.

<u><** <NLAMBDA NIL T>></u>

Bem.: Kommentar, der auch nach Aufruf von (COMMENT) nicht ausgedruckt werden soll.

<*RESERVE <LAMBDA (L) ... >>

Aufruf: Nur als Aufruf von *COM* bei eingeschaltetem COMMENT.
Param.: L ist die Liste der zu druckenden Formen.
Ergebnis: Ausdruck der Formen mittels PRINTL.

```
<*RESERVE
  <LAMBDA (L)
    (PRINTL L)
    T>>
```

<CLEAN <LAMBDA NIL ... >>

Zweck: Aktiviert den Carbage-Collector des LISP-System explizit. Dies kann bei umfangreichen Implementierungen und kleinen LISP-Systemen nötig sein.
Aufruf: durch IMPLEMENTIERE UND PARAMETRISIERE sowie durch IMPLEMENTIERE-ALG, falls GBC-FLAG≠NO-GBC und durch den Benutzer.
Ergebnis: Der Garbage-Collector des LISP-Systems wird aktiviert.
Wert: Angabe über die freigewordenen CONS-Zellen.

```
<CLEAN
  <LAMBDA NIL
    (SETQ A (CONSCOUNT))
    (GBC)
    (SETQ A (DIFFERENCE A (CONSCOUNT)))
    (PRINTL (QUOTE FREED:)
            A
            (QUOTE CELLS>>
```

<REMPROPS <LAMBDA (L) ...>>

Zweck: Zerstören der Property-Lists aller Atome, die in der Liste L zusammengefaßt sind.
Aufruf: Verschiedene APE-Funktionen oder Benutzer.
Param.: L ist eine Liste der Atome, deren Property-List zerstört werden soll.
Ergebnis: Der CDR-Verweis jedes Atoms in L wird auf NIL gesetzt.
Wert: NIL


```
<REMPROPS
  <LAMBDA (L)
    (MAPC L (QUOTE (LAMBDA (A)
                     (RPLACD A>>
```

<MEMB* <LAMBDA (A L) ... >>

Bem.: Wie MEMB, durchsucht jedoch alle Klammerungstiefen.

```
<MEMB*
  <LAMBDA (A L)
    (COND
      ((NOT L)
        NIL)
      ((MEMB A L))
      <(LISTP (CAR L))
        (OR (MEMB* A (CAR L))
            (MEMB* A (CDR L)>
      (T MEMB* A (CDR L>>
```

<NCONC* <LAMBDA L ... >>

Bem.: Wie NCONC, schaltet aber nicht nur zwei, sondern beliebig viele Listen aneinander.

```
<NCONC*
  <LAMBDA L
    (COND
      ((NOT (CDR L))
        (CAR L))
      (T (NCONC (CAR L)
                (APPLY (QUOTE NCONC*)
                       (CDR L>>
```

<APPEND* <LAMBDA L ... >>

Bem.: Wie APPEND, hängt aber nicht nur zwei, sondern beliebig viele Listen aneinander.

```
<APPEND*
  <LAMBDA L
    (COND
      ((NOT (CDR L))
        (CAR L))
      (T (APPEND (CAR L)
                 (APPLY (QUOTE APPEND*)
                        (CDR L>>
```

<ENTCONC <LAMBDA (PRT) ... >>

Zweck: Entbinden aus TCONC-Mode.
Param.: PTR muß Verweiszelle für eine TCONC-Liste sein, d.h. (CAR PTR) muß auf ihr erstes, (CDR PTR) auf ihr letztes Element verweisen.
Ergebnis: Die für PTR übergebene Variable erhält als Wert direkt die davor durch (CAR PTR) verwiesene Liste.
Wert: Diese Liste

```
<ENTCONC
  <NLAMBDA (PTR)
    (SET PRT (CAR (EVAL PTR>>
```

<INTERSECTION <LAMBDA (L1 L2 ...) ... >>

Zweck: Bildung des Durchschnitts zweier Listen.
Param.: L1 und L2 müssen gemäß ALPHORDER geordnete Listen sein.
Wert: Eine Liste, die in der durch ALPHORDER bestimmten Reihenfolge alle Elemente enthält, die in L1 und L2 enthalten sind.
Bem.: Die Elemente werden nicht kopiert.
An der Code-Stelle

```
(NOT (EQ ATOM1 ATOM2))
```

weicht diese Funktion von der Normalversion in INTERLISP ab.

```
<INTERSECTION
  <LAMBDA (L1 L2 ATOM1 ATOM2)
    (SETQ ATOM1 (CAR L1))
    (SETQ ATOM2 (CAR L2))
    (COND
      ((NOT (AND L1 L2))
        NIL)
      ((AND (ALPHORDER ATOM1 ATOM2)
            (NOT (EQ ATOM1 ATOM2)))
        (INTERSECTION (CDR L1) L2))
      <(EQ ATOM1 ATOM2)
        (CONS ATOM1 (INTERSECTION (CDR L1)
                                  (CDR L2>
      ((ALPHORDER ATOM1 ATOM2)
        (INTERSECTION L1 (CDR L2)))
      (T (PRINTL "GLEICHE NAMEN" (CAR ATOM1)
                 "ABER UNGL. VERWEISE" ATOM1
                 "UND" ATOM2
                 "VERBOTEN - INTERSECTION">>
```


5.5. Fehlerbehandlung

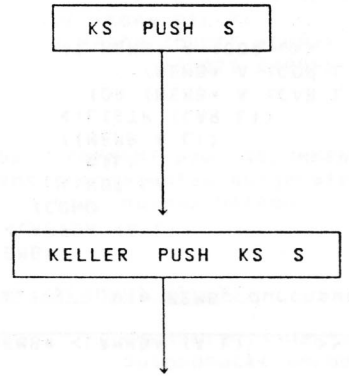
In diesem Abschnitt soll kurz auf die Arten der Fehlerbehandlung im APE-System und auf einige Probleme dabei eingegangen werden.

Eine wesentliche Änderung im ADTCOMP-System gegenüber [EKT 79] ist die intensive Berücksichtigung von Fehlersituationen, die bei Benutzung der implementierten Datentypen auftreten können. Insbesondere unzulässige Operatoren für Inkarnationen oder unzulässige Argumente für Operatoren werden jetzt strikt ausgeschlossen.

Im bisherigen System war es beispielweise möglich, auf eine Inkarnation des Datentyps SCHLANGE Operationen des Typs OFFENELISTE anzuwenden, was allerdings zu anderen Resultaten führte. Außerdem muß im Rahmen des im Kapitel 3 beschriebenen Parametrisierungskonzepts jetzt zusätzlich die Zulässigkeit der Strukturen der Argumente eines Operanden untersucht werden.

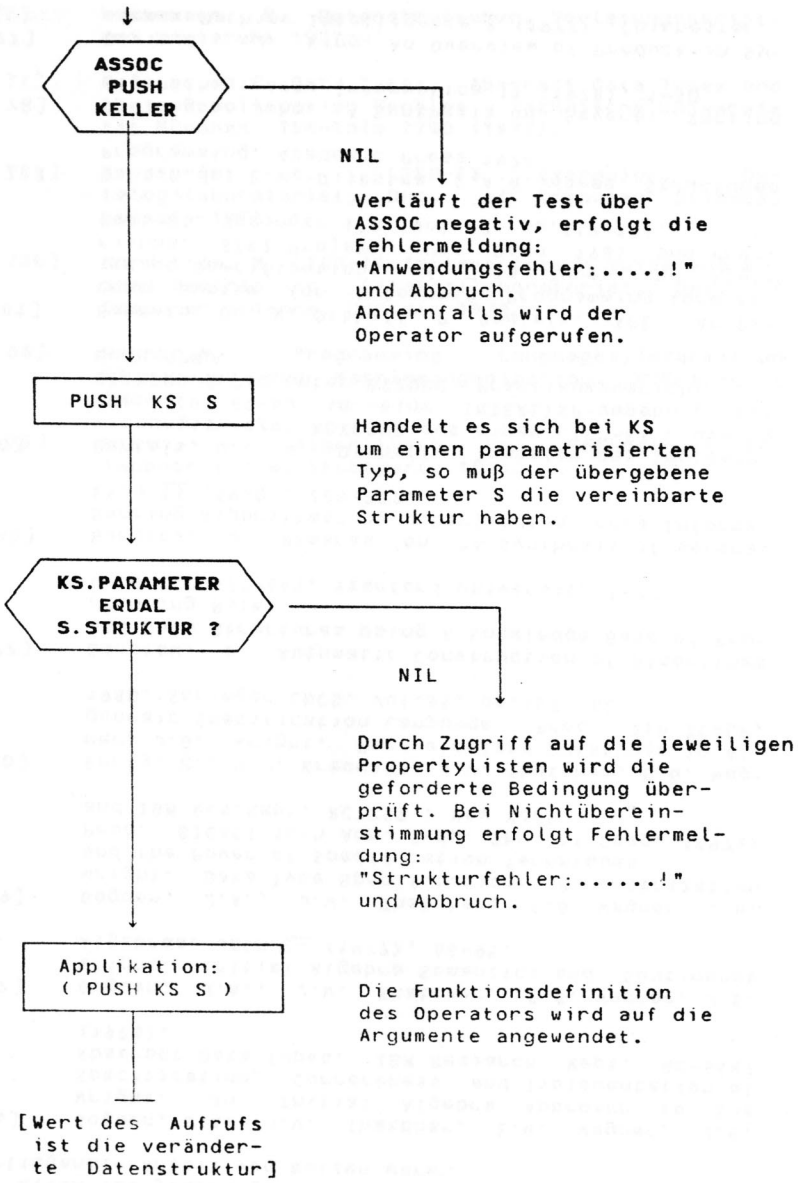
So ergibt sich z.B. für eine Inkarnation KS des parametrisierten Typs KELLER-SCHLANGE folgender Durchlauf (S bezeichnet eine SCHLANGEN-Inkarnation):

Vom Benutzer wird der Inkarnationsname (als Funktionsaufruf in INTERLISP-Syntax) sowie der Name der auf dieser Inkarnation arbeitenden Operation plus Argumente eingegeben:



Dieser Aufruf wird durch die Funktionsdefintion von KS überführt in die alte Aufrufform. Möglich wäre an dieser Stelle eine Überprüfung, ob der Parameter noch 'frei' verwendbar ist.

Der übergebene Operationsname wird unter den für den Datentyp KELLER als zulässig vereinbarten gesucht.



In ALGCOMP haben selbverständlich alle ADTCOMP-Mechanismen weiter Gültigkeit. Weiterhin werden durch die Funktion FEHLER in verschiedenen Fällen Breakpoints erzeugt, wenn entweder von dem Algorithmus eine entsprechende Fehlermeldung berechnet wird, oder , was eigentlich nicht passieren sollte, wenn die Spezifikation nicht alle möglichen Fälle erfaßt hat und der Algorithmus durch alle Alternativen "durchfällt". Die FEHLER-Funktion ist als Standardfehlerpaket konzipiert; durch neue Fehlernummern können weitere Fälle abgedeckt werden.

Ein weiterer Punkt der bereits in der Besprechung der theoretischen Konzepte im Kapitel 2 angeklungen ist, ist die Einführung sogenannter Fehlerelemente für Datenstrukturen. Im APE-System führt ein Inkarnationsaufruf ggf. zu dem (typenlosen) Objekt UNDEF. In verschiedener Hinsicht (s.u.) wäre es aber praktischer, für den Typ KELLER zum Beispiel einen sogenannten UNDEF-KELLER zu definieren. Dieser Keller besitzt als Funktionsdefinition nur ein (PRINTL "UNDEF-KELLER"), als Wert 'UNDEF-KELLER', keine erlaubten Operatoren, aber (auf der Property-Liste) die Strukturkennung KELLER. Insbesondere bei parametrisierten Datentypen ließe sich damit eine einfache Behandlung von Ausnahmefällen erreichen: Eine leere Struktur KELLER-SCHLANGE könnte auf POP noch zu UNDEF-SCHLANGE evaluieren, ohne daß ein Break erzeugt oder eine Sonderbehandlung eingeleitet werden muß. Unter geeigneten Definitionen verhalten sich die Operationen der Datentypen dann "fehlerstrikt": immer, wenn mindestens ein Argument eines Aufrufs eine UNDEF-Struktur aufweist, wird zum entsprechenden UNDEF-Element der Zielstruktur evaluiert.

Dieser Mechanismus würde die Spezifikation von Algorithmen auf parametrisierten Typen um einiges erleichtern. Im gegenwärtigen Systemdesign müssen alle Ausnahmefälle durch geeignete Relationsdefinitionen bearbeitet werden, will man die Spezifikation nicht sehr unübersichtlich gestalten. Im Personalverwaltungssystem PVS im Abschnitt 4.6. hat die zu speziellen Relationssätzen PVSREL-PNR und PVSREL-NAM geführt, in denen ein Großteil der Fallunterscheidungen UNDEF-Situationen berücksichtigt. Würden jetzt ausgezeichnete UNDEF-Strukturen definiert werden, so könnte z. B. bei Verschachtelung von Aufrufen ohne Komplikationen eine sinnvolle Berechnung durchgeführt werden, ohne daß ein "strukturloses" UNDEF eine Abarbeitung aus Kompatibilitätsgründen unterbricht.

Beispielsweise

```
(OFL HINZU 1 (OFL LIES 999))
     geht über in
(OFL HINZU 1 UNDEF-FL)
     geht über in
UNDEF-OFL
```

[OFL = OFFENELISTE-FLEXVERBUND-LISPATOM]

In vielen Spezifikationen ließe sich dies zur Vereinfachung anwenden.

6. Literatur

Nicht jede im folgenden aufgeführte Referenz ist im Text erwähnt. Allen ist jedoch gemeinsam, daß sie bei der Erzeugung der vorliegenden Arbeit von Nutzen waren.

[ADJ 76] Goguen, J.A., J.W. Thatcher, E.G. Wagner, J.B. Wright. An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. IBM Research Rept. RC-6487 (1976).

[ADJ 77] Goguen, J.A., J.W. Thatcher, E.G. Wagner, J.B. Wright. Initial Algebra Semantics and Continuous Algebras. JACM 24 (1977), 68-95.

[ADJ 79] Goguen, J.A., J.W. Thatcher, E.G. Wagner, J.B. Wright. Data Type Specification: Parametrization and the Power of Specification Techniques. Proc. SIGACT 10th Ann. Symp. Thy. of Comp. (1978) and IBM Res.Rept. RC7757 . New York.

[ADJ 80] Ehrig, H., H.J. Kreowski, J.W. Thatcher, E.G. Wagner, J.B. Wright. Parameterized Datatypes in Algebraic Specification Language. Proc. 7th ICALP, 1980. Springer LNCS, Vol.85, pp.157-168.

[BAR 77] Barstow, J. Automatic Construction of Algorithms and Data Structures Using A Knowledge Base of Programming Rules. Report CS-77-641. Stanford University 1977.

[BAR 80] Barstow, J. Remarks on "A Synthesis of several Sorting Algorithms" by J.Darlington. Acta Informatica 13 (1980), 225-227.

[BO 80] Bartels, U., W. Olthoff. Transfer des Implementierungssystems ADTCOMP aus einer LISP-F3 Umgebung auf AEG 80-60 in eine INTERLISP-Umgebung auf SIEMENS 7.760 unter BS2000. Praktikumsbericht. Uni Bonn 1980.

[BOR 81] Bartels, U., W. Olthoff, P. Raulefs. APE: An Expert System for Automatic Programming form Abstract Specifications of Data Types and Algorithms. SEKI-Projekt, Uni Bonn 1981 und Proc. GWAI-81. Springer IFB, Vol.47, pp.112-123.

[DAH 72] Dahl,O.J., E.W. Dijkstra, C.A.R. Hoare. Structured Programming. Academic Press 1972

[DAR 78] Darlington,John. A Synthesis on Several Sorting Algorithms. Acta Informatica 11 (1978), 1-30

[DK 77] Davis,R., J. King. An Overview of Production Systems. Machine Intelligence 8 (1977). Chichester.

[EK 77] Epp, B., D. Kolb. INTERLISP-Programmierhandbuch. Institut für deutsche Sprache, Mannheim 1977, 2. Auflage.

[EKP 78] Ehrig, H., H.J. Kreowski, P. Padawitz. Stepwise Specification and Implementation of Abstract Data Types. Proc. 5th ICALP. Udine 1978.

[EKP 79] Ehrig, H., H.J. Kreowski, P. Padawitz. Algebraic Implementation of Abstract Data Types. Concept, Syntax, Semantics and Correctness. Bericht 79-23. Technische Universität Berlin.

[EKRT 80a] Eigemeier, H., C. Knabe, P. Raulefs, K. Tramer. Automatic Implementation of Algebraic Specifications of Abstract Data Types. Memo SEKI-BN-79 (Nov.79), Universität Bonn und Proc. AISB-80 Conf. on Artificial Intelligence.

[EKRT 80b] Eigemeier, H., C. Knabe, P. Raulefs, K. Tramer. An expert system for automatic coding of abstract data type Specification. Proc. 10th Annual GI-Conf., Springer Informatik Fachberichte 33,1980, pp 431-441.

[EKT 79] Eigemeyer, H., C. Knabe, K. Tramer, Automatische Implementierung abstrakt definierter Datentypen. Diplomarbeit. Uni Bonn 1979.

[GK 78] Guntermann, A., D. Kolb. SIEMENS-INTERLISP. Version 4.0. Usermanual. SIEMENS AG München 1978.

[HAR 75] Haraldson, A. INTERLISP/360-370. Datalogilaboratoriet, Dep. of Computer Science. Uppsala 1975.

[HR 80] Hornung, G., P. Raulefs. Terminal algebra semantics and retractions of abstract data types. Proc. 7th ICALP, 1980. Springer LNCS, Vol.85, pp. 310-323.

[KAN 79] Kant,E. Efficiency Considerations in Programm Synthesis: A Knowledge Base Approach. Report CS-79-755. Stanford University 1979.

[KNU 68] Knuth,D.E. The Art of Computer Programming. Vol. 1: Fundametal Algorithms. Addison-Wesley 1968.

[KNU 73] Knuth,D.E. The Art of Computer Programming. Vol. 3: Sorting and Searching. Addison-Wesley 1973.

[KRE 79] Kreowski, H.J. Algebra für Informatiker. Vorlesungsskript. TU Berlin 1979.

[LO 77] Lockemann, N. Datenstrukturen. Vorlesungsskript. Uni Karlsruhe 1977.

[MAJ 77] Majster, M.E. Data Types, Abstract Data Types and their Specification Problem. Technische Universität München. TUM-Info 7740 (1977).

[NOR 78a] Nordström, M. LISP-F3 Userguide. Datalogilaboratoriet, Dep. of Computer Science. Uppsala 1978.

[NOR 78b] Nordström, M. LISP-F3 Implementation guide and System Description. Datalogilaboratoriet, Dep. of Computer Science. Uppsala 1978.

[WEG 68] Wegner,P. Programming Languages,Information Structures and Machineorganization. McGraw-Hill 1968.

[WIR 75] Wirth, N. Algorithmen und Datenstrukturen. Teubner-Verlag Stuttgart, 1975.

Funktionen-Verzeichnis

Funktion	in Datei	Kapitel	Seite
A			
ABBRUCH	PM-ALG	5.2.1.2.2.	415
ADT-IMPLEM	PM-ADT	5.1.1.1.	302
ALG-KONSTRUKTIONS-PHASE	PM-ALG	5.2.1.3.	430
ALGPARAMP	PM-ALG	5.2.1.2.2.	427
ANF	PM-ADT	5.3.1.	504
	ADT-EX		
APPEND*	PM-ADT	5.4.	524
	PM-REST		
ARG-PARAM-BIND!	PM-ALG	5.2.1.3.	444
ARGNR-OK?	PM-ALG	5.2.1.3.	446
ARGUMENTLISTEP	PM-ALG	5.2.1.2.2.	426
ASSOCMEMB	PM-ADT	5.1.1.2.5.	370
	PM-REST		
ATOMP	PM-ALG	5.2.1.2.2.	428
ATT	ALG-EX	5.3.2.	507
AUFZAEHLVORAUSS-LISTE	PM-ADT	5.1.1.2.5.	358
	PM-REST		
AXIOMENKLASS	PM-ADT	5.1.1.2.2.	318
AXIOMKLASS	PM-ADT	5.1.1.2.2.	318
B			
B1-AUFLOESUNG	PM-ADT	5.1.1.2.5.	336
B1-NB2	PM-ADT	5.1.1.2.5.	337
B2-AUFLOESUNG	PM-ADT	5.1.1.2.5.	335
B-PHASE	PM-ADT	5.1.1.1.	310
BAULISTE	PM-ADT	5.1.1.2.5.	346
	PM-REST		
BENDEF	PM-ADT	5.1.1.2.4.	329
BENUTZUNGSIMPLEM	PM-ADT	5.1.1.2.1.	315
BEREINIGE	PM-ADT	5.1.1.2.3.	327
BOOL-INTERN	PM-ALG	5.2.1.2.2.	410
C			
CHRESET	PM-ADT	5.1.1.2.1.	316
CLEAN	PM-ADT	5.4.	523
	PM-ALG		
	ADT-EX		
COMMENT	PM-ADT	5.4.	521
	PM-ALG		
CONST-BIND!	PM-ALG	5.2.1.3.	448
CONVERT-KWOTE	PM-ALG	5.4.	516
D			
DATENTYPP	PM-ALG	5.2.1.2.2.	411
DT-LISP-OPP	PM-ALG	5.2.1.2.2.	424
DT-OPERATIONP	PM-ALG	5.2.1.2.2.	425
DUPLIZIERE	ALG-EX	5.3.2.	505
DURCHL	PM-ADT	5.1.1.2.3.	323

E			
ECKIG	PM-ADT	5.1.1.2.5.	367
	PM-REST		
EIGVORAUSS	PM-ADT	5.1.1.2.5.	355
	PM-REST		
EINEEBENE	PM-ADT	5.1.1.2.5.	370
	PM-REST		
EINFUNK	PM-ADT	5.1.1.2.4.	329
EINGEDFUNK	PM-ADT	5.1.1.2.4.	330
EINSORT	PM-ADT	5.1.1.2.5.	341
EKL	PM-ADT	5.1.1.2.5.	371
	PM-ALG		
ENTCONC	PM-ADT	5.4.	525
	PM-ALG		
ENTKLAMMERUNG	PM-ALG	5.2.1.3.	445
EQ-KWOTE?	PM-ALG	5.2.1.3.	450
EXISTIERT	PM-ADT	5.1.1.2.5.	354
F			
FEHLER	ADT-EX	5.4.	519
	PM-ALG		
	ALG-EX		
	ADT-EX		
FIBO-ZAHL	ALG-EX	5.3.2.	506
FKT-SYMBOL-IN	PM-ALG	5.2.1.2.2.	413
FLAT	PM-ADT	5.1.1.2.5.	345
FOLGERE	PM-ADT	5.1.1.2.5.	366
	PM-REST		
FORM-BIND!	PM-ALG	5.2.1.3.	445
FUNKKLASS	PM-ADT	5.1.1.2.2.	317
FUNKSKLASS	PM-ADT	5.1.1.2.2.	316
FUNKTIONS-AUFRUF-NRP	PM-ALG	5.2.1.2.2.	422
FUNKTIONS-AUFRUFP	PM-ALG	5.2.1.2.2.	423
G			
GED-ZWISCHEN	PM-ADT	5.1.1.2.4.	330
	ADT-EX		
GLEICH	PM-ADT	5.1.1.2.5.	370
	PM-REST		
GLEICH-ALPHP	ALG-EX	5.3.2.	512
GLEICH-NAMP	ALG-EX	5.3.2.	515
GLEICH-PNRP	ALG-EX	5.3.2.	513
GLEICHP	ALG-EX	5.3.2.	510
GREATER-ALPHP	ALG-EX	5.3.2.	511
GREATEREQ-ALPHP	ALG-EX	5.3.2.	511
GREATEREQP	PM-ADT	5.3.2.	510
	ALG-EX		
	ADT-EX		
GROESSER-ALPHP	ALG-EX	5.3.2.	512
GROESSER-NAMP	ALG-EX	5.3.2.	515
GROESSER-PNRP	ALG-EX	5.3.2.	513
GROESSERGLEICH-ALPHP	ALG-EX	5.3.2.	512
GROESSERGLEICH-NAMP	ALG-EX	5.3.2.	516
GROESSERGLEICH-PNRP	ALG-EX	5.3.2.	513
GROESSERGLEICHP	ALG-EX	5.3.2.	511

GROESSERP	ALG-EX	5.3.2.	510
H			
HILFSALGORITHMUSP	PM-ALG	5.2.1.2.2.	427
HILFSFKT-VERMERK!	PM-ALG	5.2.1.3.	441
I			
I-PHASE	PM-ADT	5.1.1.1.	310
IMPLEMENTIERE	PM-ADT	5.1.1.1.	302
IMPLEMENTIERE-ALG	PM-ALG	5.2.1.1.	394
IMPLK	PM-ADT	5.1.1.2.5.	332
IN-BINDE!	PM-ALG	5.2.1.3.	448
IN-LISTE?	PM-ALG	5.2.1.3.	450
INDBIND	PM-ADT	5.1.1.2.5.	353
	PM-REST		
INDEX-CHECK	PM-ALG	5.2.1.3.	451
INIT	ADT-EX	5.3.1.	501
INK+OPP	PM-ALG	5.2.1.2.2.	426
INKARNATION-?	PM-ALG	5.2.1.3.	452
INSERT	ADT-EX	5.3.1.	502
INTEGERP	PM-ALG	5.2.1.2.2.	429
INTERSECTION	PM-ADT	5.4.	525
K			
KLEINER-ALPHP	ALG-EX	5.3.2.	512
KLEINER-NAMP	ALG-EX	5.3.2.	515
KLEINER-PNRP	ALG-EX	5.3.2.	513
KLEINERGLEICH-ALPHP	ALG-EX	5.3.2.	512
KLEINERGLEICH-NAMP	ALG-EX	5.3.2.	516
KLEINERGLEICH-PNRP	ALG-EX	5.3.2.	514
KLEINERGLEICHP	ALG-EX	5.3.2.	511
KLEINERP	ALG-EX	5.3.2.	510
KOPOP	PM-ADT	5.1.1.2.5.	345
	PM-REST		
KOPUSH	PM-ADT	5.1.1.2.5.	345
	PM-REST		
KOTOP	PM-ADT	5.1.1.2.5.	346
	PM-REST		
KWOTE-ATOMP	PM-ALG	5.2.1.2.2.	422
L			
L-PHASE	PM-ADT	5.1.1.1.	310
LBISEZ	PM-ADT	5.1.1.2.5.	372
	PM-ALG		
LESS-ALPHP	ALG-EX	5.3.2.	511
LESSEQ-ALPHP	ALG-EX	5.3.2.	511
LESSEQP	PM-ADT	5.3.2.	510
	ALG-EX		
	ADT-EX		
LETZTEPH	PM-ADT	5.1.1.3.	375
LINKESEITEP	PM-ALG	5.2.1.2.2.	418
LINKSTEST	PM-ADT	5.1.1.2.2.	319
LISP-DUPLIKATE	PM-ALG	5.2.1.3.	442
LISP-FKTP	PM-ALG	5.2.1.2.2.	429
LISP-OPERATIONP	PM-ALG	5.2.1.2.2.	425
LISP-PHASE	PM-ALG	5.2.1.4.	464

LISPNAMEN-VERMERK!	PM-ALG	5.2.1.3.	441
LISTSIZE	ALG-EX	5.3.2.	508
LITATOMP	PM-ALG	5.2.1.2.2.	428
LSTATTZ	PM-ADT	5.1.1.2.5.	371
	PM-ALG		
M			
MAKE-REL	PM-ALG	5.4.	517
MEMB*	PM-ADT	5.4.	524
	PM-REST		
	PM-ALG		
MENGKLASS	PM-ADT	5.1.1.2.2.	317
MODIFIZIERE-STEUERUNG	PM-ALG	5.2.1.3.	463
N			
NAMBILD	PM-ADT	5.1.1.2.1.	315
NB1-B2	PM-ADT	5.1.1.2.5.	337
NCONC*	PM-ADT	5.4.	524
NEUE-RSEITEN	PM-ALG	5.2.1.3.	452
NEXTATOM	PM-ALG	5.2.1.2.2.	429
NICHTIND	PM-ADT	5.1.1.2.5.	352
	PM-REST		
NO-KWOTE	PM-ALG	5.2.1.3.	451
NOCOMMENT	PM-ADT	5.4.	522
	PM-ALG		
NOT-ALL-NIL?	PM-ALG	5.2.1.3.	455
O			
OPERATORENKLASS	PM-ADT	5.1.1.2.3.	322
OPERATORKLASS	PM-ADT	5.1.1.2.3.	323
OR-VEREINF	PM-ADT	5.1.1.2.5.	338
	PM-REST		
P			
PA	PM-ADT	5.1.1.3.	374
PAR-REK-AUFRP	PM-ALG	5.2.1.2.2.	419
PARALLEL-REK-AUFBAU	PM-ALG	5.2.1.3.	454
PARAMETER-REIHENFOLGE?	PM-ALG	5.2.1.3.	447
PARAMETER-UEBERPRUEFUNG	PM-ALG	5.2.1.2.2.	407
PARAMETERLISTEP	PM-ALG	5.2.1.2.2.	423
PARAMETERP	PM-ALG	5.2.1.2.2.	424
PARAMETRISIERE	PM-ADT	5.1.1.1.	301
PARSERFEHLER	PM-ALG	5.2.1.2.2.	429
PATTERNLAENGE	PM-ALG	5.2.1.3.	462
PB	PM-ADT	5.1.1.3.	375
PF	PM-ADT	5.1.1.3.	373
PI	PM-ADT	5.1.1.3.	375
PMFEHLER	PM-ADT	5.1.1.2.5.	343
	PM-REST		
PRINTL	PM-ADT	5.4.	521
	PM-ALG		
	ALG-EX		
	ADT-EX		
PRINTREGLIST	PM-ADT	5.1.1.2.5.	344
PROBIERPRUEFLING	PM-ADT	5.1.1.2.2.	321
	PM-REST		

PROBIERREGEL	PM-ADT	5.1.1.2.5.	331
PRUEFCHAREIGAX	PM-ADT	5.1.1.2.5.	320
PRUEFCHAREIGOP	PM-ADT	5.1.1.2.3.	323
PU	PM-ADT	5.1.1.3.	374
R			
RADT	PM-ADT	5.1.1.1.	303
RALGSPEC	PM-ALG	5.2.1.2.1.	397
RATOM-NE	PM-ALG	5.2.1.2.2.	406
RATOM-NE-KOMSEM	PM-ADT	5.1.1.2.1.	313
RATOM-NE-SEM	PM-ADT	5.1.1.2.1.	313
RECHTESEITEP	PM-ALG	5.2.1.2.2.	419
RECHTSTEST	PM-ADT	5.1.1.2.2.	319
REGELERFOLG	PM-ADT	5.1.1.2.5.	342
	PM-REST		
REGELMENGEP	PM-ALG	5.2.1.2.2.	417
REGELP	PM-ALG	5.2.1.2.2.	418
REGELPARSER	PM-ALG	5.2.1.2.2.	415
REGELPRUEF	PM-ADT	5.1.1.2.5.	348
	PM-REST		
REGELVORAUSWAHL	PM-ADT	5.1.1.2.3.	328
REL-BIND!	PM-ALG	5.2.1.3.	446
RELP	PM-ALG	5.2.1.2.2.	421
RELWERTP	PM-ALG	5.2.1.2.2.	422
REMPROPS	PM-ADT	5.4.	523
	PM-ALG		
	ADT-EX		
REST	PM-ADT	5.3.1.	504
	ADT-EX		
RGLEICHP	PM-ALG	5.2.1.2.2.	426
RGLEICHP-NS	PM-ALG	5.2.1.2.2.	427
S			
S1-STEUERUNG	PM-ALG	5.2.1.3.	458
S2-STEUERUNG	PM-ALG	5.2.1.3.	458
S4-STEUERUNG	PM-ALG	5.2.1.3.	458
S5-STEUERUNG	PM-ALG	5.2.1.3.	458
S6-STEUERUNG	PM-ALG	5.2.1.3.	458
SAMMELGED	PM-ADT	5.1.1.2.1.	314
SCHRITT-1	PM-ALG	5.2.1.3.	434
SCHRITT-2	PM-ALG	5.2.1.3.	435
SCHRITT-3	PM-ALG	5.2.1.3.	436
SCHRITT-4	PM-ALG	5.2.1.3.	436
SCHRITT-5	PM-ALG	5.2.1.3.	437
SCHRITT-6	PM-ALG	5.2.1.3.	439
SET-ENVIRONMENT	PM-ALG	5.2.1.4.	470
SETZEIG	PM-ADT	5.1.1.2.5.	340
	PM-REST		
SETZEIG*	PM-ADT	5.1.1.2.5.	341
	PM-REST		
SKIPSEM	PM-ADT	5.1.1.2.1.	314
STANDARD-BIND!	PM-ALG	5.2.1.3.	443
STEUERUNGS-RUMPF	PM-ALG	5.2.1.3.	460
STREICHE	PM-ADT	5.1.1.2.5.	366
	PM-REST		
STRICHLISTE	PM-ADT	5.1.1.2.5.	373

SUCHBI	PM-ADT	5.1.1.2.3.	324
SUCHE	PM-ADT	5.1.1.2.3.	326
	PM-REST		
SUCHEAUSDR	PM-ADT	5.1.1.2.5.	347
	PM-REST		
SUCHWERT	PM-ADT	5.1.1.2.5.	358
	PM-REST		
T			
TEST	ALG-EX	5.3.2.	503
	ADT-EX		
TRACEDRUCK	PM-ADT	5.4.	520
	PM-ALG		
U			
UEBERS	PM-ADT	5.1.1.2.3.	321
UMBENENNUNG!	PM-ALG	5.2.1.3.	457
UNGLEICH	PM-ADT	5.1.1.2.5.	370
UNGLEICH-ALPHP	ALG-EX	5.3.2.	512
UNGLEICH-NAMP	ALG-EX	5.3.2.	515
UNGLEICH-PNRP	ALG-EX	5.3.2.	513
UNGLEICHP	ALG-EX	5.3.2.	510
V			
VARBIND	PM-ADT	5.1.1.2.5.	368
	PM-REST		
VARIABLEP	PM-ADT	5.1.1.2.5.	369
	PM-REST		
VERGLEICH	PM-ADT	5.1.1.2.5.	362
	PM-REST		
VERGLEICHP	PM-ALG	5.2.1.2.2.	420
VGLSEITEP	PM-ALG	5.2.1.2.2.	421
VM-PRUEFLINGNAMEN	PM-ADT	5.1.1.2.5.	331
	PM-REST		
VORAUSSTEST	PM-ADT	5.1.1.2.5.	350
	PM-REST		
W			
WELOPS	PM-ADT	5.1.1.2.5.	344
WERTP	PM-ALG	5.2.1.2.2.	420
WERTVGL	PM-ADT	5.1.1.2.5.	360
	PM-REST		
WOWEITER	PM-ADT	5.1.1.2.3.	325
Z			
ZAHLENLISTE	PM-ALG	5.2.1.3.	457
ZKL	PM-ADT	5.1.1.2.5.	340
	PM-REST		
Sonderzeichen			
&-PHASE	PM-ADT	5.1.1.1.	309
**	PM-ADT	5.4.	522
COM	PM-ADT	5.4.	521
	PM-ALG		
*RESERVE	PM-ADT	5.4.	522
	PM-ALG		

->-PHASE	PM-ADT	5.1.1.1.	308
?	PM-ADT	5.1.1.2.5.	353
	PM-REST		
=-PHASE	PM-ADT	5.1.1.1.	309

Statistische Daten zur Regelanwendung und Implementierung

Die im folgenden angegebenen Werte für CPU-Zeiten und CONS-Zellen stellen Mittelwerte aus bis zu 6 Einzeldaten dar.

A) Statistische Daten zu ADTCOMP-Implementierungen:

Implementierung des parametrisierten abstrakten Datentyp OFFENELISTE-INTEGER (siehe 3.2.):

verbrauchte CONS-Zellen: 128449 CONSES

verbrauchte CPU-Zeit: 216.503998 SECONDS

angewandte Regeln:

Regel	Häufigkeit
->01	3
->02A	1
->03	3
->07A	2
->08	4
=01	1
=03	3
=04	3
=06A	1
=06B	1
=06C	1
=10	3
&01	1
&06	3
&11B	1
&13B	1
&15B	2
&21	1
&29	1
&32	1
I38	1
I02	3
I08	2
I08B	3
I33	17
I10	1
I15	2
I06	3
I23	1
I13	1
I36	1
B02	3
B07	3

B) Statistische Daten zu ALGCOMP - Implementierungen:

B1) Verbrauch von CONS-Zellen und CPU-Zeit pro Implementierung

Die verbrauchten CONS-Zellen und die benötigte CPU-Zeit wird innerhalb von TRACEDRUCK durch die INTERLISP-Funktion TIME ausgegeben.

B1-Tabelle 1 Algorithmus	verbrauchte CONS-Zellen [CONSES]	verbrauchte CPU-Zeit [SECONDS]
FAK	9341	9.766000
GGT	13988	15.079000
KGV	19850	22.323990
SEQUENTIAL-SEARCH	13592	14.648999
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	15256	16.800995
BINARY-SEARCH	30171	30.326996
FIBONACCIAN-SEARCH	61283	60.795990
MERGE-SORT	33144	36.698990
BUBBLE-SORT	32982	35.584991
STRAIGHT-INSERTION-SORT	44837	48.774994
STRAIGHT-SELECTION-SORT	10924	11.842999
QUICK-SORT	111722	121.140991
CHANGE	46170	50.405991
MAXLIST	18284	19.678986
INDMAX	21584	22.977997

Fortsetzung B1-Tabelle 1:

B1-Tabelle 1 Algorithmus	verbrauchte CONS-Zellen [CONSES]	verbrauchte CPU-Zeit [SECONDS]
BUBBLE-PNR	33271	36.754990
MERGE-PNR	34770	38.776993
ABFRAGEN	17125	19.668991
SAMMMELN	28821	31.889990
GIB-WERT	6528	8.000999
SAMMELN-2	30965	34.034988
EINFUEGEN	16955	19.909988
ADRESSENLISTE	29969	32.809998
FIBONACCIAN-SEARCH-PNR	64692	69.536987
GIB-ADRESSE	27900	30.21199

B2) Angewandte Regeln und Regelanwendungsfaktor pro Implementierung

Angewandte Regeln: Es werden die Regeln aufgeführt, die bei einer Implementierung mindestens einmal angewandt wurden. Bezüglich der Häufigkeit der Anwendung einer Regel siehe B3.
Regelanwendungsfaktor: Der Regelanwendungsfaktor ist der Quotient aus den ALGCOMP-globalen Variablen SUCC-CNT und ATTEMPT-CNT. ATTEMPT-CNT zählt die bei einer Implementierung versuchten Regelanwendungen, SUCC-CNT die davon erfolgreichen.

B2-Tabelle 1 Algorithmus	Angewandte Regeln	Regel-anwendungs-faktor
FAK	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-2 S5-5 S6-1 S6-21	0.680000
GGT	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.756757
KGV	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-54 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.777778
SEQUENTIAL-SEARCH	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.794118

Fortsetzung B2-Tabelle 1:

B2-Tabelle 1 Algorithmus	Angewandte Regeln	Regel-anwendungs-faktor
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	S1-1 S1-2 S2-1 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.783784
BINARY-SEARCH	S1-1 S2-1 S2-2 S2-9 S3-1 S4-2 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.707692
FIBONACCIAN-SEARCH	S1-1 S1-2 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.680328
MERGE-SORT	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9 S5-1 S5-31 S5-8 S5-9 S6-1 S6-21	0.698630
BUBBLE-SORT	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-4 S5-5 S5-9 S6-1 S6-21	0.671233
STRAIGHT-INSERTION-SORT	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9 S5-1 S5-2 S5-31 S5-5 S5-8 S5-9 S6-1 S6-21	0.780220

Fortsetzung B2-Tabelle 1:

B2-Tabelle 1 Algorithmus	Angewandte Regeln	Regel-anwendungs-faktor
STRAIGHT-SELECTION-SORT	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9 S5-33 S5-8 S6-1 S6-21	0.642857
QUICK-SORT	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-3 S4-9 S5-1 S5-2 S5-4 S5-5 S5-9 S5-D1 S6-1 S6-21 S6-22	0.645914
CHANGE	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-4 S4-54 S4-9 S5-1 S5-31 S5-8 S5-9 S6-21 S6-23	0.627273
MAXLIST	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-55 S4-9 S5-1 S5-2 S5-31 S5-5 S5-9 S6-1 S6-21	0.711111
INDMAX	S1-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-55 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-21	0.686274

Fortsetzung B2-Tabelle 1:

B2-Tabelle 1 Algorithmus	Angewandte Regeln	Regel-anwendungs-faktor
BUBBLE-PNR	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-4 S5-5 S5-9 S6-1 S6-21	0.671233
MERGE-PNR	S1-1 S2-1 S2-9 S3-1 S4-1 S4-2 S4-51 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-23	0.698630
ABFRAGEN	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-23	0.659091
SAMMELN	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-2 S5-4 S5-5 S5-8 S5-9 S6-1 S6-23	0.634921
GIB-WERT	S1-1 S2-1 S2-9 S3-1 S4-4 S6-23	0.545455
SAMMELN-2	S1-1 S2-2 S2-2 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-2 S5-4 S5-5 S5-8 S5-9 S6-1 S6-23	0.656716

Fortsetzung B2-Tabelle 1:

B2-Tabelle 1 Algorithmus	Angewandte Regeln	Regel-anwendungs-faktor
EINFUEGEN	S1-1 S1-2 S2-1 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-23	0.674419
ADRESSENLISTE	S1-1 S2-1 S2-2 S2-9 S3-1 S4-1 S4-2 S4-9 S5-1 S5-2 S5-31 S5-33 S5-5 S5-8 S5-9 S6-1 S6-21	0.666667
FIBONACCIAN-SEARCH-PNR	S1-1 S1-2 S2-2 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-2 S5-5 S5-9 S6-1 S6-23	0.680328
GIB-ADRESSE	S1-1 S2-2 S2-9 S3-1 S4-1 S4-4 S4-9 S5-1 S5-31 S5-8 S5-9 S6-1 S6-23	0.575342

Berücksichtigt man alle erfolgreichen und alle versuchten Regelanwendungen in allen 25 Implementierungen, so ergibt sich ein "overall"-Regelanwendungsfaktor von 0.678313.

B3) Regelanwendung mit Häufigkeiten

Die Tabelle zeigt, welche Regel bei welcher Algorithmus-Implementierung angewandt wurde und welche Algorithmus-Implementierung welche Regel verwendet hat. Ein leeres Feld bedeutet "Regel nicht angewandt"; die Zahlen geben die Anzahl der Anwendungen einer Regel an.

Werte für die Regeln S4-52, S4-53 und S5-32 sind nicht angefallen, da diese Regeln zur Implementierung der dokumentierten Algorithmen nicht benötigt werden. Teilweise werden sie für Spezifikationen der Datei ALGORITHMEN.SPEC benötigt, die hier nicht protokolliert sind, teilweise wurden sie für temporäre Spezifikationen geschrieben, die sich als ineffizient oder falsch erwiesen haben. Trotzdem hat es Sinn, diese Regeln weiter in den Basen zu lassen, da ihre Wirkung nicht von anderen Regeln überdeckt wird.

B3-Tabelle 1	S1- und S2-Regeln				
Algorithmus	S1-1	S1-2	S2-1	S2-2	S2-9
FAK	2			2	2
GGT	3			3	3
KGV	3		2	3	5
SEQUENTIAL-SEARCH	3		4		4
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	2	1	5		5
BINARY-SEARCH	4		3	1	4
FIBONACCIAN-SEARCH	4	1	4	7	11
MERGE-SORT	4		7		7
BUBBLE-SORT	4		6	2	8
STRAIGHT-INSERTION-SORT	5		12		12
STRAIGHT-SELECTION-SORT	2		2		2
QUICK-SORT	9		7	15	22
CHANGE	6			7	7
MAXLIST	3		3		3
INDMAX	3		3		3

Fortsetzung B3-Tabelle 1:

B3-Tabelle 1	S1- und S2-Regeln				
Algorithmus	S1-1	S1-2	S2-1	S2-2	S2-9
BUBBLE-PNR	4		6	2	8
MERGE-PNR	4		7		7
ABFRAGEN	3		4		4
SAMMELN	3		3	2	5
GIB-WERT	2		2		2
SAMMELN-2	3		3	2	5
EINFUEGEN	2	1	5		5
ADRESSENLISTE	3		3	2	5
FIBONACCIAN-SEARCH-PNR	4	1		11	11
GIB-ADRESSE	4			4	4

B3-Tabelle 2	S3- und S4-Regeln				
Algorithmus	S3-1	S4-1	S4-2	S4-3	S4-4
FAK	2	1	1		
GGT	3	2	1		
KGV	3	2	1		
SEQUENTIAL-SEARCH	3	1	2		
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	3	2	1		
BINARY-SEARCH	4	2	2		
FIBONACCIAN-SEARCH	5	3	2		
MERGE-SORT	4	3	1		
BUBBLE-SORT	4	3	1		
STRAIGHT-INSERTION-SORT	5	4	1		
STRAIGHT-SELECTION-SORT	2	1	1		
QUICK-SORT	9	8	2	1	
CHANGE	6	4	1		1
MAXLIST	3	2	1		
INDMAX	3	2	1		
BUBBLE-PNR	4	3	1		
MERGE-PNR	4	3	1		
ABFRAGEN	3	1			2
SAMMELN	3	2			1
GIB-WERT	2				2
SAMMELN-2	3	2			1
EINFUEGEN	3	1			2
ADRESSENLISTE	3	2	1		
FIBONACCIAN-SEARCH-PNR	5	3			2
GIB-ADRESSE	4	3			1

B3-Tabelle 3	S4-Regeln			
Algorithmus	S4-51	S4-54	S4-55	S4-9
FAK				1
GGT				2
KGV		1		2
SEQUENTIAL-SEARCH				1
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES				1
BINARY-SEARCH				2
FIBONACCIAN-SEARCH				3
MERGE-SORT	3			3
BUBBLE-SORT	2			3
STRAIGHT-INSERTION-SORT	1			4
STRAIGHT-SELECTION-SORT				1
QUICK-SORT				8
CHANGE		1		4
MAXLIST			1	2
INDMAX			1	2
BUBBLE-PNR				3
MERGE-PNR	3			3
ABFRAGEN				1
SAMMELN				2
GIB-WERT				
SAMMELN-2				2
EINFUEGEN				1
ADRESSENLISTE				2
FIBONACCIAN-SEARCH-PNR				3
GIB-ADRESSE				3

B3-Tabelle 4	S5-Regeln			
Algorithmus	S5-D1	S5-1	S5-2	S5-31
FAK			2	
GGT		2	2	
KGV		5	1	
SEQUENTIAL-SEARCH		2	1	
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES		2	1	
BINARY-SEARCH		6	4	
FIBONACCIAN-SEARCH		13	8	
MERGE-SORT		2		7
BUBBLE-SORT		6	2	
STRAIGHT-INSERTION-SORT		6	1	5
STRAIGHT-SELECTION-SORT				
QUICK-SORT	12	26	8	
CHANGE		13		3
MAXLIST		3	2	1
INDMAX		5	2	1
BUBBLE-PNR		6	2	
MERGE-PNR		2		7
ABFRAGEN		3	1	
SAMMELN		5	2	
GIB-WERT				
SAMMELN-2		7	2	
EINFUEGEN		2	1	
ADRESSENLISTE		6	2	1
FIBONACCIAN-SEARCH-PNR		13	8	
GIB-ADRESSE		6		3

B3-Tabelle 5	S5-Regeln				
Algorithmus	S5-33	S5-4	S5-5	S5-8	S5-9
FAK			2		
GGT			2		2
KGV			1		3
SEQUENTIAL-SEARCH			1		2
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES			1		2
BINARY-SEARCH			4		6
FIBONACCIAN-SEARCH			8		9
MERGE-SORT				7	2
BUBBLE-SORT		1	2		3
STRAIGHT-INSERTION-SORT	2		1	5	3
STRAIGHT-SELECTION-SORT	2			2	
QUICK-SORT		2	8		20
CHANGE				3	7
MAXLIST			2		3
INDMAX			2		4
BUBBLE-PNR					3
MERGE-PNR					2
ABFRAGEN					3
SAMMELN					5
GIB-WERT					
SAMMELN-2					7
EINFUEGEN					2
ADRESSENLISTE					6
FIBONACCIAN-SEARCH-PNR					9
GIB-ADRESSE					3

B3-Tabelle 6 Algorithmus	S6-Regeln			
	S6-1	S6-21	S6-22	S6-23
FAK	1	1		
GGT	2	1		
KGV	2	1		
SEQUENTIAL-SEARCH	1	2		
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	1	2		
BINARY-SEARCH	2	2		
FIBONACCIAN-SEARCH	3	2		
MERGE-SORT	3	1		
BUBBLE-SORT	3	1		
STRAIGHT-INSERTION-SORT	4	1		
STRAIGHT-SELECTION-SORT	1	1		
QUICK-SORT	6	2	1	
CHANGE	4	1		1
MAXLIST	2	1		
INDMAX	2	1		
BUBBLE-PNR	3	1		
MERGE-PNR	3	1		
ABFRAGEN	1			2
SAMMELN	2			1
GIB-WERT				2
SAMMELN-2	2			1
EINFUEGEN	1			2
ADRESSENLISTE	2	1		
FIBONACCIAN-SEARCH-PNR	3			2
GIB-ADRESSE	3			1

B3-Tabelle 7 Regelanwendung/Schritt Algorithmus	Schritt						
	S1	S2	S3	S4	S5	S6	SUM
FAK	2	4	2	3	4	2	17
GGT	3	6	3	5	8	3	28
KGV	3	10	3	6	10	3	35
SEQUENTIAL-SEARCH	3	8	3	4	6	3	27
SEQUENTIAL-SEARCH-ON-ORDERED-TABLES	3	10	3	4	6	3	29
BINARY-SEARCH	4	8	4	6	20	4	46
FIBONACCIAN-SEARCH	5	22	5	8	38	5	83
MERGE-SORT	4	14	4	10	18	4	54
BUBBLE-SORT	4	16	4	9	14	4	51
STRAIGHT-INSERTION-SORT	5	24	5	10	21	5	70
STRAIGHT-SELECTION-SORT	2	4	2	3	4	2	17
QUICK-SORT	9	44	9	19	76	9	166
CHANGE	6	14	6	11	26	6	69
MAXLIST	3	6	3	6	11	3	32
INDMAX	3	6	3	6	14	3	35
BUBBLE-PNR	4	16	4	7	11	4	46
MERGE-PNR	4	14	4	10	11	4	47
ABFRAGEN	3	8	3	4	7	3	28
SAMMELN	3	10	3	5	12	3	36
GIB-WERT	2	4	2	2		2	12
SAMMELN-2	3	10	3	5	16	3	40
EINFUEGEN	3	10	3	4	5	3	28
ADRESSENLISTE	3	10	3	5	15	3	39
FIBONACCIAN-SEARCH-PNR	5	22	5	8	30	5	75
GIB-ADRESSE	4	8	4	7	12	4	41

Dateiverzeichnis

Im folgenden sind die Dateien des APE-System aufgeführt und mit kurzen Inhaltsbeschreibungen versehen. Darauf folgen Angaben bezüglich dieser Diplomarbeit und abschließend Hinweise zur Datenhaltung auf den Magnetbändern und zum Start des LISP-Systems.

Dateiname	enthält:
LISP.DATA.PM-ADT.nn	Patternmatcher ALGCOMP
LISP.DATA.ADT-REGELN.EXTERN	ALGCOMP-Regeln in Externdarstellung
LISP.DATA.ADT-REGELN.INTERN	ALGCOMP-Regeln in Interndarstellung
LISP.DATA.RMP.nn	Regelmanipulationspaket für ADTCOMP-Regeln
LISP.DATA.ADT-EX.nn	Anwendungsfunktionen für ADTCOMP-Implementierungen
LISP.DATA.PM-ALG.nn	ALGCOMP-Patternmatcher
LISP.DATA.ALG-REGELN.nn	ALGCOMP-Regeln
LISP.DATA.ALG-EX.nn	Anwendungsfunktionen für ALGCOMP-Implementierungen
DATENTYPEN.SPEC	Spezifikationen für ADTCOMP
DATENTYPEN.IMPLEM	Implementierungen des ADTCOMP-Systems
DATENTYPEN.IMPLEM.PARAM	Implementierungen des ADTCOMP-Systems (parametrisiert)
ALGORITHMEN.SPEC	Spezifikationen für ALGCOMP
ALGORITHMEN.IMPLEM	Implementierungen des ALGCOMP-Systems
PVS.PROTOKOLLE	Implementierungsprotokolle für Algorithmen des PVS-Systems (4.6.)

Für die vorliegende Arbeit sind folgende Dateien von Interesse:

DIPL.1	Deckblatt, Einleitung, Inhaltsverzeichnis
DIPL.2	Kapitel 2.
DIPL.3	Kapitel 3.
DIPL.4.1	Kapitel 4.1.
DIPL.4.2	Kapitel 4.2.
DIPL.4.3	Kapitel 4.3.
DIPL.4.4	Kapitel 4.4.
DIPL.4.5	Kapitel 4.5.
DIPL.4.6	Kapitel 4.6.
DIPL.5.1	Kapitel 5.1.
DIPL.5.2	Kapitel 5.2.
DIPL.5.3	Kapitel 5.3.
DIPL.5.4	Kapitel 5.4.
DIPL.5.5	Kapitel 5.5.
DIPL.6	Kapitel 6., Anhang 1, Anhang 2 und Anhang 3
DIPL.PRINT	Batch-Job zum Ausdrucken dieser Arbeit
EDOR.Q	Edor-Kommandos für die Dateien DIPL.5.2 u. DIPL.5.5
LISP.NDFILE	Zeichensätze zum Ausdruck aller Kapitel

Um mit den Dateien und den Bändern RS0277 und RS0838 zu arbeiten, sind folgende Befehle nötig:

Gesamten Platteninhalt + Bandinhalt auf Band retten

```
/option msg=fl                    (informiert über System-
                                   meldungen an die Operateure)
/do $p.save,upd,tape1=rs0277,tape2=rs0838,user=gstbart
*all
*end
```

Holen einer Datei vom Band auf Platte

```
/option msg=fl
/exec sysupd
*opn savtap,fnin=savetape.rs0277,vsnin=rs0277,d=g
*sel <filename>
*sel ......
*end
/rel mstin       (wichtig!)
```

zum Anschauen eines Bandinhalts(Dateinamen)

```
/option msg=fl
/exec sysupd
*lst savtap,sysout,fnin=savetape.rs0277,vsnin=rs0277,d=g
/rel mstin      (wichtig!)
```

zum Laden des LISP- und APE-Systems

```
/do $p.lisp4.6,(save=n)
*(load 'PM-ADT)
*(load 'ADT-REGELN.INTERN)

oder

*(load 'PM-ALG)
*(load 'PMREST)
*(load 'ALG-REGELN)
```

zum Ausdrucken dieser Arbeit

```
/e dipl.print,time=299
```