SEKI-PROJEKT

SEKI MEMO

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

Eine Algebraische Spezifikation des Software-Produkts INTAKT

Hartmut Grieneisen

Memo SEKI-82-02

EINE ALGEBRAISCHE SPEZIFIKATION

DES SOFTWARE-PRODUKTS

INTAKT

von
Hartmut Grieneisen

Abstract

A worked example of a complete specification and abstract implementation of a sizable software system is given in terms of a predecessor of the software specification language ASPIK.

The specified software system INTAKT has been developed by Siemens AG (München). INTAKT is an interactive system for analyzing and upgrading programs written in a variety of programming languages. This paper only provides a specification and abstract implementation for part of the analysis support in INTAKT. The remainder is specified in a companion paper by W. Schrupp and J. Tamme. (Int. Bericht 84/83)

Zusammenfassung

Ein ausgearbeitetes Beispiel einer vollständigen Spezifikation und abstrakten Implementierung eines umfangreichen Softwaresystems wird angegeben, das in einem Vorläufer der Softwarespezifikationssprache ASPIK beschrieben ist.

Das spezifizierte Softwaresystem INTAKT wurde bei der Siemens AG (München) entwickelt. INTAKT ist ein Dialogsystem zur Analyse und Aufbereitung von Programmen verschiedener Programmiersprachen. Diese Arbeit beschreibt Spezifikation und abstrakte Implementierung eines Teils der Analyseunterstützung in INTAKT. Der restliche Teil ist spezifiziert in der Arbeit von W. Schrupp und J. Tamme. (Int. Bericht 84/83)

Gliederung

0. Einleitung

Die vorliegende Arbeit ist eingebettet in das Projekt PROGRAMMVERIFIKATION, das unter Leitung von Herrn Prof. Dr. P. Raulefs und Herrn Dr. J. Siekmann an der Universität Bonn und der Universität Karlsruhe durchgeführt wird. Das gesamte Forschungsvorhaben ist in [RaSi80] beschrieben.

An dieser Stelle sollen einige wichtige Ideen und wesentliche Aspekte der Programmverifikation wiederholt werden, um das eigentliche Thema dieser Arbeit "Eine algebraische Spezifikation eines Anwendungsprogramms", einordnen zu können.

Die folgende Darstellung ist im wesentlichem aus [Kimm79] entnommen.

Es ist allgemein bekannt, daß sich das Kostenverhältnis Hardware/Software in der Datenverarbeitung mehr und mehr auf die Seite der Software verlagert. Machten die Kosten der Hardware im Jahre 1955 noch über 80% der Gesamtkosten aus, so soll dieser Prozentsatz im Jahre 1985 auf unter 20% geschrumpft sein. Entsprechend werden die Softwarekosten auf 80% ansteigen. Der Hauptanteil dieser Softwarekosten liegt bei der Softwarewartung und nicht bei der Softwareentwicklung. Ursache ist, daß sich aufgrund der technologischen Reife der Computer die Anwendungsbereiche erweiterten. Wurden die ersten Rechner hauptsächlich in Naturwissenschaft und Technik eingesetzt, so gibt es heute wenige Gebiete, in denen Probleme ohne Rechnerunterstützung behandelt werden. Allerdings hielt die Entwicklung der Programmiertechniken nicht Schritt mit der Hardwareentwicklung. Komplexität und Umfang von Programmen nahmen erheblich zu, obwohl die Fertigkeiten der Programmierung die gleichen der Anfangsphase waren. Diese Entwicklung hatte eine Vielzahl von unkorrekten Programmen zur Folge, die ständig zu korrigieren und zu ändern waren.

Abgesehen von den Kosten gibt es aber auch Bereiche, in denen absolut zuverlässige Programme unbedingt notwendig wären. Solche Programme werden z.B. in Raumfahrt und Verkehr benötigt. Weitere Beispiele sind Kontrollprogramme von Industrieanlagen und Kernreaktoren.

Dies alles führte zur Disziplin des SOFTWARE ENGINEERING mit den Zielen, effiziente und zuverlässige Software zu erstellen.

Bei dem Teilaspekt PROGRAMMVERIFIKATION des SOFTWAREENGINEERING handelt es sich darum, den formalen Nachweis zu führen, daß ein Programm genau das leistet was der Benutzer wünscht oder anders formuliert, zu zeigen, daß Programm X eine Lösung des Problems Y darstellt. Die Wünsche des Benutzers sind in der Anforderungsdefinition festgehalten, die zwar eine Beschreibung der beabsichigten Funktion enthält, aber keine formale Darstellung. Formale Beweise können nur in mathematischen Theorien geführt werden. Um also ein Programm verifizieren zu

können, ist es notwendig, die Anforderungsdefinition an das Programm in eine formale Spezifikation zu übersetzen. Dieser Übergang ist insofern kritisch, da es nicht möglich ist, nachzuweisen, daß die Spezifikation und die Anforderungsdefinition identisch sind.

Die Spezifikation stellt eine Lösung der Anforderungsdefinition dar, wobei die folgenden Kriterien Maßstäbe der Güte einer Spezifikation sind: Formalisiertheit, Implementierungsunabhängigkeit, Änderbarkeit und Verständlichkeit.

PROGRAMMVERIFIKATION bedeutet nun den Nachweis der Richtigkeit (der Konsistenz) eines Programmms bzgl. der formalen Spezifikation der Funktion, deren Implementierung das Programm sein soll. Ein wesentlicher Aspekt im Bonner/Karlsruher Projekt ist, daß die Spezifikation Schritt für Schritt über mehrere Abstraktionsebenen hinweg in die Implementierung überführt wird, wobei dann die Richtigkeit jeder dieser Schritte nachzuweisen ist.

Die im Projekt gewählte Methode ist die algebraische, die auf GUTTAG [Gutt75], LISKOV und ZILLES [LiZi74] und ZILLES [Zill74] zurückgeht. Die Spezifikationssprache, die in der vorliegenden Spezifikation verwendet wird, ist im Rahmen des Projekts am Bonnner Institut entwickelt worden. Sowohl die algebraische Methode als auch das Beschreibungsmittel werden im 1. Kapitel dieser Arbeit näher erläutert. Sinn dieses Kapitels ist es nicht, die algebraische Spezifikationsmethode in allen Details darzustellen, sondern einen Überblick zu geben, so daß diese Arbeit auch ohne das Studium weitere Literatur lesbar ist.

In Kapitel 2 wird das Produkt INTAKT vorgestellt, das in Teilen in dieser Arbeit spezifiziert wird. Es ist ein aus der industriellen Praxis hervorgegangenes Projekt, und es wird bei der Firma SIEMENS in München entwickelt.

Kapitel 3 enthält die Spezifikation des Auswertungsteils von INTAKT. Bei dieser Spezifikation handelt es sich um eine Spezifikation der obersten abstrakten Stufe, d.h. daß alle anderen Abstraktionsstufen bis hin zur eigentlichen Implementierung nicht spezifiziert sind.

Diese vorliegende Spezifikation ist ein Beispiel einer formalen Darstellung eines aus der industriellen Praxis stammenden Anwendungsprogramms.

In dieser Spezifikation wird die Spezifikationssprache TRIPLEX, die auf der algebraischen Spezifikationsmethode basiert, benutzt. Mittels dieses Beispiels kann die Spezifikationssprache studiert und beurteilt werden.

Außerdem läßt sich an dieser Spezifikation studieren, wie eine formale Spezifikation aussieht und welchen Umfang sie annimmt. Somit kann

der Aufwand für andere, aber ähnliche Beispiele abgeschätzt werden.
Tatsächlich ist diese Spezifikation Voraussetzung für die Verifikation von INTAKT. (Denkbar wäre auch eine andere formale Spezifikation.)

1. Spezifikation von ADT

1.1. Spezifikation von ADT durch initiale Algebren

Abstrakte Datentypen haben sich als wichtiges Werkzeug für die Spezifikation von Problemen erwiesen. Was ADT sind und wie sie algebraisch spezifiziert werden, soll in diesem Kapitel erläutert werden. Ausführlichere Information und notwendige Beweise befinden sich in [Raul79], [ADJ 76] und [Kreo78].

Datenstrukturen sind dadurch gekennzeichnet, daß es Daten gibt, die zu einer Datenmenge gehören bzw. auf mehreren verteilt sind. Interessant aber werden Datenstrukturen erst durch die Operationen, die den Zugriff auf Daten gestatten bzw. die Daten verändern. Eine solche Einheit aus Datenmengen und Operationen über diesen Mengen wird in der Mathematik eine Algebra genannt. Dies berechtigt zu der Behauptung , daß Datentypen Algebren sind.
Der syntaktische Teil eines Datentyps wird durch eine Signatur beschrieben. Die Signatur legt Namen für Datenmengen und Operationen, sowie Argument- und Wertebereich von Operationen fest.

Definition
Eine Signatur (S,Σ) besteht aus einer Menge S von Sorten und einer Familie $\Sigma=\{\Sigma_{w,s} \mid w\in S^*, s\in S\}$ von Mengen. □

Inhaltlich wird eine Signatur durch die Algebren festgelegt, die der Signatur genügen, dadurch daß sie entsprechend zu Sorten- und Operationssymbolen der Signatur, Datenmengen und Operationen besitzen.

Definition
Für eine Signatur (S,Σ) besteht eine (S,Σ)-Algebra A aus:
(1) $\forall\, s\in S$ einer Menge A_s, die Trägermenge von S heißt.
(2) $\forall\, w\in S^*,\ s\in S.\ \forall\, f\in\Sigma_{w,s}$ einer Funktion
$f_A: A_{s_1}\times\ldots\times A_{s_n}\longrightarrow A_s$ für $w=s_1\ldots s_n, w\neq\varepsilon$.
(3) $\forall\, s\in S.\ \forall\, f\in\Sigma_{\varepsilon,s}$ einem ausgezeichnetem Element $f_A\in A_s$.
Bemerkung: Für (S,Σ)-Algebra steht manchmal auch einfach Σ-Algebra. □

Man kann sich die Sorten- und Operationssymbole der Signatur als formale Parameter denken, die durch die Datenmengen und Operationen der Algebra aktualisiert werden.
Um einen ADT definieren zu können, ist es noch notwendig die strukturverträglichen Abbildungen zwischen zwei Σ-Algebren zu definieren.

Definition
A und B seien (S,Σ)-Algebren. Dann heißt eine Familie $H:=\{h_s: A_s\longrightarrow B_s\}$ von Abbildungen ein (S,Σ)-Homomorphismus, wenn gilt:

(1) $\forall s \in S.\ \forall f \in \Sigma_{\varepsilon,s}.\ h_s(f_A) = h_s(f_B)$.
(2) $\forall w = s_1 \ldots s_n \in S^*, s \in S.\ \forall a_i \in A_{s_i}\ (1 \le i \le n)$.
$h_s(f_A(a_1,\ldots,a_n)) = f_B(h_{s_1}(a_1),\ldots h_{s_n}(a_n))$.
Bezeichnung: H:A-->B. □

Eine besondere Bedeutung kommt der nächsten Definition zu.

Definition
Eine (S,Σ)-Algebra A heißt initial , wenn für jede (S,Σ)-Algebra B genau ein (S,Σ)-Homomorphismus H:A-->B existiert. □

Definition
Ein (S,Σ)-Homomorphismus H: A-->B heißt ein (S,Σ)-Isomorphismus , wenn $\forall s \in S.\ h_s$ eine Bijektion ist. Gibt es einen (S,Σ)-Isomorhpismus zwischen A und B, dann heißen A und B isomorph . □

Tatsächlich gilt, daß initiale (S,Σ)-Algebren isomorph sind, d.h. daß sie durch Umbenennung auseinander hervorgehen und daß sich jede andere (S,Σ)-Algebra durch einen (S,Σ)-Homomorphismus erreichen läßt.

Mit diesen bisher definierten Begriffen, läßt sich nun auch charakterisieren was 'abstrakt' in ADT eigentlich bedeutet. Abstrakt bedeutet ja unabhängig von der Darstellung und das bedeutet eindeutig bis auf isomorphe Strukturen. Es ist in der Mathematik üblich isomorphe Strukturen als identisch zu behandeln. Das berechtigt zur folgenden Definition [ADJ 76]:

Definition
Ein ADT ist die Isomorphieklasse einer initialen (S,Σ)-Algebra.
Bemerkung: Tatsächlich gibt es auch andere Ansätze zur Definition von ADT, z.B. mittels terminalen Algebren ([Wan78],[HoRa79]). Auf diese Ansätze wird hier aber nicht eingegangen. □

Man ist natürlich auch daran interessiert, ob eine solche initiale Algebra immer existiert und wie sie ausschaut.

Definition
Die Menge der (S,Σ)-Terme einer Signatur (S,Σ) ist definiert durch
(1) $\forall s \in S.\ \forall \Sigma_{\varepsilon,s} \in \Sigma.\ \Sigma_{\varepsilon,s} \subseteq T_{\Sigma,s}$.
(2) $\forall s_1 \ldots s_n \in S^*, s \in S.\ \forall f \in \Sigma_{s_1 \ldots s_n, s}.\ \forall t_i \in T_{\Sigma,s_i}\ (1 \le i \le n)$.
$f(t_1,\ldots,t_n) \subseteq T_{\Sigma,s}$.
$T_{\Sigma,s}$ bildet die Menge aller Terme der Sorte s. □

Definition
Die Termalgebra T_Σ einer Signatur (S,Σ) ist definiert durch:
(1) $\forall s \in S.\ T_{\Sigma,s}$ ist Trägermenge der Sorte s.
(2) $\forall s \in S.\ \forall f \in \Sigma_{\varepsilon,s}.\ f_{T_\Sigma} := f$.
(3) $\forall\ w = s_1 \ldots s_n \in S^*, s \in S.\ \forall\ f \in \Sigma_{w,s}.\ f_{T_\Sigma}: T_{\Sigma,s_1} \times \ldots \times T_{\Sigma,s_n} \longrightarrow T_{\Sigma,s}$ ist die

durch $\forall\ t_i \in T_{\Sigma,s_i}.\ f_{T_\Sigma}(t_1,\ldots t_n) := f(t_1,\ldots,t_n)$ definierte Funktion. □

Die Beantwortung der Frage nach der Existenz und des Aussehens einer initialen (S,Σ)-Algebra erfolgt durch das folgende wichtige Theorem.

Theorem
Für jede Signatur (S,Σ) ist die Termalgebra T_Σ eine initiale Algebra. □

Tatsächlich erfüllt die Termalgebra noch eine weitere Eigenschaft bzw. Anforderung, die charakteristisch für einen ADT ist. Die Termalgebra ist operationserzeugt, d.h. daß alle Daten ausschließlich durch die in Σ enthaltenen Operationen erzeugt werden. Dies wird durch die Initialität der Termalgebren garantiert.

Zusammenfassend gilt: Jede Signatur (S,Σ) spezifiziert einen ADT, der durch die Isomorphieklasse der initialen (S,Σ)-Algebren definiert ist.

Das bisher dargestellte Konzept zur Spezifikation von Algebren durch Signaturen und somit von ADT reicht nicht aus, wenn der Datentyp noch mit weiteren Operationen ausgestattet werden soll. Ursache ist, daß es nicht möglich ist, Eigenschaften von Operationen auszudrücken und zu fordern, die in den Vergleichsalgebren gelten. Was man haben möchte sind initiale Algebren, die bestimmten Beschränkungen bzw. Axiomen genügen. Mittel um diese Beschränkungen auszudrücken sind prädikatenlogische Formeln, Ungleichungen und Gleichungen. In diesem hier beschriebenen Ansatz werden Gleichungen verwandt.

Auf der syntaktischen Ebene ist es also notwendig, das Konzept der Signaturen durch Gleichungen zu ergänzen und auf der semantischen Ebene muß präzisiert werden, wann eine Algebra, die durch die Gleichung festgelegten Eigenschaften besitzt, wann sie die Gleichung erfüllt. Ein Beispiel einer solchen Gleichung ist: not(not(x))=x.

Als nächstes wird geklärt, was man unter einer Variablen versteht und was eine Termalgebra mit Variablen ist.

Definition
(S,Σ) sei eine Signatur.
(1) $\forall s \in S.$ X_s sei eine unendliche, abzählbare Menge von Variablensymbolen der Sorte s, so daß:
1. $\forall s' \in S. s \neq s' \Rightarrow X_s \cap X_{s'} = \{\}$.
2. $X_s \cap \Sigma = \{\}$.
(2) $(S,\Sigma(x))$ bildet eine Signatur mit Variablen mit:
$\Sigma(x) := \{(\Sigma(x)_{w,s} \mid w \in S^*, s \in S\}$ mit:

1. $\forall s \in S.\ \Sigma(x)_{\varepsilon,s} := \Sigma_{\varepsilon,s} \cup X_s$.
2. $\forall w \in S^*, s \in S.\ \Sigma(x)_{w,s} := \Sigma_{w,s}$.

(3) Die $\Sigma(x)$- Termalgebra $T_{\Sigma(x)}$ bezeichnet man als Σ-Algebra $T_\Sigma(x)$ oder als freie Algebra über X:
1. $\forall s \in S.\ T_\Sigma(x)_s := T_{\Sigma(x),s}$.
2. $\forall w \in S^*, s \in S.\ \forall f \in \Sigma_{w,s}.\ fT_{\Sigma(x)} := fT_{\Sigma(x)}$.

(4) Für eine Σ-Algebra A heißt die Familie von Abbildungen $B: X \to A$, $B = \{b_s \mid \forall s \in S.\ b_s: X_s \to A_s\}$ Belegung der Variablen aus X in A. □

Terme mit Variablen lassen sich in Σ-Algebren interpretieren, indem die Operationssymbole durch die aktuellen Operationen interpretiert und die Variablen mit Werten belegt werden. Es gilt, daß durch die Belegung ein eindeutiger Σ-Homomorphismus H* definiert wird.

Theorem
Sei $(S,\Sigma(x))$ eine Signatur mit Variablen.
Zu jeder Σ-Algebra A und jeder Belegung $B: X \to A$ existiert genau ein (S,Σ)-Homomorhismus $H^*: T_\Sigma(x) \to A$ mit $h_s^*(x) = b_s(x)$ für alle $x \in X_s, s \in S$. □

Für diese freien Σ-Algebren gilt analog zu initialen, daß sie bis auf Isomorphie eindeutig bestimmt sind.
Aufbauend auf Termen mit Variablen lassen sich nun Gleichungen und Spezifikationen mit Gleichungen definieren.

Definition
Sei $(S,\Sigma(x))$ eine Signatur mit Variablen.
(1) Für jede Sorte $s \in S$ heißt ein geordnetes Paar $e = (L,R)$ von Σ-Termen L und R der Sorte s mit Variablen in X Gleichung der Sorte s.
(2) Sei $\mathcal{E} = \{E_s \mid s \in S\}$ eine Mengenfamilie von Gleichungen der Sorte s. Dann heißt $\langle S,\Sigma,\mathcal{E}\rangle$ eine Spezifikation. □

Die Gleichungen einer Spezifikation $SPEC = \langle S,\Sigma,\mathcal{E}\rangle$ sind Anforderungen an Σ-Algebren. Sei B eine Belegung der Variablen und H* der eindeutige (S,Σ)-Homomorphismus, der B auf alle Σ-Terme fortsetzt. Dann greifen die Terme $h_s^*(L)$ und $h_s^*(R)$ auf Daten der Σ-Algebren zu. Sind sie gleich, so erfüllt die Σ-Algebra die Gleichung.

Definition
Sei $SPEC = \langle S,\Sigma,\mathcal{E}\rangle$ eine Spezifikation und $e = (L,R) \in \mathcal{E}$ eine Gleichung der Sorte s und H* der eindeutige Homomorphismus, der B fortsetzt.
Eine (S,Σ)-Algebra A erfüllt die Gleichung $e = (L,R)$, wenn für alle Belegungen $B: X \to A$ gilt: $H^*(L) = H^*(R)$.
A ist eine $(\Sigma,\mathcal{E})$-Algebra, wenn A alle Gleichungen aus $\mathcal{E}$ erfüllt. □

Folgende Definition ist nach allem bisher Ausgeführtem sinnvoll [ADJ 76].

Definition

Ein ADT ist die Isomorphieklasse der initialen (S,Σ)-Algebren, die $\mathcal{E}$ erfüllen. □

Im allgemeinen erfüllen die Termalgebren die Gleichungen aus $\mathcal{E}$ nicht, da Terme nur dann gleich sind, wenn sie syntaktisch gleich sind. Es stellt sich also wiederum die Frage nach der Existenz einer initialen $(\Sigma,\mathcal{E})$-Algebra. Eine solche Algebra läßt sich folgendermaßen konstruieren.
Die Gleichungen aus $\mathcal{E}$ induzieren für jede Sorte s eine Äquivalenzrelation auf den Termen der Sorte s. Alle zu einem Term t äquivalenten Terme werden in einer Klasse zusammmengefaßt und damit identifiziert. Die Mengen aller Äquivalenzklassen aller Sorten bilden die Datenmenge der Algebra. Die Operationen der Algebra werden über die Äquivalenzklassen definiert, wobei die Operationen verträglich mit der Äquivalenzrelation sein müssen.

Definition
Sei $\langle S,\Sigma,\mathcal{E}\rangle$ eine Spezifikation.
(1) Die Gleichungen $\mathcal{E}$ induzieren auf den Termen aus T_Σ eine Familie von Relationen $\equiv_{\mathcal{E}} = \{\equiv_s \mid s \in S\}$ die definiert ist durch:
1. $H^*(L) \equiv_s H^*(R)$ für alle $(L,R) \in \mathcal{E}$ und $B: X \to T_\Sigma$.
2. $\forall s \in S\ \equiv_s$ ist Äquivalenzrelation.
3. $\forall w = s_1 \ldots s_n \in S^*, s \in S.\ \forall f \in \Sigma_{w,s}.\ \forall t_i, t_i' \in T_{\Sigma,s_i}\ (1 \le i \le n)$.
$t_i \equiv_{s_i} t_i' \Rightarrow f(t_1,\ldots,t_n) \equiv_s f(t_1',\ldots,t_n')$.

Anmerkung: Die so definierte Relationenfamilie wird als Kongruenz bezeichnet. $\equiv_{\mathcal{E}}$ ist die durch $\mathcal{E}$ erzeugte, kleinste Kongruenzrelation, die die durch Eigenschaft (1) bestimmte Relation enthält.
(2) Für jeden Term $T_{\Sigma,s}$ werden in der Äquivalenzklasse $[t]$ alle zu t äquivalenten Terme zusammengefaßt: $[t] := \{t' \mid t \equiv_s t'\}$
(3) Die Mengen aller Äquivalenzklassen
$(T_\Sigma/\equiv_{\mathcal{E}})_s = \{[t] \mid t \in T_{\Sigma,s}\}$ für alle $s \in S$
bilden die Datenmengen der Quotiententermalgebra $T_{\Sigma,\mathcal{E}}$, deren Operationen definiert sind durch:
$\forall w = s_1 \ldots s_n \in S, s \in S.\ \forall f \in \Sigma_{w,s}.\ \forall [t_i] \in (T_\Sigma/\equiv_{\mathcal{E}})_{s_i}\ (1 \le i \le n)$.
$f_\Sigma/\equiv_{\mathcal{E}}([t_1],\ldots[t_n]) := [f(t_1,\ldots,t_n)]$ □

Es gilt das Theorem:

Thorem
$T_{\Sigma,\mathcal{E}}$ ist eine initiale $(\Sigma,\mathcal{E})$-Algebra. □

Zusammenfassend gilt somit: Jede Spezifikation $SPEC = \langle S,\Sigma,\mathcal{E}\rangle$ spezifiziert einen ADT, der durch die Isomorphieklasse der initialen $(\Sigma,\mathcal{E})$-Algebren definiert ist.

1.2. Der CTA-Ansatz zur Spezifikation von ADT

Die algebraische Spezifikation durch $\langle S,\Sigma,E\rangle$ ist von hohem abstrakten Niveau [Hess81]. Für die Spezifikation von Problemen bzw. deren Lösungen ist es manchmal natürlicher, konkrete Modelle zu definieren, anstatt die Spezifikation durch ein Axiomensystem festzulegen. Eine Spezifikation in diesem Sinne besteht aus einer konkreten Algebra, in der die Operationen explizit definiert sind. Man spricht dann auch von algorithmischer Spezifikation.

Der ADT kann dann wie folgt definiert werden:
Bemerkung zur Notation: Im folgenden werden bestimmte Operationsnamen mit einem * als Prefix versehen. Immer dann, wenn mit einem Term ein Element der Trägermenge bzw. des Herbrand-Universums gemeint ist, erhält der Operationsname einen * als Prefix. Eine Operation $op(t_1,\ldots,t_n)$ (ohne *) kennzeichnet das Element, das man erhält, wenn Operation op auf die Argumente t_1 bis t_n angewandt wird. $*op(t_1,\ldots,t_n)$ kennnzeichnet hingegen ein Element aus dem Herbrand-Universum.

Definition
Eine Σ-Algebra C ist eine kanonische Termalgebra CTA wenn:
(1) $C_s \subseteq T_{\Sigma,s}$ für jedes $s \in S$, d.h. die Trägermenge ist Teil des Herbrand-Universums.
(2) $*op(t_1,\ldots t_n) \in C \Rightarrow t_i \in C$.
(3) $*op(t_1,\ldots,t_n) \in C \Rightarrow op(t_1,\ldots,t_n) = *op(t_1,\ldots,t_n)$. □

Theorem
Sei $\langle S,\Sigma,E\rangle$ eine Spezifikation.
Dann gibt es eine initiale (Σ,E)-Algebra C, die eine CTA ist. □

Der ADT, der durch eine CTA spezifiziert wird, ist definiert als die Isomorphieklasse der CTA.

Eine solche Spezifikation durch eine konkrete Algebra ist etwas weniger abstrakt als die algebraische Spezifikation. Beide Ebenen sind allerdings sinnvoll, und in der Spezifikationssprache TRIPLEX [BG 82a] hat man versucht, einen Rahmen zu liefern, der beide Abatraktionsebenen umfaßt.

1.3. Spezifikationen in TRIPLEX

Die Spezifikationssprache TRIPLEX [BG 82b] liefert einen syntaktischen Rahmen zur Definition einer CTA. Grundsätzlich werden drei verschiedene Arten von Spezifikationen unterschieden. Es handelt sich dabei um Einfache Spezifikationen, Spezifikationen mit Parametern und Spezifikationen von Parametern. Alle diese Spezifikationen folgen einem identischen Schema und werden durch die Schlüsselwörter SSPEC, PSPEC und PARM unterschieden.

Das Schema ist in Kopf und Rumpf geteilt.

1.3.1. Das Spezifikationsschema

Die Beschreibung des Schemas basiert auf den SSPEC.

1. Der Spezifikationskopf

Der Kopf der Spezifikation ist die Schnittstelle zum Benutzer. Hier sind alle Informationen enthalten, die von der Spezifikation nach außen angeboten werden.

Im einzelnen besteht der Kopf aus folgenden Teilen:

1.1. Use-Clause

In diesem Abschnitt werden alle Spezifikationen aufgelistet, die von der neuen Spezifikation benutzt werden, d.h. daß alle Operationen dieser Spezifikationen, der neuen Spezifikation zur Verfügung stehen. Tatsächlich besteht noch die Möglichkeit diese Operationen auf einige spezielle durch einen restrict-Abschnitt zu beschränken. Zu beachten ist, daß in den Use-Abschnitten sämtlicher Spezifikationen keine Zyklen auftreten, da dadurch die hierarchische Struktur der Spezifikationen verletzt würde. Es wird vorausgesetzt, daß alle Spezifikationen die Spezifikation Bool benutzen. Die Spezifikation Bool ist hingegen die einzige Spezifikation, die keine andere benutzt, also eine leere Use-Clause besitzt.
Die Liste der benutzten Spezifikationen wird getrennt nach den drei Arten SSPEC, PSPEC und PARM aufgeführt. Dabei kann eine SSPEC keine PSPEC oder PARM benutzen.

1.2. Public-Clause

Dieser Abschnitt enthält die Signatur der CTA. Insbesondere sind dies Namen für neue Sorten und Operationsnamen mit Angabe der Funk-

tionalität.
Die hier aufgeführten Operationen können von allen Spezifikationen benutzt werden, die in ihrem Use-Abschnitt den Namen dieser Spezifikation enthalten.
Die benutzten Sorten müssen bekannt sein, d.h. entweder sind sie neu oder in einer benutzten Spezifikation definiert.

1.3. Properties

Dieser Abschnitt enthält eine Beschreibung von Eigenschaften der Operationen, die oberhalb der algoritmischen Definition liegt. Aus den Properties einer Funktion sollte die informelle Beschreibung der Funktion ablesbar sein. Beschreibungsmittel für Properties sind im wesentlichen Gleichungen mit Variablen und Prädikatenlogik erster Stufe. Dies entspricht der Idee der algebraischen Spezifikation. Informationen über die Spezifikation als ganzes kann hier ebenfalls abgelegt werden. Somit dienen die Properties der Dokumentation der Spezifikation.

2. Der Spezifikationsrumpf

Der Spezifikationsrumpf enthält die Definition der Trägermenge und der Operationen der CTA.

2.1. Definition der Trägermenge

Die Trägermenge einer Sorte s ist eine Teilmenge des Herbrand-Universums der Sorte s.
Im einfachen Fall sind Trägermenge und Herbrand-Universum identisch. Aufgespannt wird das Herbrand-Universum durch die Konstruktoren. Die Konstruktoren sind mit einem Stern (*) als Prefix versehene Operationsnamen, die auch in der Public-Clause vorkommen müssen. Wann immer eine Operation mit Prefix * vorkommt, ist ein Element aus dem Herbrand-Universum gemeint.
Im anderen Fall ist die Trägermenge C_s echte Teilmenge des Herbrand-Universums$T_{\Sigma,s}$. Dann kann es sinnvoll sein, Hilfsfunktionen über dem Herbrand-Universum zu definieren, mit denen die Akzeptor-Funktion, eigentlich ein Prädikat (is.s) definiert wird, die die Elemente der Trägermenge charakterisiert.
$C_s=\{t \mid t \in T_{\Sigma,s} \wedge is.s\ (t)=true\}$
Sowohl Hilfsfunktionen als auch Akzeptorfunktionen fehlen im ersten Fall. Abweichend von der Definition der CTA enthält jede Trägermenge noch ein Error-Element error.s.

2.2. Definition der CTA-Operationen

Dieser Abschnitt besteht im allgemeinen aus drei Teilen.

Unter dem Schlüsselwort "define constructors" werden die Konstruktoren als öffentliche Operationen definiert. In den Fällen, in denen die Trägermenge gleich dem Herbrand-Universum ist, ist op(x) automatisch durch *op(x) definiert. Ist die Trägermenge echte Teilmenge des Herbrand-Universums, so ist die öffentliche Operation entsprechend Forderung (3) der Definition der CTA, $op(t_1,\ldots,t_n)= *op(t_1,\ldots,t_n)$, wenn $*op(t_1,\ldots,t_n)$ Element der Trägermenge, mittels des weiter unten erklärten Schemas zur Definition von Operationen zu definieren.

Unter dem Schlüsselwort "private operations" ist die Funktionalität der Operationen notiert, die intern benutzt werden, aber außerhalb der Spezifikation nicht zur Verfügung stehen sollen.

Der Teil mit dem Schlüsselwort "define operations" enthält schließlich die Definition aller öffentlicher Operationen und aller privater Hilfsfunktionen. Zur Definition von Operationen steht folgendes Schema zur Verfügung:
Die linke Seite der Operationsdefinition besteht aus einem Operationsaufruf mit Variablen der Sorten, entsprechend der Funktionalität der Operation.
Die rechte Seite besteht aus einem Operationsausdruck. Ein Operationsausdruck kann sein:
- eine Operationsaufruffolge mit bekannten Operationen und Variablen, die auch auf der linken Seite der Operationsdefinition benutzt werden
- ein if-then-else expression, wobei der then bzw. der else Teil wiederum ein Operationsausdruck sein kann.
- ein case expression, in dem der strukturelle Aufbau, der durch die Konstruktoren definierten Trägermenge ausgenutzt wird. Abhängig vom äußersten Konstruktor wird als Ergebnis ein entsprechender Operationsausdruck angegeben.
- ein let expresssion, das praktisch nur eine abkürzende Schreibweise darstellt. Durch let x=t in y wird festgelegt, daß die Variable x im Operationsausdruck y stellvertretend für den Operationsausdruck t steht.
- eine 0-stellige Funktion oder
- die Konstante error.s, die das Fehlerelement der Sorte s, ebenfalls mit error.s bezeichnet, liefert.

Für alle CTA und alle Sorten s wird automatisch eine Operation
eq.s: s s --> bool generiert, die true liefert, falls die angegebenen Elemente der Trägermenge C_s syntaktisch gleich sind.

1.3.2. Parametrisierung

Wenn man von einem Datentyp Keller spricht, so meint man den Keller und seine Operationen unabhängig vom Inhalt des Kellers. Im konkreten Fall ist der Inhalt allerdings wesentlich.

Auf dieser Idee basieren die parametrisierten Spezifikationen, die Parameter-Spezifikationen und das Konzept der Instanzbildung.

1. Parametrisierte Spezifikationen

Die parametrisierten Spezifikationen sind durch das Schlüsselwort "PSPEC" gekennzeichnet. Der Spezifikationsname wird durch eine Liste ergänzt, aus der die Parameter einschließlich der zugehörigen Parameter-Spezifikation hervorgehen. Die Parametersorte wird in der Spezifikation wie jede andere Sorte auch behandelt. Für die Parameter gilt, daß sie die in der Parameter-Spezifikation geforderten Eigenschaften erfüllen müssen.

2. Parameter-Spezifikationen

Parameter-Spezifikationen sind durch das Schlüsselwort "PARM" gekennzeichnet. Eine Parameter Spezifikation besteht nur aus dem Kopf des Spezifikationsschemas.
- Jede Parameter-Spezifikation benutzt die SSPEC Bool.
- Im Abschnitt Public-Clause können Sorten- und Operationsnamen für Parameter eingeführt werden.
- Im Abschnitt Properties werden die Anforderungen, die die Operationen erfüllen müssen formuliert.

Im Prinzip wird durch eine Parameter-Spezifikation eine Schnittstelle definiert, in der festgelegt wird, welche Eigenschaften die aktuellen Parameter erfüllen müssen.

3. Instanzbildung

Das Bilden einer Instanz bedeutet, daß zumindest ein Parameter einer parametrisierten Spezifikation durch einen aktuellen Parameter ersetzt wird. Ein aktueller Parameter kann eine SSPEC, eine PSPEC oder ein PARM sein.
Das Schlüsselwort "Instantiate" kennzeichnet die Instanzbildung. Dabei wird folgende Information benötigt:
- Der Name der PSPEC und der neue Name der aktualisierten Spezifikation,
- Die Liste der formalen Parameter und korrespondieren dazu die Liste der aktuellen Parameter.

Zusätzlich besteht die Möglichkeit, Sorten und Operationen umzubenennen.

Tatsächlich sollen Spezifikationen in Kommunikation mit dem Rechner geschrieben werden. Dann ist die Instanzbildung als Kommando zum Erzeugen einer SSPEC bzw. PSPEC zu verstehen.

Bei der Instanzbildung ist es möglich, daß eine Instanzbildung eine weitere nach sich zieht. In diesen Fällen ist beabsichtigt, daß der Rechner dem Benutzer ein Schema liefert, das dieser an wenigen, vorbereiteten Stellen mit entsprechenden Eingaben auszufüllen hat.

Diese abgeleitete Instanzbildung wird beim ersten Auftreten in der Spezifikation (INSTANTIATE Baum to Symboltabelle_1) erwähnt und erläutert und bei allen weiteren Fällen als gegeben und in analoger Weise durchgeführt vorausgesetzt.

2. Beschreibung von INTAKT

INTAKT ist der Name eines Softwareprodukts, das bei der Firma SIEMENS, München entwickelt wird. INTAKT stellt ein Analyseinstrument dar, das eingesetzt werden soll, um zu Aussagen über die Qualität eines gegebenen Programmsystems zu gelangen.

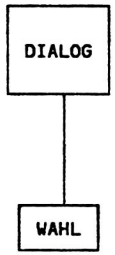
Unter einem Programmsystem versteht man dabei eine Menge von Übersetzungseinheiten, die in der Praxis in der Regel in verschiedenen Quellsprachen geschrieben sind. In der ersten Version von INTAKT wird allerdings davon ausgegangen, daß alle Quellmodule des Programmsystems in SPL 3 geschrieben sind.

Zusammenfassend läßt sich sagen, daß INTAKT rein statische Aussagen über ein Programmsystem liefert. INTAKT dient nicht dazu, die syntaktische Korrektheit von Quellmodulen zu überprüfen, vielmehr wird vorausgesetzt, daß alle Quellmodule syntaktisch richtig sind. Die Ergebnisse, die INTAKT liefern soll, sind im einzelnen in der Beschreibung der jeweiligen Auswertefunktionen enthalten.

2.1. Der SIEMENS Entwurf

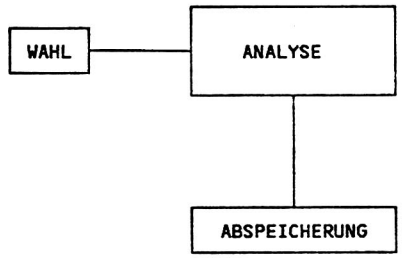
INTAKT besteht hauptsächlich aus den drei Teilen:
DIALOG, ANALYSE und AUSWERTUNG.

Im DIALOG werden die Benutzereingaben ermittelt.

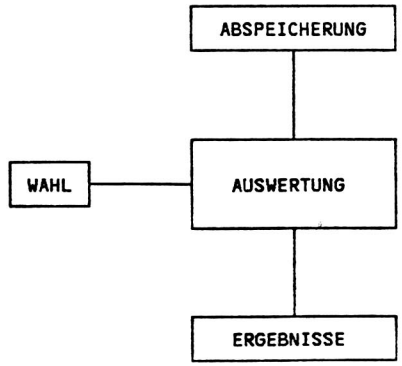


In WAHL werden die Benutzereingaben festgehalten und ANALYSE und AUSWERTUNG bekanntgegeben.

In ANALYSE werden die im DIALOG angegebenen Quellprogramme syntaktisch analysiert. Das Ergebnis dieser Syntaxanalyse sind die Strukturbäume und die Symboltabellen. Diese werden in der ABSPEICHERUNG abgelegt.



Die AUSWERTUNG verarbeitet die Resultate der ANALYSE, d.h. die Strukturbäume und die Symboltabellen und gibt die Ergebnisse aus.
In WAHL erfährt die AUSWERTUNG welche Verarbeitung sich der Benutzer gewünscht hat.



2.2. Grundlagen für die AUSWERTUNG

Ausgangspunkt für INTAKT und auch Ausgangspunkt für die AUSWERTUNG ist ein Programmsytem. Ein solches Programmsystem besteht aus mehreren Quellmodulen, die auf verschiedenen Quelldateien verteilt sind. Für die AUSWERTUNG wird vorausgesetzt, daß es sich um Quellmodule der Programmiersprache SPL 3 handelt.

Die zentralen Datenstrukturen der AUSWERTUNG sind die Symboltabelle (ST) und der Strukturbaum (SB). Diese Datenstrukturen sind ausführlich in den entsprechenden Spezifikationen beschrieben.

Zu jedem Quellmodul muß also erst die ST bzw. der SB erzeugt werden, damit die Auswertefunktionen darauf zugreifen können. Diese Überführung in die auswertbare Form geschieht in zwei Schritten.

Nachdem das durch den Quellmodulnamen gekennzeichnete Quellmodul von der Quelldatei eingelesen wurde, wird es aufbereitet. Die AUFBEREITUNG, die ebenfalls im Rahmen des Projekts PROGRAMMVERIFIKATION spezifiziert wurde, entspricht im wesentlichem einer lexikalischen Aufbereitung des Quellmoduls. Stichworte der AUFBEREITUNG sind: Entfernen von Compile-Anweisungen, Auflösung von Includes (Copies, Makros), Entfernen von Kommentaren und redundanten Blanks.

Dieser bereinigte Quelltext ist Eingabe der ANALYSE, in der die syntaktische Analyse (einschließlich der lexikalischen Analyse) der Source erfolgt und in der die ST und der SB erzeugt wird.

Dieser Analysebaustein wird nicht spezifiziert, und es wird von dieser Transformation eines Quellmoduls abstrahiert, indem die Existenz der Operationen vorausgesetzt wird, die die ST bzw. den SB des Quellmoduls liefern. Diese Operationen stehen der AUSWERTUNG zur Verfügung. All diese Operationen und Datenstrukturen sind in der Spezifikation SCHNITTSTELLE zusammengefaßt.

3. Die Spezifikation

3.1. Allgemein benötigte Spezifikationen

3.1.1. Basisspezifikationen

Die folgenden Spezifikationen werden fast von allen anderen Spezifikationen benötigt, und sie werden deshalb als Basisspezifikationen bezeichnet.
Aufgelistet nach ihrer Art sind dies:

SSPEC: Bool	PARM: Elem	PSPEC:	Liste(#x:Elem)
Nat			Tupel(#x_1,#x_2:Elem)
Char			Baum(#x:Elem)

INSTANTIATIONS: Charstring (Zeichenkette), als Aktualisierung von Liste mit Parameter char
Name, ebenfalls als Aktualisierung von Liste mit Parameter char

SSPEC BOOL

PUBLIC SORTS : bool

PUBLIC OPS :
true:-->bool
false:-->bool
not:bool-->bool
and,or:bool bool-->bool

PROPERTIES :
∀ bool b: not (not (b))=b
∧ ((b=true ∧ not (b)=false)
∨ (b=false ∧ not (b)=true))

∀ bool a,b,c:
and (a,b)=and(b,a)
∧ (and (a,and (b,c))=and (and (a,b),c))
∧ ((a=false ∧ and (a,b)=false)
∨ (a=true ∧ and (a,b)=b))

∀ bool a,b,c:
or (a,b)=or (b,a)
∧ or (a,or (b,c))=or (or (a,b),c)
∧ ((a=false ∧ or (a,b)=b)

```
     v (a=true ∧ or (a,b)=true))

V bool a,b:
  not (and (a,b))=or (not (a),not (b))

CONSTRUCTORS :
*true
*false

DEFINE OPS :
not (x):= case x is
          *true-->false
          *false-->true
          esac ;

and (X,Y):= case x is
            *true-->y
            *false-->false
            esac ;

or (x,y):= case x is
           *true-->true
           *false-->y
           esac ;

/*
Bemerkung: In den folgenden Spezifikationen wird nicht die funktionelle  Schreibweise der aussagenlogischen Operationen verwandt,  sondern,
um die Lesbarkeit zu erhöhen, die klassische Infix_Notation.
*/

ENDSPEC


SSPEC NAT

USE SSPEC : Bool

PUBLIC SORTS : nat

PUBLIC OPS :
0:-->nat
succ:nat-->nat
add,sub:nat nat-->nat
mult,div:nat nat-->nat
incr:nat-->nat
ist_gt:nat nat-->bool
```



```
PROPERTIES :
/*
add (n1,n2) liefert die Summe von n1 und n2.
*/
V nat n1,n2,n3:
   add (n1,n2)=add (n2,n1)
   ∧ add (n1,add (n2,n3))=add (add (n1,n2),n3)
   ∧ ((n1=0 ∧ add (n1,n2)=n2)
      v (∃ nat n: n1=succ (n)
                  ∧ add (n1,n2)=succ (add (n,n2))))

/*
ist_gt (n1,n2) liefert true, falls n1>n2.
*/
V nat n1,n2:
   (n1=0 ∧ ist_gt (n1,n2)=false)
   v ∃ nat n: n1=succ (n)
              ∧ ((n2=0 ∧ ist_gt (n1,n2)=true)
                 v ∃ nat m: n2=succ (m)
                     ∧ ist_gt (n1,n2)=ist_gt (n,m))

/*
sub (n1,n2) liefert die Differenz von n1 und n2,  falls n1≥n2,  anson-
sten error.
*/
V nat n1,n2:
   (∃ nat n,m: n1=succ (n) ∧ n2=succ (m)
               ∧ sub (n1,n2)=sub (n,m))
   v (n1=0 ∧ n2=0 ∧ sub (n1,n2)=0)
   v (ist_gt (n1,n2)=true ∧ n2=0 ∧ sub (n1,n2)=n1)
   v (ist_gt (n1,n2)=false ∧ sub (n1,n2)=error)

/*
mult (n1,n2) liefert das Produkt von n1 und n2.
*/
V nat n1,n2,n3:
   mult (n1,n2)=mult (n2,n1)
   ∧ mult (n1,mult (n2,n3))=mult (mult (n1,n2),n3)
   ∧ ((n1=0 ∧ mult (n1,n2)=0)
      v (n2=0 ∧ mult (n1,n2)=0)
      v (∃ nat n: n2=succ (n)
           ∧ mult (n1,n2)=add (n1,mult (n1,n))))

/*
div (n1,n2) liefert den ganzzahligen Quotienten von n1 und n2.
*/
V nat n1,n2:    (n2=0 ∧ div (n1,n2)=error)
    v (n1=0 ∧ n2≠0 ∧ div (n1,n2)=0)
```

```
    v (ist_gt (n1,n2)=false ∧ div (n1,n2)=0)
    v (mult (n1,n2)=add (succ (0),div (sub (n1,n2),n2)))

/*
incr (n) inkrementiert n um 1.
*/
V nat n: incr (n)=add (succ (0),n)

CONSTRUCTORS :
*0
*succ

DEFINE OPS :
ist_gt (x,y):= case x is
               *0-->false
               *succ (x1)--> case y is
                             *0-->true
                             *succ (y1)-->ist_gt (x1,y1)
                             esac
               esac ;

add (x,y):= case x is
            *0-->y
            *succ (x1)-->succ (add (x1),y)
            esac ;

sub (x,y):= case x is
            *0--> case y is
                  *0-->0
                  *succ (y1)-->error.nat
                  esac
            *succ (x1)--> case y is
                          *0-->succ (x1)
                          *succ (y1)-->sub (x1,y1)
                          esac
            esac ;

mult (x,y):= case x is
             *0-->0
             *succ (x1)--> case y is
                           *0-->0
                           *succ (y1)-->add (x,mult (x,y1))
                           esac
             esac ;

div (x,y):=
  if ist_gt (x,y)
    then case y is
```



```
         *0-->error.nat
         *succ (y1)--> case x1 is
                       *0-->0
                       *succ (x2)-->add (succ (0),div (sub (x,y),y))
                       esac
         esac
    else
  fi ;

incr (x):=add (succ (0),x);

/*
Bemerkung: In den folgenden Spezifikationen wird vereinfachend für
succ (...(0)...) die natürliche Zahl, die mit diesem Term gemeint ist,
geschrieben.
*/

ENDSPEC


SSPEC CHAR

USE SSPECS : Bool

PUBLIC SORTS : char

PUBLIC OPS :
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z:-->char
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z:-->char
1,2,3,4,5,6,7,8,9,0:-->char
!,",%,&,/,(,),=,?,*,_,:,;,>,',+,#,-,.,,,<:-->char

CONSTRUCTORS :
*a,*b,*c,*d,*e,*f,*g,*h,*i,*j,*k,*l,*m,*n,*o,*p,*q,
*r,*s,*t,*u,*v,*w,*x,*y,*z,*A,*B,*C,*D,*E,*F,*G,*H,
*I,*J,*K,*L,*M,*N,*O,*P,*Q,*R,*S,*T,*U,*V,*W,*X,*Y,
*Z,*1,*2,*3,*4,*5,*6,*7,*8,*9,*0,*!,*",*%,*&,*/,
*(,*),*=,*?,**,*_,*:,*;,*>,*',*+,*#,*-,*.,*,,*<

ENDSPEC


PARM ELEM

USE SSPECS : Bool
```

```
PUBLIC SORTS : elem

ENDSPEC


PSPEC LISTE (#x:Elem)

USE SSPECS : Bool,Nat
    PARM   : #x

PUBLIC SORTS : liste

PUBLIC OPS :
nil:-->liste
cons:elem-->liste
car:liste-->elem
cdr:liste-->liste
append:liste liste -->liste
entferne:elem liste-->liste
lösche_doppelte_elem:liste-->liste
l_rest,l_anfang:elem liste-->liste
ist_enth:elem liste -->bool
ist_leer:liste-->bool
i-tes_element:liste nat-->elem

PROPERTIES :
/*
car (l) liefert das erste Element der Liste l.
*/
∀ liste l:
    (ist_leer (l)=true ∧ car (l)=error)
  ∨ (∃ elem e: ∃ liste l1:
      l=cons (e,l1) ∧ car (l)=e)

/*
cdr (l) liefert die Liste l ohne das erste Element.
*/
∀ liste l:
    (ist_leer (l)=true ∧ cdr (l)=error)
  ∨ (∃ elem e: ∃ liste l1:
      l=cons (e,l1) ∧ cdr (l)=l1)

/*
append (l1,l2) fügt die zwei Listen l1 und l2 zu einer Liste zusammen.
*/
∀ liste l1,l2,l3:
  append (l1,append (l2,l3))=append (append (l1,l2),l3)
```



```
  ∧ ( (ist_leer (l1)=true ∧ append (l1,l2)=l2)
    ∨ (append (l1,l2)=cons (car (l1),append (cdr (l1),l2)))
    ∨ (ist_leer (l1)=false
      ∧ append (cons (car (l1),nil),cdr (l1))=l1))

/*
entferne (e,l) liefert die Liste l, in der alle Elemente e gelöscht
sind.
*/
∀ liste l: ∀ elem e:
  ist_enth (e,entferne (e,l))=false

/*
ist_leer (l) liefert true, falls l die leere Liste ist.
*/
∀ liste l: ist_leer (l)=true
  ⇒ ¬ ∃ elem e: ist_enth (e,l)=true

/*
ist_enth (e,l) liefert true, falls das Element e in der Liste l vor-
kommt.
*/
∀ elem e: ∀ liste l:
  ist_enth (e,l)=true ⇒ (car (l)=e ∨ ist_enth (e,cdr (l))=true)

/*
lösche_doppelte_elem (l) liefert die Liste l, in der kein Element dop-
pelt oder mehrfach vorkommt.
*/
∀ liste l: ∀ elem e:
  ist_enth (e,l)=true
  ⇒ (ist_enth (e,l_anfang (e,lösche_doppelte_elem (l)))=true
    ∧ ist_enth (e,l_rest (e,lösche_doppelte_elem (l)))=false)

/*
l_anfang (e,l) liefert die Liste, die aus den ersten Elementen von l
einschließlich e besteht, oder die Liste l, falls e nicht in l enthal-
ten ist.
l_rest (e,l) liefert die Liste, die alle Elemente aus l enthält, die
übrigbleiben, wenn man den Anfang der Liste bis einschließlich Element
e löscht.
*/
∀ liste l: ∀ elem e:
  append (l_anfang (e,l),l_rest (e,l))=l

∀ liste l: ∀ elem e:
  ∃ liste l1: l1=l_anfang (e,l)
    ∧ ( (ist_leer (l)=true ⇒ ist_leer (l1)=true)
```

```
        v (ist_enth (e,l)=false ⇒ l1=l)
        v (ist_enth (e,l)=true
            ⇒ V nat n: (n≠0 ∧ n≤anzahl (l1))
                ⇒ (i-tes_element (l1,n)=i-tes_element (l,n)
                    ∧ i-tes_element (l1,anzahl (l1))=e))

/*
i-tes_element  (l,i)  liefert das i-te Element der Liste l,  falls i>0
und i≤anzahl (l).
*/
V liste l: V nat i:
    (i-tes_element (l,i)=error ∧ (i>anzahl (l) v i=o)
   v (i=1 ∧ i-tes_element (l,i)=car (l))
   v (i-tes_element (l,i)=
        i-tes_element (cdr (l),sub (i,1)))

CONSTRUCTORS :
*nil
*cons

PRIVATE OPS :
anzahl_1:liste nat-->nat

DEFINE OPS :
car (l):= case l is
          *nil-->error.elem
          *cons (e,l1)-->e
          esac ;

cdr (l):= case l is
          *nil-->error.liste
          *cons (e,l1)-->l1
          esac ;

append (x,y):= case x is
               *nil-->y
               *cons (e,x1)-->cons (e,append (x1,y))
               esac ;

entferne (e,l):= case l is
                 *nil-->l
                 *cons (e1,l1)--> if eq.elem (e,e1)
                                  then entferne (e,l1)
                                  else cons (e1,entferne (e,l1))
                                 fi
                 esac ;

anzahl (l):=anzahl_1 (l,0);
```



```
anzahl_1 (l,n):= case l is
                 *nil-->n
                 *cons (e,l1)-->anzahl_1 (l1,incr (n))
                 esac ;

lösche_doppelte_elem (l):=
  case l is
  *nil-->nil
  *cons (e,l1)-->cons (e,lösche_doppelte_elem (entferne (e,l1)))
  esac ;

l_anfang (e,l):=
  case l is
  *nil-->nil
  *cons (e1,l)--> if eq.elem (e,e1)
                  then cons (e,nil)
                  else cons (e1,l_anfang (e,l1))
                 fi
  esac ;

l_rest (e,l):=
  case l is
  *nil-->nil
  *cons (e1,l1)--> if eq.elem (e,e1)
                   then l1
                   else l_rest (e,l1)
                  fi
  esac ;

ist_leer (l):= case l is
               *nil-->true
               *cons (e,l1)-->false
               esac ;

ist_enth (e,l):=
  case l is
  *nil-->false
  *cons (e1,l1)--> if eq.elem (e,e1)
                   then true
                   else ist_enth (e,l1)
                  fi
  esac ;

i-tes_element (l,n):=
   if ist_gt (n,anzahl (l)) or eq.nat (n,0)
     then error.elem
     else case l is
          *nil-->error.elem
```

```
        *cons (e,l1)--> if eq.nat (n,succ (0))
                        then l1
                        else i-tes_element (l1,sub (n,succ (0)))
                        fi
        esac
   fi ;

ENDSPEC


PSPEC TUPEL (#x1,#x2:Elem)

USE SSPECS : Bool
     PARM  : #x1, #x2

PUBLIC SORTS : tupel

PUBLIC OPS :
erz_tup:#x1.elem #x2.elem-->tupel
p_1:#x1.elem tupel-->tupel
p_2:#x2.elem tupel-->tupel
s_1:tupel-->#x1.elem
s_2:tupel-->#x2.elem

PROPERTIES :
/*
s_1 (t) und s_2 (t) sind Selektor-Operationen,  die die erste bzw. die
zweite Komonente des Tupels liefern.
p_1  (x,t) und p_2 (x,t) sind Schreib-Operationen,  die die erste bzw.
zweite Komponente des Tupels mit x überschreiben.
*/
V tupel t: ∃ elem x,y:
   t=erz_tup (x,y) ∧ s_1 (t)=x ∧ s_2 (t)=y
   ∧ V elem e: p_1 (e,t)=erz_tup (e,y)
               ∧ p_2 (e,t)=erz_tup (x,e)

V tupel t: V elem x,y:
   p_1 (x,p_1 (y,t))=p_1 (x,t)
   ∧ p_1 (s_1 (t),t)=t
   ∧ p_2 (x,p_2 (y,t))=p_2 (x,t)
   ∧ p_2 (s_2 (t),t)=t

CONSTRUCTORS :
*erz_tup

DEFINE OPS :
p_1 (e,t):= case t is
```



```
           *erz_tup (x,y)-->erz_tup (e,y)
           esac ;

p_2 (e,t):= case t is
           *erz_tup (x,y)-->erz_tup (x,e)
           esac ;

s_1 (t):= case t is
          *erz_tup (x,y)-->x
          esac ;

s_2 (t):= case t is
          *erz_tup (x,y)-->y
          esac ;

/*
Bemerkung:  In analoger Weise werden auch sämtliche n_Tupel,  die  für
die Auswertung benötigt werden, spezifiziert. Um Schreibarbeit zu spa-
ren sind diese Spezifikationen weggelassen.
Für  die  Operationsnamen wurde folgendes festgelegt,  das am Beispiel
n=4 erläutert wird.
Name des Tupels: 4_Tupel
Name des Konstruktors: erz_4_tup
Name der Selektor- bzw. Schreiboperation: svier_i bzw. pvier_i,  wobei
i zwischen 1 und 4 liegt und die jeweilige Komponente meint.
*/

ENDSPEC


PSPEC BAUM (#x:Elem)

USE SSPECS : Bool
    PSPEC  : Liste (#x)
    PARM   : #x

PUBLIC SORTS : baum

PUBLIC OPS :
erz_baum:elem-->baum
son:baum baum-->baum
anf:elem elem baum-->baum
wurzel:baum-->elem
preorder:baum-->liste
l_d_söhne:elem baum-->liste
vater:elem baum-->elem
t_baum:elem baum-->baum
```

```
blätter:baum-->liste
ist_in_baum:elem baum-->bool

PROPERTIES :
/*
ist_in_baum (x,b) liefert true, falls der Knoten x ein Knoten des
Baumes b ist.
*/
V baum b: V elem x:
   (b=erz_baum (x)
    v ∃ baum b1,b2: b=son (b1,b2)
      ∧ (ist_in_baum (x,b1)=true v ist_in_baum (x,b2)=true)))
  ⇒ ist_in_baum (x,b)=true

/*
wurzel (b) liefert die Wurzel des Baumes b.
*/
V baum b: V elem x:
  ((b=erz_baum (x) ⇒ wurzel (b)=x)
   v ∃ baum b1,b2: b=son (b1,b2)
       ∧ wurzel (b)=wurzel (b1))
  ∧ vater (wurzel (b))=error

/*
preorder (b) liefert die Liste aller Knoten in b in PREORDER Reihen-
folge.
*/
V baum b: V elem x:
  ((b=erz_baum (x) ⇒ preorder (b)=cons (x,nil))
   v ∃ baum b1,b2: b=son (b1,b2)
    ∧ preorder (b)=append (preorder (b1),preorder (b2)))
  ∧ (ist_in_baum (x,b)=true ⇒ ist_enth (x,preorder (b))=true)
  ∧ car (preorder (b))=wurzel (b)

/*
t_baum (x,b) liefert den Teilbaum aus b, dessen Wurzel x ist, sofern x
in Baum b vorkommt.
*/
V baum b: V elem x:
  ist_in_baum (x,b)=true
    ⇒ (t_baum (x,b)≠error
      ∧ (V elem y: ist_in_baum (y,t_baum (x,b))=true
         ⇒ ist_in_baum (y,b)=true)
      ∧ wurzel (t_baum (x,b))=x))

/*
l_d_söhne (x,b) liefert die Liste der Elemente, die Nachfolger des
Knotens x in Baum b sind. Diese Liste ist leer, wenn Knoten x Blatt in
```



```
Baum b ist, oder wenn Knoten x nicht in Baum b vorkommt.
*/
V baum b: V elem x:
  ∃ liste l: l=l_d_söhne (x,b)
  ∧ V elem y: ist_enth (y,l)=true ⇒ vater (y,b)=x

/*
anf (x,y,b) fügt in Baum b Knoten x an den Knoten y als rechten Sohn
an, sofern Knoten y in Baum b vorkommt. Im anderen Fall bleibt Baum b
unverändert.
*/
V baum b: V elem x:
  ist_in_baum (y,b)=true
  ⇒ (ist_enth (x,l_d_söhne (y,anf (x,y,b)))=true
    ∧ i-tes_element (l_d_söhne (y,anf (x,y,b)),
                     anzahl (l_d_söhne (y,anf (x,y,b))))=x

/*
vater (x,b) liefert den direkten Vorgänger von Knoten x in Baum b oder
error, wenn x gleich der Wurzel von b ist oder x nicht in b vorkommt.
*/
V baum b: V elem x:
  ∃ elem y: y=vater (x,b)
  ∧ ((y=error ⇒ (x=wurzel (b) v ist_in_baum (x,b)=false))
     v (y≠error ⇒ ist_enth (x,l_d_söhne (y,b))=true))

/*
blätter (b) liefert die Liste aller Blätter des Baumes b.
*/
V baum b: V elem x:
  ist_enth (x,blätter (b))=true
  ⇒ ist_leer (l_d_söhne (x,b))=true

CONSTRUCTORS :
*erz_baum
*son

/*
Der Term *son (b1,b2) ist so zu interpretieren:
Der Baum b2 wird als rechter Nachfolger an die Wurzel von b1 ange-
hängt.
*/

PRIVATE OPS
vater_1:liste elem elem baum-->elem
blätter_1:baum-->liste

DEFINE OPS :
```

```
wurzel (t):= case t is
             *erz_baum (e)-->e
             *son (t1,t2)-->wurzel (t1)
             esac ;

preorder (t):= case t is
               *erz_baum (e)-->cons (e,nil)
               *son (t1,t2)-->append (preorder (t1),preorder (t2))
               esac ;

anf (e1,e2,t):=
   if eq.elem (e2,wurzel (t))
     then son (t,erz_baum (e1))
     else case t is
          *erz_baum (e)-->erz_baum (e)
          *son (t1,t2)--> if ist_in_baum (e2,t1)
                            then son (anf (e1,e2,t1),t2)
                            else son (t1,anf (e1,e2,t2))
                          fi
          esac
   fi ;

l_d_söhne (e,t):=
   case t is
   *erz_baum (e1)-->nil
   *son (t1,t2)-->
    if eq.elem (e,wurzel (t1))
      then append (l_d_söhne (e,t1),cons (wurzel (t2),nil))
      else case t1 is
           *erz_baum (e2)-->l_d_söhne (e,t2)
           *son (b1,b2)-->
             if ist_in_baum (e,b1)
               then l_d_söhne (e,b1)
               else if ist_in_baum (e,b2)
                       then l_d_söhne (e,b2)
                       else nil
                    fi
             fi
           esac
    fi
   esac ;

vater (e,t):=
   if eq.elem (e,wurzel (t))
     then error.elem
     else vater_1 (l_d_söhne (wurzel (t),t)),wurzel (t),e,t)
   fi ;
```



```
vater_1 (l,v,s,t):=
   case l is
   *nil-->error.elem
   *cons (e,l1)-->
     if ist_enth (s,l)
       then v
       else if ist_in_baum (s,t_baum (e,t))
               then vater_1 (l_d_söhne (e,t),e,s,t_baum (e,t)
               else vater_1 (l1,v,s,t)
            fi
     fi
   esac ;

ist_in_baum (e,t):=
   case t is
   *erz_baum (e1)--> if eq.elem (e,e1)
                       then true
                       else false
   *son (t1,t2)-->ist_in_baum (e,t1) or ist_in_baum (e,t2)
   esac ;

t_baum (e,t):=
   case t is
   *erz_baum (e1)--> if eq.elem (e,e1)
                       then erz_baum (e1)
                       else error.baum
   *son (t1,t2)--> if ist_in_baum (e,t1)
                     then if eq.elem (wurzel (t1),e)
                             then son (t1,t2)
                             else t_baum (e,t1)
                          fi
                     else t_baum (e,t2)
                   fi
   esac ;

blätter (t):=
   case t is
   *erz_baum (e)-->cons (e,nil)
   *son (t1,t2)-->
     case t1 is
     *erz_baum (b1)--> case t2 is
                       *erz_baum (e1)-->cons (e1,nil)
                       otherwise-->blätter_1 (t2)
                       esac
     *son (b1,b2)-->append (blätter_1 (t1),blätter (t2)
     esac
   esac ;
```

```
blätter_1 (t):=
   case t is
   *erz_baum (e)-->error.liste
   *son (t1,t2)--> case t1 is
                   *erz_baum (e1)-->blätter (t2)
                   *son (b1,b2)-->blätter (t)
                   esac
   esac ;


INSTANTIATE Liste to Charstring
ACTUALIZE : #x by Char
            SORTS : elem by char
RENAME : SORTS : liste by charstring
           OPS : nil by blank
                 cons by create
                 append by concat

/*
Bemerkung: In den folgenden Spezifikationen werden die Konstruktoren
eines charstring weggelassen, wenn es eindeutig ist, daß es sich um
einen charstring handelt.
Bsp. Statt create (E,create (N,create (D,blank))) schreibt man einfach
END.
*/

END


INSTANTIATE Liste to Name
ACTUALIZE : #x by Char
            SORTS : elem by char
RENAME : SORTS : liste by name
           OPS : nil by bl
                 cons by cr
                 append by cc

/*
Bemerkung: Charstring und Name spezifizieren den gleichen Datentyp. Da
es offensichtlich ist, für Namen von Prozeduren, Dateien oder ähnli-
chem tatsächlich die Spezifikation Name zu benutzen, hingegen für
Terminalsymbole einer Grammatik, Attributen, etc. die Spezifikation
Charstring, wurde dieser Datentyp namentlich unterschieden. Analog zu
Charstring gilt, daß die Konstruktoren im eindeutigen Fall weggelassen
werden.
*/
```



```
END
```

3.1.2. Die Symboltabelle (ST)

Die ST besteht im Prinzip aus einer Tabelle, die in zwei Spalten geteilt ist. Die erste Spalte enthält die Namen der im Programmtext vorkommenden Bezeichner (Namen). Die zweite Spalte enthält zu den jeweiligen Bezeichnern eine Liste von Attributen, die die in der lexikalischen und syntaktischen Analyse gesammelte Information über diesen Bezeichner darstellen.

3.1.2.1. Organisation

In blockorientierten Programmiersprachen (ALGOL, PL/1, SPL 3) gilt, daß die Attribute eines Namens von der Umgebung abhängen, in der der Name deklariert ist. So können die Attribute eines Namens in unterschiedlichen Blöcken völlig verschieden sein. Die ST wird deshalb baumförmig strukturiert. Jeder Knoten in der ST ist entweder ein Block oder ein Gebiet.

Ein Block ist ein Prozedurblock oder ein Beginblock, der Verweise auf die enthaltenen Prozedurblöcke, Beginblöcke und Gebiete enthält.

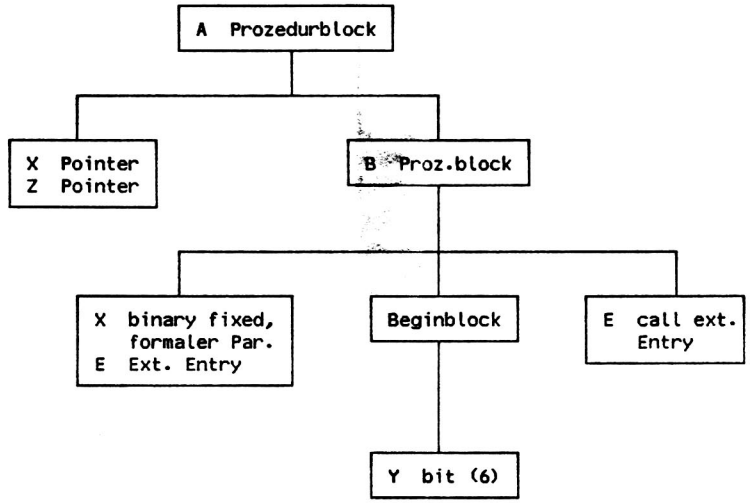
Ein Gebiet enthält die im entsprechenden Teil des Programms deklarierten Namen und die zugehörigen Attribute.

3.1.2.2. Beispiel

Das Programm:

```
A : PROCEDURE;
    DECLARE (X,Z) POINTER;
    B : PROCEDURE (X);
        DECLARE E ENTRY EXTERNAL;
        DECLARE X BINARY FIXED;
        BEGIN
             DECLARE Y BIT (6);
             ...
        END;
        CALL E;
        ...
    END;
    ...
END A;
```


Die Symboltabelle:



Bemerkung: Es wird vorausgesetzt, daß die Aufrufe externer Entries explizit als Eintrag in die Symboltabelle von ANALYSE geschrieben werden.

```
INSTANTIATE Liste to Attributliste
ACTUALIZE : #x by Charstring
            SORTS : elem by charstring
RENAME : SORTS : liste by attributliste
           OPS : nil by nil_attr
                 cons by cons_attr
                 ist_enth by ist_enth_attr

/*
Bemerkung: Um Schreibarbeit bei der Umbenennung der Operationsnamen zu
sparen,  wird in dieser Spezifikation und in allen folgenden vorausge-
setzt, daß sämtliche Operationsnamen der Aktualisierung mit den Opera-
tionsnamen der parametrisierten Spezifikation übereinstimmen und durch
eine Endung ergänzt werden,  die aus dem aufgeführten Beispiel hervor-
geht.
In  dieser  Aktualisierung  wird  jeder Operationsname der PSPEC Liste
durch das Suffix "_attr" erweitert.
*/

END


INSTANTIATE Tupel to Eintrag
ACTUALIZE : #x1 by Name
            #x2 by Attributliste
            SORTS : #x1.elem by name
                    #x2.elem by attributliste
RENAME : SORTS : tupel by eintrag
           OPS : erz_tup by erz_etr

                 s_1 by setr_1
                 s_2 by setr_2
                 p_1 by petr_1
                 s_2 by petr_2

/*
Bemerkung: Bei der Aktualisierung von n-Tupeln wird jede Schreib- bzw.
Selektoroperation durch das Einfügen einer Abkürzung in entsprechender
Weise umbenannt.
*/

END


INSTANTIATE Liste to Gebiet
ACTUALIZE : #x by Eintrag
```



```
            SORTS : elem by eintrag
RENAME : SORTS : liste by gebiet
           OPS : nil by nil_gb
                 cons by cons_gb
END


SSPEC BLOCK

USE SSPECS : Bool,Name

PUBLIC SORTS : block
PUBLIC OPS :
proz_blck:name-->block
beg_blck:-->block

CONSTRUCTORS :
*proz_blck
*beg_blck

ENDSPEC


INSTANTIATE Tupel to Block_Nr
ACTUALIZE : #x1 by Block
            #x2 by Nat
            SORTS : #x1.elem by block
                    #x2.elem by nat
RENAME : SORTS : tupel by block_nr
           OPS : erz_tup by erz_bnr
                 s_1 by sbnr_1
END


SSPEC SYMTAB

USE SSPECS : Bool, Block_Nr, Gebiet

PUBLIC SORTS : symtab

PUBLIC OPS :
blck_b:block_nr-->symtab
blck_g:gebiet-->symtab

CONSTRUCTORS :
```

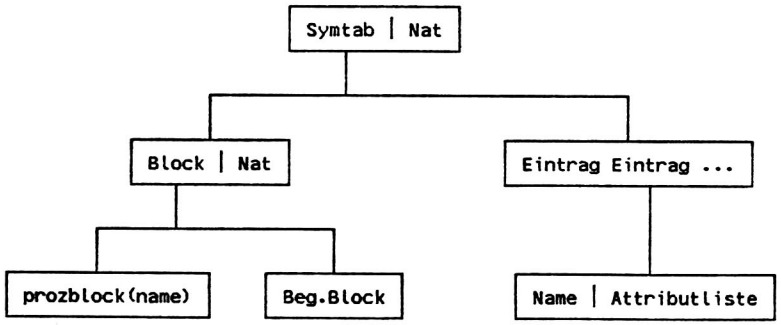
```
*blck_b
*blck_g

ENDSPEC


INSTANTIATE Tupel to Knoten_st
ACTUALIZE : #x1 by Symtab
            #x2 by Nat
            SORTS : #x1.elem by symtab
                    #x2.elem by nat
RENAME : SORTS : tupel by knoten_st
           OPS : erz_tup by erz_kst
                 s_1 by skst_1
                 s_2 by skst_2
                 p_1 by pkst_1
                 p_2 by pkst_2

/*
Die zweite Komponente Nat eines Knotens der ST dient  der  eindeutigen
Identifizierung eines Knotens.

Ein Knoten der ST hat somit folgenden Aufbau:
Die  Kanten  des  Baumes sind von oben nach unten als "besteht aus" zu
lesen. Eine Verzweigung wird durch "oder" interpretiert.
```




```
*/

END
```



```
INSTANTIATE Baum to Symboltabelle_1
ACTUALIZE : #x by Knoten_st
            SORTS : #x.elem by knoten_st
RENAME : SORTS : baum by st
           OPS : erz_baum by erz_st
                 son by son_st

/*
Bemerkung: Im Abschnitt Instanzbildung des Kapitels über Parametrisie-
rung (S.  16) wurde bbereits angedeutet,  daß das Bilden einer Instanz
eine  weitere Instanzbildung zur Folge haben kann.  In dieser Aktuali-
sierung tritt dieser Fall auf.

Außer der Aktualisierung Baum zu Symboltabelle_1  durch  Ersetzen  des
Parameters  x  durch  die  SSPEC  Knoten_st,  ist noch die PSPEC Liste
(#x.elem) entsprechend zu aktualisieren.
Dies geschieht wie folgt:
Die Sorte liste wird in l(knoten_st) umbenannt. Sämtliche Listenopera-
tionen erhalten die Endung "_lst".
*/

END


INSTANTIATE Liste to L(Name)
ACTUALIZE : #x by Name
            SORTS : #x.elem by name
RENAME : SORTS : liste by l(name)
           OPS : nil by nil_ln
                 cons by cons_ln
                 ist_enth by ist_enth_ln
END


SSPEC SYMBOLTABELLE

USE SSPECS : Bool, Nat, Symboltabelle_1, L(Name)

PUBLIC OPS :
ist_proz_block:knoten_st-->knoten_st
ist_beg_block:knoten_st-->bool
ist_entry_enth:knoten_st-->bool
entryname:knoten_st-->name
blockname:knoten_st-->name
give_blocknr:knoten_st-->nat
ist_dekl_in:name gebiet-->bool
```

```
entryliste:knoten_st-->l(name)

PROPERTIES :
/*
ist_dekl_in (n,gb) liefert true,  falls der Name n im Gebiet gb dekla-
riert ist.
*/
∀ gebiet gb: ∀ name n:
  ∃ bool b: b=ist_dekl_in (n,gb)
    ∧ ((b=false
        ∧ ∀ eintrag e: ist_enth_gb (e,gb)=true
                       ⇒ setr_1 (e)≠n)
       ∨ (b=true
           ∧ ∃ eintrag e: ist_enth_gb (e,gb)=true
               setr_1 (e)=n))

/*
ist_proz_block  (kst)  liefert true,  falls es sich bei dem Knoten kst
der ST um einen Prozedurblock handelt.
*/
∀ knoten_st kst: ∃ symtab s: ∃ nat n:
  kst=erz_kst (s,n)
  ∧ ∃ bool b: b=ist_proz_block (k) ∧ b=ist_proz_block_1 (s)

∀ symtab s: ∃ bool b:
  b=ist_proz_block_1 (s)
  ∧ ((∃ gebiet gb: s=blck_g (gb) ∧ b=false)
     ∨ (∃ blck_nr bnr: s=blck_b (bnr)
          ∧ ∃ block bk: ∃ nat n:
              bnr=erz_bnr (bk,n) ∧ b=ist_proz_block_2 (bk)))

∀ block bk: ∃ bool b:
  b=ist_proz_block_2 (bk)
  ∧ ((bk=beg_blck ∧ b=false)
    ∨ (∃ name n: bk=proz_blck (n) ∧ b=true))

/*
ist_beg_blck (kst) liefert true,  falls es sich bei dem Knoten kst der
ST um einen Begin Block handelt.
*/
∀ knoten_kst: ist_beg_blck (kst)=true ⇒ ist_proz_blck (kst)=false

/*
ist_entry_enth (kst) liefert true,  wenn es sich bei dem Knoten der ST
um ein Gebiet handelt, in dem die Deklaration eines Sec. Entry enthal-
ten ist.
*/
∀ knoten_st kst: ist_entry_enth (kst)=true
```



```
  ⇒ ∃ gebiet gb: ∃ nat n: kst=erz_kst (blck_g (gb),n)
       ∧ ∃ eintrag e:
           ist_enth_gb (e,gb)=true
           ∧ ist_enth_attr (Sec. Entry,setr_2 (e))=true

/*
entryname (kst) liefert,  wenn kst ein Gebiet darstellt,  das Deklara-
tionen von Sec.  Entries enthält,  den Namen des ersten Sec.  Entry in
der Deklaration.
*/
∀ knoten_st kst: ∃ name n: n=entryname (kst)
  ∧ (n≠error
    ⇒ ∃ nat n1: ∃ gebiet gb:
        kst=erz_kst (blck_g (gb),n1)
        ∧ ∃ eintrag e: ∃ nat n2: ist_enth_gb (e,gb)=true
            ∧ i-tes_element (gb,n2)=e
            ∧ ist_enth_attr (Sec. Entry,setr_2 (e))=true
            ∧ n=setr_1 (e)
            ∧ ∀ nat n3: n3=0 ∨ n3≥n2
                ∨ ¬ ∃ eintrag e1: ist_enth_gb (e1,gb)=true
                      ∧ i-tes_element (gb,n3)=e1
                      ∧ ist_enth_attr (Sec. Entry,setr_2 (e1))=true)

/*
blockname (kst)  liefert den Namen des Prozedurblocks,  sofern es sich
beim Knoten kst um einen Prozedurblock handelt.
*/
∀ knoten_st kst: ∃ name p:
  p=blockname (kst)
  ∧ (p≠error ⇒ ∃ block bk: ∃ nat n1,n2:
                 kst=erz_kst (blck_b (erz_bnr (bk,n1)),n2)
                 ∧ bk=proz_blck (p))

/*
give_blocknr (kst) liefert die Blocknummmer des Knotens kst, falls kst
Prozedurblock oder Beginblock ist.
*/
∀ knoten_st kst: ∃ nat n:
  n≠error ⇒ ∃ block bk: ∃ nat n1:
              kst=erz_kst (blck_b (erz_bnr (bk,n)),n1)

/*
entryliste  (kst)  liefert alle Namen,  der im Knoten kst deklarierten
Sec. Entries, sofern kst ein Gebiet darstellt.
*/
∀ knoten_st kst: ∃ l(name) le:
  le=entryliste (kst)
  ∧ (ist_leer_ln (le)=false
```

```
    ⇒ ∀ name n: ist_enth_ln (n,le)=true
        ⇒ ∃ nat n1: ∃ gebiet gb:
            kst=erz_kst (blck_g (gb),n1)
            ∧ ∃ eintrag e: ist_enth_gb (e,gb)=true
                ∧ ist_enth_attr (Sec. Entry,setr_2(e))_true
                ∧ n=setr_1 (e))

PRIVATE OPS :
ist_proz_block_1:symtab-->bool
ist_proz_block_2:block-->bool
ist_beg_block_1:symtab-->bool
ist_beg_block_2:block-->bool
ist_entry_enth_1:symtab-->bool
ist_entry_enth_2:block-->bool
blockname_1:symtab-->name
blockname_2:block-->name
entryname_1:symtab-->name
entryname_2:gebiet-->name
give_blocknr_1:symtab-->nat
entryliste_1:symtab-->l(name)
entryliste_2:gebiet-->l(name)

DEFINE OPS :
ist_proz_block (kst):=ist_proz_block_1 (skst_1 (kst));

ist_proz_block_1 (s):=
   case s is
   *blck_b (bn)-->ist_proz_block_2 (sbnr_1 (bn))
   *blck_g (g)-->false
   esac ;

ist_proz_block (b):= case b is
                     *proz_blck (p)-->true
                     *beg_blck-->false
                     esac ;

ist_begin_block (kst):=ist_begin_block_1 (skst_1 (kst));

ist_begin_block_1 (s):=
   case s is
   *blck_b (bn)-->ist_begin_block_2 (sbnr_1 (bn))
   *blck_g (g)-->false
   esac ;

ist_begin_block_2 (b):=
   case b is
   *proz_blck (p)-->false
   *beg_blck-->true
```



```
   esac ;

ist_entry_enth (kst):=ist_entry_enth_1 (skst_1 (kst));

ist_entry_enth_1 (s):= case s is
                       *blck_b (bn)-->false
                       *blck_g (g)-->ist_entry_enth_2 (g)
                       esac ;

ist_entry_enth_2 (g):=
   case g is
   *nil_gb-->false
   *cons_gb (e,rest)--> if ist_enth_attr (Sec. Entry,setr_2 (e))
                         then true
                         else false
                        fi
   esac ;

blockname (kst):=blockname_1 (skst_1 (kst));

blockname_1 (s):=
   case s is
   *blck_b (bn)-->blockname_2 (sbnr_1 (bn))
   *blck_g (g)-->error.name
   esac ;

blockname_2 (b):= case b is
                  *proz_blck (bn)-->p
                  *beg_blck-->error.name
                  esac ;

entryname (kst):=entryname_1 (skst_1 (kst));

entryname_1 (s):=
   case s is
   *blck_b (bn)-->error.name
   *blck_g (g)-->entryname_2 (g)
   esac ;

entryname_2 (g):=
   case g is
   *nil_gb-->error.name
   *cons_gb (e,rest)-->
      if ist_enth_attr (Sec. Entry,setr_2 (e))
        then setr_1 (e)
        else entryname_2 (rest)
      fi
   esac ;
```

```
give_blocknr (kst):=give_blocknr_1 (skst_1 (kst));

give_blocknr_1 (s):= case s is
                     *blck_b-->sbnr_2 (bn)
                     *blck_g (gb)-->error.nat
                     esac ;

entryliste (kst):=entryliste_1 (skst_1 (kst));

entryliste_1 (s):= case s is
                   *blck_b (bn)-->error.l (name)
                   *blck_g (gb)-->entryliste_2 (gb)
                   esac ;

entryliste_2 (gb):=
   case gb is
   *nil_gb-->nil_ln
   *cons_gb (e,rest)-->
      if ist_enth_attr (Sec. Entry,setr_2 (e))
        then cons_ln (setr_1 (e),entryliste_2 (rest))
        else entryliste_2 (rest)
      fi
   esac ;

ist_dekl_in (n,gb):=
   case gb is
   *nil_gb-->false
   *cons_gb (e,rest)--> if eq.name (setr_1 (e)=n)
                        then true
                        else ist_dekl_in (n,rest)
                       fi
   esac ;

ENDSPEC
```


3.1.3. Der Strukturbaum (SB)

Der SB enthält die syntaktische Zerlegung eines Programms. Jeder Knoten des SB besteht dabei nicht nur aus einem terminalen oder nichtterminalen Symbol der zugrundeliegenden Grammatik, sondern mit jedem Knoten sind noch weitere Informationen verbunden, die während der Zerlegung gesammelt werden.

Ein solcher Knoten wird Stammsatz genannt, und er ist in den Schlüssel und den Datenteil gegliedert.

Der folgenden Beschreibung liegt die SIEMENS-Darstellung zugrunde. Nicht alle Informationen im Stammsatz werden in dieser Spezifikation der Auswertefunktionen tatsächlich benötigt. Diese Teile werden zwar mitspezifiziert, aber nicht weiter berücksichtigt.

3.1.3.1. Der Schlüssel

Der Schlüssel ist ein Feld, das aus mehreren Komponenten besteht und das in der folgenden Tabelle erläutert ist.

	Komponente	Bedeutung	Spezifikation
1.	Modulname	Name des Quellmoduls	Name
2.	Identifikationsnummer des Moduls	unberücksichtigt	Nat
3.	Abspeicherungskomponente	unberücksichtigt	Charstring
4.	Gruppe	unberücksichtigt	Charstring
5.	Identifikationsnummer des Untersuchungsobjekts	unberücksichtigt	Nat
6.	Satzart	Terminale und nichtt. Symbole der Grammatik	Charstring
7.	Statementnummer	Nr des Statements im Quellprogramm	Nat
8.	Zeilennummer	Nr der Zeile im Quellprogramm	Nat
9.	Includeaufrufnummer	Sequentielle Nummerierung der aufgelösten Includes	Nat
10.	Includezeilennummer	Zeile im Includemember	Nat
11.	Spaltennummer	Spaltennummer im Quellprogramm	Nat

3.1.3.2. Der Datenteil

Analog zum Schlüssel besteht der Datenteil aus mehreren Komponenten:

Komponente	Bedeutung	Spezifikation
1. Identifikationsnummer	Nr zur eindeutigen Identifizierung des Stammsatzes im SB	Nat
2. Name	unberücksichtigt	Name
3. Inhalt	Name von Identifiern, Zahlen	Name
4. Datentyp	unberücksichtigt	Charstring
5. Statementnummernbereich	Bereich in Stmts. z.B. einer Prozedur	Zahlenpaar
6. Zeilennummernbereich	Bereich in Zeilen	Zahlenpaar
7. Includeaufrufbereich	Bereich der benutzten Includes	Zahlenpaar
8. Includezeilenbereich	Bereich des Incl.members in Zeilen	Zahlenpaar
9. Spaltennummernbereich	Bereich in Spalten	Zahlenpaar
10. Blocknummer	Verweis auf den entsprechenden Block der ST	Nat

Bemerkung: In Abweichung von der SIEMENS Darstellung sind die Identifikationsnummer und die Blocknummer zusätzlich in den Datenteil aufgenommen.

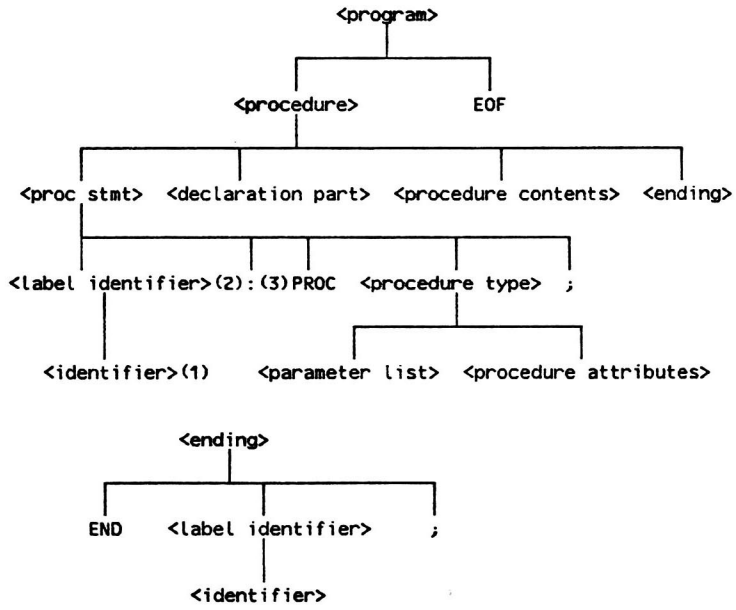
3.1.3.3. Beispiel

1. Das Programm:

```
     1 3 5 7 9.................27
1    P:PROCEDURE;
2      DECLARE(I,J)BINARY FIXED;
3      I=1; J=1;
4    END P;
```


2.Der Syntaxbaum

Der Syntaxbaum basiert auf der zugrundeliegenden SPL 3 Grammatik. Aus Platzgründen wird zuerst der zugehörige Syntaxbaum dargestellt und dieser auch noch verkürzt. Jeder Knoten in diesem Syntaxbaum steht stellvertretend für einen gesammten Stammsatz. Tatsächlich ist der Syntaxbaum ein Teil des Strukturbaums. Dieser Teil entsteht aus dem SB, indem sämtliche Komponenten bis auf die Komponente Satzart im Schlüssel des Stammsatzes weggelassen werden.



3.Einige Stammsätze

1. Schlüssel:	1. P	Datenteil: 1. 1
	2. ?	2. ?
	3. ?	3. P
	4. ?	4. ?
	5. ?	5. 1 - 1

	6.	Identifier	6. 1 - 1
	7.	1	7. undef.
	8.	1	8. undef.
	9.	undef.	9. 3 - 3
	10.	undef.	10. 1
	11.	3	
2. Schlüssel:	1.	P	Datenteil: 1. 2
	2.	?	2. ?
	3.	?	3. P
	4.	?	4. ?
	5.	?	5. 1 - 1
	6.	Label identifier	6. 1 - 1
	7.	1	7. undef.
	8.	1	8. undef.
	9.	undef.	9. 3 - 3
	10.	undef.	10. 1
	11.	3	
3. Schlüssel:	1.	P	Datenteil: 1. ?
	2.	?	2. ?
	3.	?	3. :
	4.	?	4. ?
	5.	?	5. 1 - 1
	6.	:	6. 1 - 1
	7.	1	7. undef.
	8.	1	8. undef.
	9.	undef.	9. 3 - 3
	10.	undef.	10. 1
	11.	4	

Bemerkung: Tatsächlich werden viele Teile des Stammsatzes hier nicht benötigt. Dies wird durch das Fragezeichen angedeutet.

In der SIEMENS Version von INTAKT werden Informationen über ein Quellprogramm, die bereits in der AUFBEREITUNG gewonnen werden, ebenfalls in Stammsätzen abgelegt. Außerdem sind von INTAKT noch weitere Versionen geplant. Dies zusammen erklärt, warum der Stammsatz aus so vielen Komponenten besteht.

Es ist einsichtig, daß der SB sehr umfangreich wird. In der Implementierung von INTAKT wird es nicht möglich sein, den SB als ganzes zu betrachten, wie es in dieser Spezifikation geschieht. Deshalb teilt SIEMENS den Stammsatz in Schlüssel und Datenteil. Die Verkettung von Stammsätzen wird in einer sogenannten Strukturdatei festgehalten. Da

dies Implementierungsentscheidungen sind, wird in dieser Spezifikation in dieser Hinsicht von der SIEMENS Darstellung abgewichen.

3.1.3.4. Spezifikation des SB

```
INSTANTIATE Tupel to Zahlenpaar
ACTUALIZE : #x1 by Nat
            #x2 by Nat
             SORTS : #x1.elem by nat
                     #x2.elem by nat
RENAME : SORTS : tupel to zahlenpaar
           OPS : erz_tup by erz_zp
                 s_1 by szp_1
                 s_2 by szp_2
END


INSTANTIATE 11_Tupel to Schlüssel
ACTUALIZE : #x1 by Name, #x2 by Nat, #x3,#x4 by Charstring
            #x5 by Nat, #x6 by Charstring
            #x7, #x8, #x9, #x10, #x11 by Nat
            SORTS : #x1.elem by name, #x2.elem by nat
                    #x3.elem, #x4.elem by charstring
                    #x5.elem by nat, #x6.elem by charstring
                    #x7.elem, #x8.elem, #x9.elem by nat
                    #x10.elem, #x11.elem by nat
RENAME : SORTS : 11_tupel by schlüssel
           OPS : erz_11_tup by erz_sl
                 self_1 by ssl_1
END


INSTANTIATE 10_Tupel to Datenteil

ACTUALIZE : #x1 by Nat,#x2, #x3 by Name, #x4 by Charstring
            #x5, #x6, #x7, #x8, #x9 by Zahlenpaar
            #x10 by Nat
            SORTS : #x1.elem by nat, #x2.elem, #x3.elem by name
                    #x4.elem by charstring, #x5.elem by nat
                    #x6.elem, #x7.elem, #x8.elem by nat
                    #x9.elem by zahlenpaar
                    #x10.elem by nat
RENAME : SORTS : 10_tupel by datenteil
           OPS : erz_10_tup by erz_dt
                 szehn_1 by sdt_1
END
```



```
INSTANTIATE Tupel to Stammsatz
ACTUALIZE : #x1 by Schlüssel
            #x2 by Datenteil
             SORTS : #x1.elem by schlüsssel
                     #x2.elem by datenteil
RENAME : SORTS : tupel by stammsatz
           OPS : erz_tup by erz_sts
                 s_1 by ssts_1
END


INSTANTIATE Baum to Strukturbaum_1
ACTUALIZE : #x by Stammsatz
             SORTS : #x.elem by stammsatz
RENAME : SORTS : baum by sb
           OPS : erz_baum by erz_sb
                 son by son_sb
                 vater by vater_sb

END


SSPEC STRUKTURBAUM

USE SSPECS : Strukturbaum_1, Nat, Name, Charstring

PUBLIC OPS :
enth_vater_proc:stammsatz sb-->stammsatz
enth_vater_proc_beg:stammsatz sb-->stammsatz
gen_l_stmtnr:nat sb-->l(stammsatz)
blocknr_zu_stmtnr:nat sb-->nat
vq_sts_proc_beg_zu_blocknr:nat sb-->stammsatz
vq_sa:name charstring sb-->stammsatz

PROPERTIES :
/*
enth_vater_proc  (s,sb)  liefert zum Stammsatz s den Vater mit Satzart
"procedure" im Strukturbaum sb oder error,  falls s nicht Stammsatz im
Strukturbaum sb ist oder keinen Vater mit Satzart "procedure" hat.
*/
∀ sb sb: ∀ stammsatz s: ∃ stammsatz s1:
  s1=enth_vater_proc (s,sb)
  ∧ (s1≠error
    ⇒ ssl_6 (ssts_1 (s1))=procedure
      ∧ ist_in_baum_sb (s1,sb)=true
      ∧ ist_in_baum_sb (s,t_baum_sb (s1,sb))=true
```

```
      ∧ ¬ ∃ stammsatz s2: ist_in_baum_sb (s2,sb)=true
            ∧ ssl_6 (ssts_1 (s2))=procedure
            ∧ ist_in_baum_sb (s,t_baum_sb (s2,sb))=true
            ∧ ist_in_baum_sb (s2,t_baum_sb (s1,sb))=true))

/*
enth_vater_proc_beg (s,sb) liefert in analoger Weise zur Operation
enth_vater_proc den Vater mit Satzart "procedure" bzw. "begin block"
im Strukturbaum sb.
*/
∀ sb sb: ∀ stammsatz s: ∃ stammsatz s1:
  s1=enth_vater_proc_beg (s,sb)
  ∧ (s1≠error
    ⇒ ((ssl_6 (ssts_1 (s1))=procedure
      ∨ ssl_6 (ssts_1 (s1))=begin block)
     ∧ ist_in_baum_sb (s,t_baum_sb (s1,sb))=true
     ∧ ist_in_baum (s1,sb)=true
     ∧ ¬ ∃ stammsatz s2: ist_in_baum_sb (s2,sb)=true
           ∧ (ssl_6 (ssts_1 (s2))=procedure
             ∨ ssl_6 (ssts_1 (s2))=begin block)
           ∧ ist_in_baum_sb (s2,t_baum_sb (s1,sb))=true
           ∧ ist_in_baum_sb (s,t_baum_sb (s2,sb))=true))

/*
gen_l_stmtnr (n,sb) liefert die Liste aller Stammsätze im Strukturbaum
sb mit Statementnummer n.
*/
∀ sb sb: ∀ nat n: ∃ l(stammsatz) l:
  l=gen_l_stmtnr (n,sb)
  ∧ ((ist_leer_lsb (l)=true
      ∧ ¬ ∃ stammsatz s:
            ist_in_baum_sb (s,sb)=true ∧ ssl_7 (ssts_1 (s))=n)
     ∨ ∀ stammsatz s1: ist_enth_lsb (s1,l)=true
         ⇒ ( ssl_7 (ssts_1 (s1))=n ∧ ist_in_baum_sb (s1)=true)
  ∧ ¬ ∃ stammsatz s2: ist_in_baum_sb (s2,sb)=true
        ∧ ssl_7 (ssts_1 (s2))=n
        ∧ ist_enth_lsb (s2,l)=false

/*
blocknr_zu_stmtnr (n,sb) liefert die Blocknummer eines Stammsatzes mit
Statementnummer n.
*/
∀ nat n: ∀ sb sb: ∃ nat n1:
  n1=blocknr_zu_stmtnr (n,sb)
  ∧ (n1≠error
    ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ ssl_7 (ssts_1 (s))=n)
/*
```



```
v_qual_sts_proc_beg_zu_blocknr (n,sb) liefert den Stammsatz im SB sb,
der die Blocknummer n und Satzart "procedure" oder "begin block" hat.
*/
∀ nat n: ∀ sb sb: ∃ stammsatz s:
  s=v_qual_sts_proc_beg_zu_blocknr
  ∧ (s≠error
   ⇒ ((ssl_6 (ssts_1 (s))=procedure
       ∨ ssl_6 (ssts_1 (s))=begin block)
      ∧ ist_in_baum_sb (s,sb)=true
      ∧ sdt_10 (ssts_2 (s))=n)

/*
vq_sa (inh,sa,sb) liefert den Stammsatz im SB sb, der mit den ange-
gebenen Parametern Inhalt inh und Satzart sa übereinstimmt.
*/
∀ sb sb: ∀ charstring sa: ∀ name inh:
  ∃ stammsatz s:   s=vq_sa (inh,sa,sb)
  ∧ (s≠error
    ⇒ (ist_in_baum_sb (s,sb)=true
       ∧ sdt_3 (ssts_2 (s))=inh
       ∧ ssl_6 (ssts_1 (s))=sa))

PRIVATE OPS :
gen_l_stmtnr_1:nat l(stammsatz)-->l(stammsatz)
blocknr_zu_stmtnr_1:nat l(stammsatz)-->nat
vq_sts_proc_beg_zu_blocknr_1:nat l(stammsatz)-->stammsatz
vq_sa_1:charstring charstring l(stammsatz)-->stammsatz

DEFINE OPS :
enth_vater_proc (s,sb):=
   if eq.stammsatz (s,wurzel_sb (sb))
     then error.stammsatz
     else
      if eq.stammsatz (ssl_6 (ssts_1 (vater_sb (s,sb)),procedure)
        then vater_sb (s,sb)
        else enth_vater_proc (vater_sb (s,sb),sb)
      fi
   fi ;

enth_vater_proc_beg (s,sb):=
   if eq.stammsatz (s,wurzel_sb (sb)
     then error.stammsatz
     else
      if eq.stammsatz (ssl_6 (ssts_1 (vater_sb (s,sb)),procedure)
         or eq.stammsatz (ssl_6 (ssts_1 (vater_sb (s,sb)),
                                                 begin block))
        then vater_sb (s,sb)
        else enth_vater_proc_beg (vater_sb (s,sb),sb)
```

```
      fi
   fi ;

gen_l_stmtnr (n,sb):=
   gen_l_stmtnr_1 (n,preorder_sb (sb))

gen_l_stmtnr_1 (n,lsb):=
   case lsb is
   *nil_lsb-->nil_lsb
   *cons_lsb (s,rest)-->
      if eq.nat (ssl_7 (ssts_1 (s)),n))
        than cons_lsb (s,gen_l_stmtnr_1 (n,rest))
        else gen_l_stmtnr_1 (n,rest)
      fi
   esac ;

blocknr_zu_stmtnr (n,sb):=
   blocknr_zu_stmtnr_1 (n,preorder_sb (sb));

blocknr_zu_stmtnr_1 (n,lsb):=
   case lsb is
   *nil_lsb-->error.nat
   *cons_lsb (s,rest)-->
      if eq.nat (ssl_7 (ssts_1 (s)),n)
        then sdt_10 (ssts_2 (s))
        else blocknr_zu_stmtnr_1 (n,rest)
      fi
   esac ;

vq_sa (inh,sa,sb):=
   vq_sa_1 (inh,sa,preorder_sb (sb));

vq_sa_1 (inh,sa,lsb):=
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),sa)
         and eq.charstring (sdt_3 (ssts_2 (s)),inh)
        then s
        else vq_sa_1 (inh,sa,rest)
      fi
   esac ;

vq_sts_proc_beg_zu_blocknr (n,sb):=
   vq_sts_proc_beg_zu_blocknr_1 (n,preorder_sb (sb));

vq_sts_proc_beg_zu_blocknr_1 (n,lsb):=
   case lsb is
```



```
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),procedure)
          or eq.charstring (ssl_6 (ssts_1 (s)),begin block)
         and eq.nat (sdt_10 (ssts_2 (s)),n)
        then s
        else vq_sts_proc_beg_zu_blocknr_1 (n,rest)
      fi
   esac ;

ENDSPEC
```

3.1.4. Die Schnittstelle

In dem Modul Schnittstelle sind alle Datentypen zusammengefaßt, die Voraussetzung für die AUSWERTUNG sind. Diese Datentypen überbrücken den hier nicht spezifizierten Teil der AUFBEREITUNG und der ANALYSE. Insbesondere wird in diesem Modul dargestellt, welche Operationen notwendig sind, um die St und den SB eines durch den Namen qualifizierten Quellmoduls eines gegebenen Programmsytems zu erhalten.

Ein Programmsystem besteht aus einer Menge von Quelldateien $D_1,\ldots,D_i$. Jede dieser Quelldateien D_s enthält eine Menge von Quellmodulen $P_{s1},\ldots,P_{sx}$. Innerhalb einer Datei sind die Quellmodule eindeutig durch ihren Namen gekennzeichnet.

Die Spezifikation Schnittstelle benutzt die Spezifikation Analyse und die Spezifikation Dialog_System_Benutzer, und sie enthält weitere Operationen, die notwendig sind, um zu einem gegebenen Entrynamen, den zugehörigen Quellmodul zu bestimmen.

3.1.4.1. Spezifikation Analyse

Der Datentyp Datei wird als spezifiziert vorausgesetzt. Anschaulich enthält eine Datei eine Menge von Quellmodulen und alle notwendigen Operationen, um diese Quellmodule zu handhaben.

```
INSTANTITE Liste to L(Quelldatei)
ACTUALIZE : #x by Datei
          SORTS : #x.elem by datei
RENAME : SORTS : liste by l(dat)
         OPS : nil by nil_ld
END


SSPEC ANALYSE

USE SSPECS : Symboltabelle, Strukturbaum
             L(Name), L(Quelldatei)

PUBLIC OPS :
lies_sb:name datei-->sb
lies_sb:name datei-->st
name_quelldatei:datei-->name
inhalt_quelldatei:datei-->l(name)
lies_dat:name l(dat)-->datei
```



```
PROPERTIES :
/*
lies_sb (n,d) lefert den SB des durch den Namen n eindeutig gekennzeichneten Quellmoduls der Datei d. Diese Operation umfaßt das Einlesen des Quellmoduls von der Datei und die Transformation in den SB.
*/

/*
lies_st (n,d) liefert analog zu lies_sb (n,d) die ST zum Quellmodul.
*/

/*
name_quelldatei (d) liefert zur Datei d den zugehörigen Dateinamen.
*/

/*
inhalt_quelldatei (d) liefert die Liste der Namen sämtlicher Quellmodule, die sich auf der Datei d befinden.
*/

/*
lies_dat (n,ld) liefert die Datei mit Namen n aus der Liste ld der Quelldateien.
*/

ENDSPEC
```

3.1.4.2. Spezifikation Dialog_System_Benutzer

Dieser Datentyp dient der Darstellung des Dialogs System Benutzer.

In der AUSWERTUNG werden die Quellmodule durch Namen identifiziert. Allerdings ist diese Zuordnung nicht immer eindeutig. Im konkretem Ablauf von INTAKT wird an nicht eindeutigen Stellen ein Dialog mit dem Benutzer am Bildschirm geführt, um den Quellmodul eindeutig festzulegen. Dieser Dialog wird durch diese Spezifikation simuliert.

In dieser Spezifikation wird der Begriff "Secondary Entry" verwandt: Jeder Quellmodul und jeder Prozedur kann neben der eigentlichen Einsprungstelle noch weitere Einsprungstellen besitzen, die als Secondary Entries bezeichnet werden. Secondary Entries werden durch das Entry Statement deklariert.

```
Beispiel:   P : PROCEDURE;
                ...
```

```
        E : ENTRY;
            ...
        END P;
```

Diese Prozedur kann über den eigentlichen Entry P und den Secondary Entry E aufgerufen werden.

```
INSTANTIATE Tupel to Name_Dat
ACTUALIZE : #x1 by Name
            #x2 by Datei
            SORTS : #x1.elem by name
                    #x2.elem.datei
RENAME : SORTS : tupel by name_dat
         OPS : erz_tup by erz_nd
               s_1 by snd_1
END


INSTANTIATE Liste to L(Name_Dat)
ACTUALIZE : #x by Name_Dat
            SORTS : #x.elem by name_dat
RENAME : SORTS : liste by l(name_dat)
         OPS : nil by nil_lnd
END


SSPEC DIALOG_SYTEM_BENUTZER

USE SSPECS : L(Name), L(Name_Dat), L(Quelldatei), Analyse

PUBLIC OPS :
dia_benutzer_modul:name l(dat)-->dat
dia_benutzer_entry:name name l(name_dat)-->name_dat


PROPERTIES :
/*
dia_benutzer_modul (n,ld)
Modulnamen innerhalb einer Quelldatei sind eindeutig. Dies gilt jedoch
nicht für die Modulnamen zweier Quelldateien. Ist also ein Modul zu
analysieren, dessen Quelldatei nicht eindeutig ist, muß der Benutzer
aufgrund des Modulnamens und der Liste der Quelldateien, die ein Modul
mit dem entsprechendem Namen enthalten, entscheiden, welche Quelldatei
zu verwenden ist.
Diese Operation liefert also zu dem Modulnamen n aus der Liste der
```



```
Quelldateien ld eine Datei.
*/
V l(dat) ld: V name n: ∃ quelldatei d:
  d=dia_benutzer_modul (n,ld)
  ∧ ist_enth_ld (d,ld)=true
  ∧ ist_enth_ln (n,inhalt_quelldatei (d))=true

/*
dia_benutzer_entry (m,e,lnd)
Beim Aufruf externer Prozeduren ist nicht immer eindeutig, um welchen
Entry es sich handelt. Es können mehrere Prozeduren mit gleichem Namen
auf den Quelldateien existieren, oder es können mehrere Prozeduren mit
einem Sec. Entry gleichen Namens existieren. Aus diesem Grund sind in
einem solchen Fall, alle Informationen zu sammeln, die notwendig sind,
damit der Benutzer entscheiden kann, welcher Entry tatsächlich gemeint
ist.
Diese Information besteht aus:
1. dem Namen des Moduls, aus dem der Aufruf stammt,
2. dem Namen des aufgerufenen Entry,
3. der Liste, die sämtliche Modulnamen, qualifiziert mit der zugehöri-
   gen Quelldatei, enthält, in denen ein Entry mit entsprechendem Na-
   men vorkommt. Ist der Modulname nicht gleich dem Entrynamen, so
   existiert in diesem Modul ein Sec. Entry mit entsprechendem Namen.
   Diese Liste der nicht eindeutigen Entries wird durch die Operation
   gen_l_nicht_eind_entries (n,ld) der SSPEC Schnittstelle erzeugt.
*/
V name m,e: V l(name_dat) lnd: ∃ name_dat nd:
  nd=dia_benutzer_entry (m,e,lnd)
  ∧ (snd_1 (nd)=e v ∃ name m': snd_1 (nd)=m')
  ∧ ist_enth_lnd (nd,lnd)=true

ENDSPEC
```

3.1.4.3. Spezifikation der Schnittstelle

SSPEC SCHNITTSTELLE

USE SSPECS : Analyse, Dialog_System_Benutzer

PUBLIC OPS :
liefere_eind_modul:name l(dat)-->datei
ist_mehrdeutig:name l(dat)-->bool
gen_l_n_eind_entries:name l(dat)-->l(name_dat)
ist_sec_entry_in_modul:name sb-->bool
sec_entries_in_modul:sb-->l(name)
liefere_eind_entry:name name l(dat)-->name_dat
dateien_mit_id_modulnamen:name l(dat)-->l(dat)

PROPERTIES :
/*
ist_mehrdeutig (m,ld) liefert true, falls es mindestens zwei Quelldateien in der Liste der Quelldateien ld gibt, die einen Modul mit Namen m enthalten.
*/
∀ name m: ∀ l(dat) ld:
  ist_mehrdeutig (m,ld)=false
  ∨ ∃ dat d1,d2: ist_enth_ld (d1,ld)=true
      ∧ ist_enth_ld (d2,ld)=true
      ∧ ist_enth_ln (m,inhalt_quelldatei (d1))=true
      ∧ ist_enth_ln (m,inhalt_quelldatei (d2))=true
      ∧ d1≠d2

/*
bestimme_dat (m,ld) liefert aus der Liste der Quelldateien ld, die erste Quelldatei, die ein Modul mit Namen m enthält. Diese Operation wird nur dann aufgerufen, wenn es nur eine Datei mit einem Quellmodul m gibt.
*/
∀ l(dat) ld: ∀ name m: ∃ dat d:
  d=bestimme_dat (m,ld)
  ∧ (d≠error
    ⇒ (ist_enth_ld (d,ld)=true
      ∧ ist_enth_ln (m,inhalt_quelldatei (d))=true))

/*
liefere_eind_modul (m,ld) liefert, sofern von den Quelldateien nur eine Datei einen Modul m enthält, diese Datei, sofern mehrere Dateien existieren, die Datei, die der Benutzer ausgewertet wünscht.
*/
∀ name m: ∀ l(dat) ld: ∃ dat d:
  d=liefere_eind_modul (m,ld)

  ∧ (d≠error
    ⇒ (((ist_mehrdeutig (m,ld)=false ∧ d=bestimme_dat (m,ld))
      ∨ d=dia_benutzer_modul (m,dateien_mit_id_modulnamen))
      ∧ ist_enth_ld (d,ld)=true
      ∧ ist_enth_ln (m,inhalt_quelldatei (d))=true))

/*
gen_l_n_eind_entries (e,ld) lefert zum Entrynamen e alle Quellmodulnamen qualifiziert mit der zugehörigen Datei, die einen Entry e bzw. Sec. Entry e besitzen.
*/
∀ name e: ∀ l(dat) ld: ∃ l(name_dat) l:
  l=gen_l_n_eind_entries (e,ld)
  ∧ (ist_leer_lnd (l)=false
    ⇒ ∀ name_dat nd: ist_enth_lnd (nd,l)=true
        ⇒ ((snd_1 (nd)=e
            ∨ ist_sec_entry_in_modul (snd_1 (nd),
                lies_sb (snd_1 (nd),snd_2 (nd)))=true)
          ∧ ¬ ∃ name_dat nd': ist_enth_lnd (nd',l)=false
              ∧ (snd_1 (nd')=e
                ∨ ist_sec_entry_in_modul (snd_1 (nd'),
                    lies_sb (snd_1 (nd'),snd_2 (nd')))=true)))

/*
liefere_eind_entry (m,e,ld) liefert ein Tupel, das aus dem Modulnamen und der Quelldatei, die diesen Modul enthält, besteht. Der Modul besitzt einen Entry e oder einen Sec. Entry e. m ist der Modulname des Moduls, das den Aufruf des externen Entry e enthält.
*/
∀ name m,e: ∀ l(dat) ld: ∃ name_dat nd:
  nd=liefere_eind_entry (m,e,ld)
  ∧ (nd≠error
    ⇒ ((anzahl_lnd (gen_l_nicht_eind_entries (n,ld))>1
        ∧ nd=dia_benutzer_entry (m,e,
            gen_l_n_eind_entries (e,ld)))
      ∨ nd=car_lnd (gen_l_n_eind_entries (e,ld)))

/*
dateien_mit_id_modulnamen (m,ld) liefert alle Quelldateien, die ein Modul mit Namen m enthalten. Diese Operation wird nur aufgerufen, wenn es tatsächlich mehrere Dateien mit gleichem Modul gibt.
*/
∀ l(dat) ld: ∀ name m: ∃ l(dat) l:
  l=dateien_mit_id_modulnamen (m,ld)
  ∧ ((ist_leer_ld (l)=true
    ∧ ¬ ∃ dat d:
        ist_enth_ld (d,ld)=true
        ∧ ist_enth_ln (m,inhalt_quelldatei (d))=true)

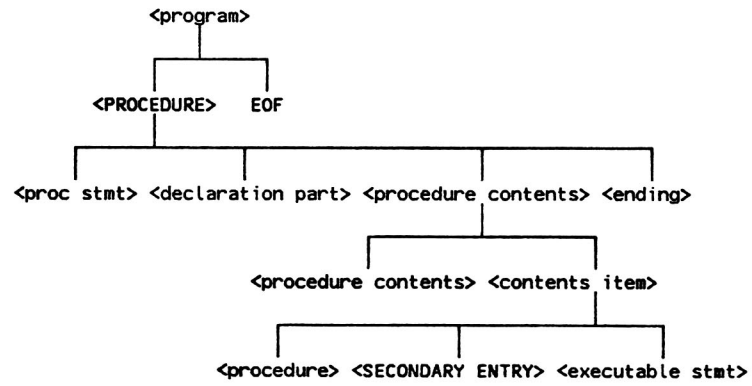
```
    ∨ ∀ dat d': ist_enth_ld (d',l)=true
        ⇒ (ist_enth_ln (m,inhalt_quelldatei (d))=true
          ∧ ¬ ∃ dat d'': ist_enth_ld (d'',ld)=true
                ∧ ist_enth_ln (m,inhalt_quelldatei (d''))=true
                ∧ ist_enth_ld (d'',l)=false))

/*
ist_sec_entry_in_modul (e,sb) liefert true, falls e ein Sec. Entry des
durch den SB sb gekennzeichneten Quellmoduls ist.
*/
∀ name e: ∀ sb sb:
  ist_sec_entry_in_modul (e,sb)=true
  ⇒ ist_enth_ln (e,sec_entries _in_modul (sb))=true

/*
sec_entries_in_modul (sb) liefert die Liste der Namen aller Sec.  Ent-
ries der äußersten Prozedur im durch sb gegebenen Quellmodul.

Für jeden Sec. Entry gilt, daß im SB ein Stammsatz mit Satzart
"Secondary Entry" oder "Entry Statement" existiert.  Der Name des Sec.
Entry ist im Feld Inhalt des Stammsatzes enthalten. Für alle Sec. Ent-
ries der äußersten Prozedur gilt zusätzlich, daß sie einen Vorgänger
mit Satzart "Procedure" besitzen, dessen Vater die Wurzel des SB ist.

Beispiel eines solchen SB, repräsentiert durch die Komponente Satzart:
```




```
*/
∀ sb sb: ∃ l (name) ln:
  ln=sec_entries_in_modul (sb)
  ∧ ∀ name n: ist_enth_ln (n,ln)=true
```



```
      ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ (ssl_6 (ssts_1 (s))=secondary entry
            ∨ ssl_6 (ssts_1 (s))=entry stmt)
          ∧ sdt_3 (ssts_2 (s))=n
          ∧ vater_sb (enth_vater_proc (s,sb),sb)=wurzel (sb)
          ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
                ∧ (ssl_6 (ssts_1 (s))=secondary entry
                  ∨ ssl_6 (ssts_1 (s))=entry stmt)
                ∧ vater_sb (enth_vater_proc (s',sb),sb)
                                              =wurzel (sb)
                ∧ ist_enth_ln (sdt_3 (ssts_2 (s')),ln)=false

PRIVATE OPS :
bestimme_dat:name l(dat)-->datei
ist_zweideutig:name l(dat)-->bool
gen_l_n_eind_entries_1:name l(name) datei-->l(name_dat)
gen_l_n_eind_entries_2:name name datei-->l(name_dat)
sec_entries_in_modul_1:l(stammsatz) sb-->l(name)

DEFINE OPS :
liefere_eind_modul (n,ldat):=
   if ist_mehrdeutig (n,ldat)
     then dia_benutzer_modul (n,dateien_mit_id_modulnamen (n,ldat))
     else bestimme_dat (n,ldat)
   fi ;

ist_mehrdeutig (n,ldat):=
   case ldat is
   *nil_ld-->false
   *cons_ld (d,rest)--> if ist_enth_ln (n,inhalt_quelldatei (d))
                         then ist_zweideutig (n,rest)
                         else ist_mehrdeutig (n,rest)
                       fi
   esac ;

ist_zweideutig (n,ldat):=
   case ldat is
   *nil_ld-->false
   *cons_ld (d,rest)--> if ist_enth_ln (n,inhalt_quelldatei (d))
                         then true
                         else ist_zweideutig (n,rest)
                       fi
   esac ;

bestimme_dat (n,ldat):=
   case ldat is
   *nil_ld-->error.datei
   *cons_ld (d,rest)--> if ist_enth_ld (n,inhalt_quelldatei (d))
```

```
                    then d
                    else bestimme_dat (n,rest)
                   fi
   esac ;

gen_l_n_eind_entries (e,ldat):=
   case ldat is
   *nil_ld-->nil_lnd
   *cons_ld (d,rest)-->append_lnd (gen_l_n_eind_entries_1 (
                                        e,inhalt_quelldatei (d),d),
                                gen_l_n_eind_entries (e,rest))
   esac ;

liefere_eind_entry (m,n,ldat):=
   let x=gen_l_n_eind_entries (n,ldat) in
    if ist_gt (anzahl_lnd (x,succ (0)))
      then dia_benutzer_entry (m,n,x)
      else if ist_leer_lnd (x)
             then error.name_dat
             else car_lnd (x)
           fi
    fi ;

dateien_mit_id_modulnamen (n,ld):=
   case ldat 'i
   *nil_ld-->nil_ld
   *cons_ld (d,rest)-->
      if ist_enth_ln (n,inhalt_quelldatei (d))
        then cons_ld (d,dateien_mit_id_modulnamen (n,rest))
        else dateien_mit_id_modulnamen (n,rest)
      fi
   esac ;

gen_l_n_eind_entries_1 (n,ln,d):=
   case ln is
   *nil_ln-->nil_lnd
   *cons_ln (l,rest)-->
      if eq.name (l,n)
        then cons_lnd (erz_nd (l,d),nil_lnd)
        else append_lnd (gen_l_n_eind_entries_2 (n,l,d),
                         gen_l_n_eind_entries_1 (n,rest,d))
      fi
   esac ;

gen_l_n_eind_entries_2 (e,m,d):=
   if ist_sec_entry_in_modul (n,lies_sb (m,d))
     then cons_lnd (erz_nd (m,d),nil_lnd)
     else nil_lnd
```



```
   fi ;

ist_sec_entry_in_modul (n,sb):=
   ist_enth_ln (n,sec_entries_in_modul (sb));

sec_entries_in_modul (sb):=
   sec_entries_in_modul_1 (preorder_sb (sb),sb);

sec_entries_in_modul_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_ln
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
          or eq.charstring (ssl_6 (ssts_1 (s)),entry stmt))
         and eq.charstring (vater_sb (enth_vater_proc (s,sb),sb),
                                                  wurzel_sb (sb))
        then cons_ln (sdt_3 (ssts_2 (s)),
                      sec_entries_in_modul_1 (rest,sb))
        else sec_entries_in_modul_1 (rest,sb)
    fi
   esac ;

ENDSPEC
```

3.2. Spezifikation der Auswertefunktionen

3.2.1. Vollständigkeit der Komponenten

Diese Auswertefunktion ist in vier Teilfunktionen gegliedert. Die Spezifikation Vollständigkeit der Komponenten ist eine einfache Zusammenfassung dieser Teilfunktionen. Die Idee für diese Auswertefunktionen liegt darin, daß in der Praxis ausgelieferte Programmsysteme oft nicht ablauffähig sind, da im Programmsystem Module fehlen.

3.2.1.1. Vollständigkeit der Entries

3.2.1.1.1. Beschreibung

Ausgehend von einer Menge von Quellprogrammen ist zu überprüfen, welche externen Entries bzw. externen Secondary Entries in diesen Quellprogrammen aufgerufen werden und ob die zugehörigen Quellmodule überhaupt im Programmsystem, d.h. auf den gegebenen Quelldateien (L(Quelldatei)) vorliegen. Mit den aufgerufenen Quellprogrammen ist analog zu verfahren.

Die Ausgabe der Funktion besteht in der Liste der aufgerufenen, aber nicht vorhandenen externen Entries.

Beispiel: Sei A das folgende auszuwertende Quellmodul:

```
A : PROCEDURE OPTIONS (Main);
    DECLARE E ENRTY (INTEGER) EXTERNAL;
    ...
    CALL E(3);
    ...
END A;
```

Die Quelldateien sind also nach einem Quellmodul E bzw. nach einem Quellmodul mit Secondary Entry E zu durchsuchen. Im Falle, daß kein zugehöriger Entry gefunden wird, ist der Name E auszugeben.

Die Aufrufe von externen Entries sind in der Symboltabelle durch Einträge mit dem Attribut "EXTCALL" für externe Prozeduren bzw. "EXTFCTCALL" für externe Funktionen gekennzeichnet. Im Namensfeld des Eintrags befindet sich der Name des Entry.

3.2.1.1.2. Spezifikation Vollständigkeit der Entries

SSPEC VOLLSTÄNDIGKEIT DER ENTRIES

USE SSPECS : Schnittstelle, L(Name), L(Name_Dat), L(Quelldatei)

PUBLIC OPS :
op_vollst_d_entries: l(name) l(dat)—>l(name)
gen_l_name_dat: l(name) l(dat)—>l(name_dat)
gen_l_aufr_ext_entries:name_dat—>l(name)

PROPERTIES :

```
/*
gen_l_name_dat  (ln,ld) liefert die Liste,  in der zu jedem Modulnamen
aus ln, die Datei, die den Modul enthält, bestimmt ist.
*/
∀ l(name) ln: ∀ l(dat) ld: ∃ l(name_dat) l:
  l=gen_l_name_dat (ln,ld)
  ∧ (l≠error
     ⇒ ∀ name_dat nd: ist_enth_lnd (nd,l)=true
         ⇒ (snd_2 (nd)=liefere-eind_modul (snd_1 (nd),ld)
           ∧ ist_enth_ln (snd_1 (nd),ln)=true))

/*
gen_l_aufr_ext_entries  (nd) liefert die Liste der aufgerufenen externen Entries,  in dem durch das Tupel Modulname_Datei nd eindeutig gekennzeichnetem Modul.
*/
∀ name_dat nd: ∃ l(name)l:
  l=gen_l_aufr_ext_entries (nd)
  ∧ (ist_leer_ln (l)=false
    ⇒ ∀ name n: ist_enth_ln (n,l)=true
        ⇒ ∃ knoten_st kst:
            ist_in_baum_st (kst,
               lies_st (snd_1 (nd),snd_2 (nd))=true
            ∧ (∃ gebiet gb: ∃ nat n1:
                 kst=erz_kst (blck_g (gb),n1)
               ∧ ∃ eintrag e: ist_enth_gb (e,gb)=true
                   ∧ (ist_enth_attr (extcall,setr_2 (e))=true
                     ∨ ist_enth_attr (extfctcall,setr_2 (e))=true)
              ∧ setr_1 (e)=n)

/*
op_vollst_d_entries (ln,ld) liefert ausgehend von der Liste der Modulnamen (ln) und der Liste der Quelldateien ld,  die Liste der Namen der aufgerufenen, aber nicht vorhandenen externen Entries.
*/
∀ l(name) ln: ∀ l(dat) ld: ∃ l(name) l:
```

```
l=op_vollst_d_entries (ln,ld)
∧ ∀ name n: ist_enth_ln (n,l)=true
    ⇒ (ist_leer_lnd (gen_l_nicht_eind_entries (n,ld))=true
      ∧ ∃ l(name_dat) lnd: ∃ nat r: r=anzahl_lnd (lnd)
          ∧ ∀ nat i: 1<i≤r
             ⇒ (ist_enth_ln (snd_1 (i-tes_element_lnd (lnd,i)),
                    gen_l_aufruf_ext_entries (
                       i-tes_element_lnd (lnd,i-1)))=true
               ∨ (∃ name e:
                    ist_sec_entry_in_modul (e,
                      lies_sb (snd_1 (i-tes_element_lnd (lnd,i)),
                               snd_2 (i-tes_element_lnd (lnd,i))))
                                                          =true
                    ∧ ist_enth_ln (e,
                        gen_l_aufr_ext_entries (
                          i-tes_element (lnd,i-1)))=true)
          ∧ ist_enth_lnd (car_lnd (lnd),
                          gen_l_name_dat (ln,ld))=true
          ∧ ist_enth_ln (n,
              gen_l_aufr_ext_entries (i-tes_element_lnd (lnd,r)))
```

```
/*
op_vollst_d_entries_1 (lnd,ld)
```
Diese Operation wird mit der Liste, die zu jedem Modulnamen die zugehörige Datei enthält, aufgerufen. Die Operationsdefinition ist eine Aufruffolge der Operationen lösche_doppelte_elem_ln und op_vollst_d_entries_2. lösche_doppelte_elem_ln bewirkt, daß in der durch die Operation op_vollst_d_entries_2 gelieferten Liste jeder Name genau einmal vorkommt.

Beispiel: In Modul A und in Modul B werde der Entry E aufgerufen. E existiere nicht als Entry. Ohne Löschen der doppelten Namen, würde E zweimal in der Ergebnisliste auftauchen.
```
*/

/*
op_vollst_d_entries_2 (lnd1,lnd2,ld)
```
lnd1 ist die Liste der auszuwertenden Quellmodule. lnd2 enthält alle Quellmodule, die bereits abgearbeitet wurden. Diese Liste wird benötigt, damit nicht in einem Zyklus immer wieder die gleichen Module ausgewertet werden. Diese Operation bearbeitet die Liste lnd1 und ruft die Operation op_vollst_d_entries_3 mit dem ersten Element von lnd1 auf, sofern lnd1 nicht leer ist.
```
*/

/*
op_vollst_d_entries_3 (m,lentry,lnd1,lnd2,ld)
```
Die Operation wird mit den folgenden Parametern aufgerufen:
```





```
dem Modulnamen m, der Liste der Namen, der im Modul m aufgerufenen externen Entries lentry (lenty enthält keine doppelten Namen), der Liste der abzuarbeitenden Quellmodule lnd1 und der Liste der bereits ausgewerteten Quellmodule lnd2.

Die Operation arbeitet die Liste lentry sequentiell ab.
   Bei leerer Liste wird die Operation op_vollst_d_entries mit lnd1, lnd2 und ld aufgerufen, die den nächsten Modul aus lnd1 bearbeitet.
   Bei nicht leerer liste wird mittels Operationen der Schnittstelle entschieden, ob es einen Entry zum Aufruf gibt, bzw. welcher Entry gemeint ist, falls es mehrere gibt.
   Gibt es keinen passenden Entry, dann wird der Entryname an die Liste der nicht vorhanden Entries angehangen und die Entryliste wird weiter abgearbeitet.
   Gibt es einen passenden Entry, dann ist zu prüfen, ob der Modul mit diesem Entry bereits ausgewertet wurde, ob der Modul bereits in lnd2 enthalten ist. Falls ja, wird die Entryliste weiter abgearbeitet. Falls nein, wird der Modul samt zugehöriger Datei an die abzuarbeitende Modulliste lnd1 und an die abgearbeitete Modulliste lnd2 angehangen und das nächste Element der Entryliste wird unter Berücksichtigung der geänderten Parameter verarbeitet.
   Gibt es mehrere mögliche Entries, so liefern die Operationen der Schnittstelle den auszuwertenden Modul, und es gibt somit einen passenden Entry (s.o.).
*/

PRIVATE OPS :
op_vollst_d_entries_1:L(name_dat) l(dat)-->l(name)
op_vollst_d_entries_2:l(name_dat) l(name_dat) l(dat)-->l(name)
op_vollst_d_entries_3:
        name l(name) l(name_dat) l(name_dat) l(dat)-->l(name)
gen_l_aufr_ext_entries_1:l(knoten_st)-->l(name)
gen_l_aufr_ext_entries_2:symtab-->l(name)
gen_l_aufr_ext_entries_3:gebiet-->l(name)
gen_l_aufr_ext_entries_4:eintrag-->l(name)

DEFINE OPS :
op_vollst_d_entries (ln,ld):=
   gen_l_aufr_ext_entries_1 (gen_l_name_dat (ln,ld),ld);

op_vollst_d_entries_1 (lnd,ld):=
   lösche_doppelte_elem_ln (op_vollst_d_entries_2 (lnd,lnd,ld));

op_vollst_d_entries_2 (lna,lnb,ld):=
   case lna is
   *nil_lnd-->nil_ln
   *cons_lnd (n,rest)-->
      op_vollst_d_entries_3 (snd_1 (n),
```
```

```
        lösche_doppelte_elem (gen_l_aufr_ext_entries (n)),
                                              rest,lnb,ld)
   esac ;

op_vollst_d_entries_3 (m,le,lna,lnb,ld):=
   case le is
   *nil_ln-->op_vollst_d_entries_2 (lna,lnb,ld)
   *cons_ln (e,rest)-->
     let x=gen_l_n_eind_entries (e,ld) in
      if eq.nat (anzahl_lnd (x),succ (0))
        then cons_ln (e,op_vollst_d_entries_3 (m,rest,lna,lnb,ld))
        else if eq.nat (anzahl_lnd (x),succ (0))
               then if ist_enth_lnd (car_lnd (x),lnb)
                      then op_vollst_d_entries_3 (m,rest,lna,lnb,ld)
                      else op_vollst_d_entries_3 (m,rest,
                             append_lnd (lna,x),append_lnd (lnb,x))
                    fi
               else if ist_gt (anzahl_lnd (x),succ (0))
                      then if ist_enth_lnd (
                                  dia_benutzer_entry (m,e,x),lnb)
                             then op_vollst_d_entries_3 (
                                            m,rest,lna,lnb,ld)
                             else op_vollst_d_entries_3 (m,rest,
      append_lnd (lna,cons_lnd (dia_benutzer_entry (m,e,x),nil_lnd)),
      append_lnd                  (lnb,cons_lnd                  (
                         dia_benutzer_entry (m,e,x),nil_lnd)),ld)
                           fi
                      else error.l (name)
                    fi
             fi
      fi
   esac ;

gen_l_name_dat (ln,ld):=
   case ln is
   *nil_ln-->nil_lnd
   cons_ln (n,rest)-->cons_lnd (erz_nd (n,liefere_eind_modul (n,ld)),
                                gen_l_name_dat (rest,ld))
   esac ;

gen_l_aufr_ext_entries (nd):=
   gen_l_aufr_ext_entries_1 (
     preorder_st (lies_st (snd_1 (nd),snd_2 (nd))));

gen_l_aufr_ext_entries_1 (lst):=
   case lst is
   *nil_lst-->nil_ln
   *cons_lst (k,rest)-->
```



```
     -->append_ln (gen_l_aufr_ext_entries_2 (skst_1 (k)),
                   gen_l_aufr_ext_entries_1 (rest))
   esac ;

gen_l_aufr_ext_entries_2 (s):=
   case s is
   *blck_b (b)-->nil_ln
   *blck_g (g)-->gen_l_aufr_ext_entries_3 (g)
   esac ;

gen_l_aufr_ext_entries_3 (g):=
   case g is
   *nil_gb-->nil_ln
   *cons_gb (e,rest)-->append_ln (gen_l_aufr_ext_entries_4 (e),
                                  gen_l_aufr_ext_entries_3 (rest))
   esac ;

gen_l_aufr_ext_entries_4 (eintr):=
   if ist_enth_attr (extcall,setr_2 (eintr))
      or ist_enth_attr (extfctcall (setr_2 (eintr)))
     then cons_ln (setr_1 (eintr),nil_ln)
     else nil_ln
   fi ;

ENDSPEC
```

3.2.1.2. Vollständigkeit der Externverweise

3.2.1.2.1. Beschreibung

Ausgehend von einer Menge von Quellprogrammen wird überprüft, ob externe Entries in den Quellmodulen deklariert werden und ob diese auch aufgerufen werden. Für jeden Quellmodul wird also die Liste der deklarierten aber nicht aufgerufenen exteren Entries, einschließlich Statementnummer der Deklaration ermittelt und qualifiziert mit dem Quellmodulnamen ausgegeben.

Beispiel: Sie B das folgende Quellmodul

```
B : PROCEDURE;
    DECLARE E ENTRY (BIT);
    ...
END B;
```

Da E als externer Entry in B dekariert ist, aber nicht aufgerufen wird, wird die Liste (B_(E_2)), wenn 2 die Statementnummer der Deklaration ist, ausgegeben.

Externe Entries werden durch ein Declare-Statement mit Schlüsselwort Entry deklariert. Ein solches Declare-Statement darf nicht das Variable Attribut enthalten. Das Scope Attribut "external" ist optional.

3.2.1.2.2. Spezifikation Vollständigkeit der Externverweise

```
INSTANTIATE Tupel to Name_Nat
ACTUALIZE : #x1 by Name
            #x2 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by nat
RENAME : SORTS : tupel by name_nat
           OPS : erz_tup by erz_ni
END
```

```
INSTANTIATE Liste to L(Name_Nat)
ACTUALIZE : #x by Name_Nat
            SORTS : #x.elem by name_nat
RENAME : SORTS : liste by l(name_nat)
           OPS : nil by nil_lni
END
```



```
INSTANTIATE Tupel to Name_L(Name_Nat)
ACTUALIZE : #x1 by Name
            #x2 by L(Name_Nat)
            SORTS : #x1.elem by name
                    #x2.elem by l(name_nat)
RENAME : SORTS : tupel by name_l(name_nat)
           OPS : erz_tup by erz_nlni
END
```

```
INSTANTIATE Liste to L(Name_L(Name_Nat))
ACTUALIZE : #x by Name_L(Name_Nat)
            SORTS : #x.elem by name_l(name_nat)
RENAME : SORTS : liste by l(name_l(name_nat))
           OPS : nil by nil_lnlni
END
```

```
SSPEC VOLLSTÄNDIGKEIT DER EXTERNVERWEISE

USE SSPECS : Schnittstelle, L(Name), L(Name_L(Name_Nat)), L(Nat)

PUBLIC OPS :
op_vollst_externverweise:l(name) l(dat)-->l(name_l(name_nat))
ist_aufruf_vorh:name_nat sb-->bool
gen_l_dcl_ext_entries:sb-->l(stammsatz)
gen_l_sts_sa_dcl_part:l(stammsatz)-->l(stammsatz)
gen_l_stmtnr_aller_entry_dcl:l(stammsatz) sb-->l(nat)
gen_l_entrynamen:l(nat) l(stammsatz)-->l(name_nat)
lösche_entry_variable:L(name_nat) sb-->l(name_nat)

PROPERTIES :
/*
gen_l_sts_sa_dcl_part  (lsb) liefert alle Stammsätze aus der Liste von
Stammsätzen lsb, die die Satzart "declaration part" haben.
*/
∀ l(stammsatz) lsb: ∃ l(stammsatz) l:
  l=gen_l_sts_sa_dcl_part (lsb)
  ∧ (ist_leer_lsb (l)=false
     ⇒ (∀ stammsatz s: ist_enth_lsb (s,l)=true
          ⇒ ssl_6 (ssts_1 (s))=declaration part))
  ∧ ¬ ∃ stammsatz s': ist_enth_lsb (s',lsb)=true
        ∧ ssl_6 (ssts_1 (s))=declaration part
        ∧ ist_ent_lsb (s',l)=false

/*
```

op_vollst_externverweise (ln,ld) liefert zu jedem Quellmodul aus der Liste der Quellmodulnamen ln die Liste der deklarierten externen Entries (Name und Statementnummer der Deklaration), die nicht aufgerufen werden.
*/
```
V l(name) ln: V l(dat) ld:
  ∃ l(name_l(name_nat)) l:
   l=op_vollst_externverweise (ln,ld)
   ∧ V name_l(name_nat) nt:
       ist_enth_lnlni (nt,l)=true
       ⇒ (ist_enth_ln (snlni_1 (nt),ln)=true
          ∧ V name_nat n:
             ist_enth_lni (n,snlni_2 (nt))=true
             ⇒ (ist_enth_lni (n, gen_l_dcl_ext_entries (
                   lies_sb (snlni_1 (nt),
                     liefere_eind_modul (snlni_1 (nt),ld))))=true
               ∧ ist_aufruf_vorhanden (n,lies_sb (
                   snlni_1 (nt),liefere_eind_modul (
                     snlni_1 (nt),ld)))=false)
```

/*
gen_l_dcl_ext_entries (sb) liefert die Liste aller im SB sb vorhandenen Deklarationen von externen Entries in Form von Paaren bestehend aus dem Entrynamen Und der Statementnummer der Deklaration.
Die folgenden Hilfsfunktionen werden benötigt:
gen_l_sts_sa_dcl_part (lsb) liefert alle Stammsätze mit Satzart "declaration part", die in lsb enthalten sind.
gen_l_stmtnr_aller_entry_dcl (lsb,sb) liefert die Liste aller Stmtnummern aller Entrydeklarationen. lsb enthält nur die Stammsätze mit Satzart "declaration part".
gen_l_entrynamen (lnat,lsb) erzeugt zu jeder Stmtnummer aus lnat das Paar (Stmtnr, Name des zugehörigen Entry).
lösche_entry_variable (lns,sb) löscht aus dem Ergebnis der Operation gen_l_entrynamen (lnat,lsb) alle Paare, die nicht einer Deklaration eines externen Entry entsprechen, sondern einer Entry-Variablen.
*/
```
V sb sb: ∃ l(name_nat) lns:
  lns=gen_l_dcl_ext_entries (sb)
  ∧ V name_nat ns: ist_enth_lni (ns,lns)=false
      ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ ssl_7 (ssts_1 (s))=sni_2 (ns)
          ∧ ssl_6 (ssts_1 (s))=entry
          ∧ ¬ (∃ stammsatz s1: ist_in_baum_sb (s1,sb)=true
                ∧ ssl_7 (ssts_1 (s1))=sni_2 (ns)
                ∧ ssl_6 (ssts_1 (s1))=variable)
          ∧ (∃ stammsatz s2: ist_in_baum_sb (s2,sb)=true
             ∧ ssl_7 (ssts_1 (s2))=sni_2 (ns)
             ∧ sdt_3 (ssts_2 (s2))=sni_1 (ns))
```



```
          ∧ ¬ (∃ stammsatz s': ist_in_baum_sb (s',sb)=true
                ∧ ssl_7 (ssts_1 (s'))=sni_2 (ns)
                ∧ (∃ stammsatz s3: ist_in_baum_sb (s3,sb)=true
                     ∧ ssl_6 (ssts_1 (s3))=declaration part
                     ∧ ist_in_baum_sb (s',t_baum_sb (s3,sb))=true
                     ∧ ss_6 (ssts_1 (s'))=entry
                     ∧ ¬ (∃ stammsatz s4:
                            ist_in_baum_sb (s4,sb)=true
                            ∧ ssl_7 (ssts_1 (s4))=
                              ssl_7 (ssts_1 (s'))
                            ∧ ssl_6 (ssts_1 (s4))=variable)))
```

/*
ist_aufruf_vorhanden (ns,sb) liefert true, falls es zum durch ns gekennzeichnetem Entry einen Aufruf gibt.
*/
```
V name_nat_ns: V sb sb:
  ist_aufruf_vorhanden (ns,sb)=true
  ⇒ (∃ stammsatz s: ist_in_baum_sb (s,sb)=true
       ∧ sdt_3 (ssts_2 (s))=sni_1 (ns)
       ∧ (ssl_6 (ssts_1 (s))=callext
          ∨ ssl_6 (ssts_1 (s))=callextfct)
```

PRIVATE OPS :
```
ist_aufruf_vorh_1:name_nat l(stammsatz)-->bool
op_vollst_externverweise_1:name l(dat)-->l(name_nat)
op_vollst_externverweise_2:l(name_nat) sb-->l(name_nat)
gen_l_entrynamen_1:nat l(stammsatz)-->l(name_nat)
lösche_entry_variable_1:name_nat l(stammsatz)-->l(name_nat)
gen_l_stmtnr_aller_entry_dcl_1:l(stammsatz)-->l(nat)
```

DEFINE OPS :
```
ist_aufruf_vorh (n,sb):=
   ist_aufruf_vorh_1 (n,preorder_sb (sb));

ist_aufruf_vorh_1 (n,l):=
   case l is
   *nil_lsb-->false
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),callext)
          or eq.charstring (ssl_6 (ssts_1 (s)),callextfct))
         and eq.charstring (sdt_3 (ssts_2 (s)),sni_1 (n))
       then true
       else ist_aufruf_vorh_1 (n,rest)
      fi
   esac ;

op_vollst_externverweis (ln,ld):=
```

```
   case ln is
   *nil_ln-->nil_lnlni
   *cons_ln (n,rest)-->
      cons_lnlni (erz_nlni (n,op_vollst_externverweise_1 (n,ld)),
                  op_vollst_externverweise (rest,ld))
   esac ;

op_vollst_externverweise_1 (n,ld):=
   op_vollst_externverweise_2 (
      gen_l_dcl_ext_entries (lies_sb (n,liefere_eind_modul (n,ld)),
      lies_sb (n,liefere_eind_modul (n,ld));

gen_l_dcl_ext_entries (sb):=
   lösche_entry_variable (gen_l_entrynamen (
    gen_l_stmtnr_aller_entry_dcl (gen_l_sts_sa_dcl_part (
       preorder_sb (sb),sb),preorder_sb (sb)),sb);

gen_l_sts_sa_dcl_part (lsb):=
   case lsb is
  *nil_lsb-->nil_lsb
  *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),declaration part)
        then cons_lsb (s,gen_l_sts_sa_dcl_part (rest))
        else gen_l_sts_sa_dcl_part (rest)
      fi ;

op_vollst_externverweise_2 (l,sb):=
   case l is
   *nil_lni-->nil_lni
   *cons_lni (l,rest)-->
      if ist_aufruf_vorh (l,rest)
        then op_vollst_externverweise_2 (rest,sb)
        else cons_lni (l,op_vollst_externverweise_2 (rest,sb))
      fi
   esac ;

gen_l_entrynamen (lnat,lsb):=
   case lnat is
   *nil_lnat-->nil_lni
   *cons_lnat (n,rest)-->
     append_lni (gen_l_entrynamen_1 (n,lsb),
                 gen_l_entrynamen (rest,lsb))
   esac ;

gen_l_stmtnr_aller_entry_dcl_1 (lsb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
```



```
      if eq.charstring (ssl_6 (ssts_1 (s)),entry)
        then cons_lnat (ssl_7 (ssts_1 (s)),s),
                        gen_l_stmtnr_aller_entry_dcl_1 (rest))
        else gen_l_stmtnr_aller_entry_dcl_1 (rest)
      fi
   esac ;

gen_l_stmtnr_aller_entry_dcl (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
     append_lnat (gen_l_stmtnr_aller_entry_dcl_1 (
                    preorder_sb (t_baum_sb (s,rest))),
                  gen_l_stmtnr_aller_entry_dcl (rest,sb))
   esac ;

lösche_entry_variable_1 (nn,lsb):=
   case lsb is
   *nil_lsb-->cons_lni (nn,nil_lni)
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),variable)
        then nil_lni
        else lösche_entry_variable_1 (nn,rest)
      fi
   esac ;

lösche_entry_variable (lni,sb):=
   case lni is
   *nil_lni-->nil_ln
   *cons_lni (l,rest)-->
     append_lni (lösche_entry_variable_1 (l,
                   gen_l_stmtnr (sni_2 (l),sb))
                 lösche_entry_variable (rest,sb))
   esac ;

gen_l_entrynamen_1 (n,lsb):=
   case lsb is
   *nil_lsb-->nil_lni
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),declare identifier)
         and eq.nat (ssl_7 (ssts_1 (s)),n)
        then cons_lni (erz_ni (sdt_3 (ssts_2 (s)),n),
                       gen_l_entrynamen_1 (n,rest))
        else gen_l_entrynamen_1 (n,rest)
      fi
   esac ;

ENDSPEC
```

3.2.1.3. Vollständigkeit Secondary Entries

3.2.1.3.1. Beschreibung

Ausgehend von einer Menge von Quellprogrammen ist für jedes Quellmodul die Liste aller enthaltener Secondary Entries, die mit dem Quellmodulnamen zu qualifizieren sind, zu erzeugen.

Beispiel: Sei M das folgende auszuwertende Quellmodul:

```
M : Procedure;
    ...
E1: ENTRY (BIT);
    ...
E2: ENTRY (BINARY FIXED);
    ...
END M;
```

Die Ausgabe der Auswertefunktion besteht in der Liste (M_(E1 E2)).
Bemerkung: Secondary Entries interner Prozeduren werden nicht berücksichtigt.

3.2.1.3.2. Spezifikation Vollständigkeit Secondary Entries

```
INSTANTIATE Tupel to Name_L(Name)
ACTUALIZE : #x1 by Name
            #x2 by L(Name)
            SORTS : #x1.elem by name
                    #x2.elem by l(name)
RENAME : SORTS : tupel by name_l(name)
         OPS : erz_tup by erz_nln
END


INSTANTIATE Liste to L(Name_L(Name))
ACTUALIZE : #x by Name_L(Name)
            SORTS : #x.elem by name_l(name)
RENAME : SORTS : liste by l(name_l(name))
         OPS : nil by nil_lnln
END
```



```
SSPEC VOLLSTÄNDIGKEIT SECONDARY ENTRIES

USE SSPECS : Schnittstelle, L(Name_L(Name))

PUBLIC OPS :
op_vollst_sec_entries:l(name) l(dat)-->l(name_l(name))

PROPERTIES :
/*
op_vollst_sec_entries  (ln ,ld)  liefert zu jedem Modul aus der  Liste
der Modulnamen, die Liste der Sec. Entries des Moduls.
*/
∀ l(name) ln: ∀ l(dat) ld: ∃ l(name_l(name)) l:
  l=op_vollst_sec_entries (ln,ld)
  ∧ ∀ name_l(name)) n: ist_enth_lnln (n,l)=true
      ⇒ (ist_enth_ln (snln_1 (n),ln)=true
         ∧ (∀ name e: ist_enth_ln (e,snln_2 (n))=true
              ⇒ ist_enth_ln (e,
                  sec_entries_in_modul (lies_sb (snln_1 (n),
                    liefere_eind_modul (snln_1 (n),ld))))=true))

DEFINE OPS :
op_vollst_sec_entries(ln,ld):=
   case ln is
   *nil-ln-->nil_ln
   *cons_ln(n,rest)-->
     cons_lnln(erz_nln(m,sec_entries_in_modul(
                              lies_sb(m,liefere_eind_modul(m,ld)))),
               op_vollst_sec_entries(rest,ld)
   esac ;

ENDSPEC
```

3.2.1.4. Vollständigkeit Nicht Eindeutige Entries

3.2.1.4.1. Beschreibung

Ausgangspunk dieser Teilauswertefunktion ist eine Menge von Quellprogrammen. Für jeden Quellmodul ist zu prüfen, ob der Ansprungpunkt der im Quellmodul aufgerufenen externen Entries eindeutig ist.
Die Ausgabe besteht in einer Liste, in der zu jedem Quellmodulnamen alle im Quellmodul aufgerufenen externen Entries vermerkt sind. Zu jedem Entrynamen sind die Namen der Quellmodule, qualifiziert mit dem Dateinamen, aufgeführt, die einen Entry oder Secondary Entry mit entsprechendem Namen besitzen.

Beispiel: In Modul A werden die externen Entries E1 und E2 aufgerufen. In Quelldatei D3 existiere ein Modul M1 mit Secondary Entry E1 und in Quelldatei D1 existiere ein Modul M2 mit Secondary Entry E1. Außerdem existiere in D3 ein Modul E2 und in D2 ein Modul M3 mit Secondary Entry E2.
(A_(E1_((M1_D3),(M2_D1)),
E2_((E2_D3),(M3_D2)))

Bemerkung: Eindeutige Fälle, d.h. es gibt zu einem aufgerufenen Entry genau einen Ansprungpunkt, werden nicht in die Liste aufgenommen.

3.2.1.4.2. Spezifikation Vollständigkeit Nicht Eindeutige Entries

```
INSTANTIATE Tupel to Name_Name
ACTUALIZE : #x1 by Name
            #x2 by Name
            SORTS : #x1.elem by name
                    #x2.elem by name
RENAME : SORTS : tupel by name_name
           OPS : erz_tup by erz_nn
END


INSTANTIATE Liste to L(Name_Name)
ACTUALIZE : #x by Name_Name
            SORTS : #x.elem by name_name
RENAME : SORTS : liste by l(name_name)
           OPS : nil by nil_lnn
END


INSTANTIATE Tupel to Entry_mögl_entries
```



```
ACTUALIZE : #x1 by Name
            #x2 by L(Name_Name)
            SORTS : #x1.elem by name
                    #x2.elem by l(name_name)
RENAME : SORTS : tupel by entry_mögl_entries
           OPS : erz_tup by erz_eme
END


INSTANTIATE Tupel to Nicht_eind_entry
ACTUALIZE : #x1 by Name
            #x2 by L(Entry_mögl_entries)
            SORTS : #x1.elem by name
                    #x2.elem by l(entry_mögl_entries)
RENAME : SORTS : tupel by l(entry_mögl_entries)
           OPS : erz_tup by erz_eme
END


INSTANTIATE Liste to L(Nicht_eind_entry)
ACTUALIZE : #x by Nicht_eind_entry
            SORTS : #x.elem by nicht_eind_entry
RENAME : SORTS : liste by l(nicht_eind_entry)
           OPS : nil by nil_lnee
END


INSTANTIATE Liste to L(Entry_mögl_entries)
ACTUALIZE : #x by Entry_mögl_entries
            SORTS : #x.elem by entry_mögl_entries
RENAME : SORTS : liste by l(entry_mögl_entries)
           OPS : nil by nil_leme
END


SSPEC VOLLSTÄNDIGKEIT NICHT EINDEUTIGE ENTRIES

USE SSPECS : Schnittstelle, Vollständigkeit der Entries,
             L(Nicht_eind_entries)

PUBLIC OPS :
op_vollst_nicht_eind_entries:l(name) l(dat)-->l(nicht_eind_entry)
gen_l_entry_mögl_entries:l(name) l(dat)-->l(entry_mögl_entries)
entferne_leere_elem:l(nicht_eind_entry)-->l(nicht_eind_entry)
```

```
lösche_eind_fälle:l(nicht_eind_entry)-->l(nicht_eind_entry)
name_zur_datei:l(name_dat)-->l(name_name)

PROPERTIES :
/*
op_vollst_nicht_eind_entry (ln,ld) liefert für jeden Modul der Liste
der Modulnamen ln die Liste der im Modul aufgerufenen Entries, für die
der Ansprungpunkt nicht eindeutig ist, d.h. daß es mehrere Quellmodule
auf den Quelldateien mit identischen Entrynamen gibt.
*/
∀ l(name)ln: ∀ l(dat) ld: ∃ l(nicht_eind_entry) l:
  l=op_vollst_nicht_eind_entry (ln,ld)
  ∧ (ist_leer_lnee (l)=false
    ⇒ (ist_enth_ln (snee_1 (n),ln)=true
       ∧ (∃ l(entry_mögl_entries) le:
            le=snee_2 (n)
            ∧ ∀ entry_mögl_entries em:
                ist_enth_leme (em,le)=true
                ⇒ (ist_enth_ln (seme_1 (em),
                   gen_l_aufr_ext_entries (erz_nd (snee_1 (n),
                     liefere_eind_modul (snee_1 (n),ld))))=true
                   ∧ seme_2 (em)=name_zur_datei (
                       gen_l_nicht_eind_entries (snee_1 (n),ld))
                   ∧ anzahl_lnn (seme_2 (em))>1))))

/*
name_zur_datei (lnd) ersetzt jedes Element Modulname-Datei in lnd
durch das Paar Modulname-Quelldateiname.
*/
∀ l(name_dat) lnd: ∃ l(name_name) lnn:
  lnn=name_zur_datei (lnd)
  ∧ ∀ name_name n: ist_enth_lnn (n,lnn)=true
      ⇒ ∃ datei d: ist_enth_lnd (erz_nd (snn_1 (n),d)),lnd)=true
          ∧ snn_2 (n)=name_quelldatei (d)

/*
entferne_leere_elem (lnee) streicht in der Ausgabeliste lnee die Ele-
mente, deren zweite Komponente, d.i. die Liste der im Modul aufgerufe-
nen Entries, leer ist.
*/
∀ l(nicht_eind_entry) lnee: ∃ l(nicht_eind_entry) l:
  l=entferne_leere_elem (lnee)
  ∧ ∀ nicht_eind_entry e:
      ist_enth_lnee (e,l)=true
      ⇒ ist_leer_leme (snee_2 (e))=false

/*
lösche_eind_fälle (lnee) als Operationsaufruffolge von entf_leere_elem
```



```
(lösche_eind_fälle_1 (lnee)) bewirkt, daß nur solche Elemente in der
Ausgabeliste erscheinen, für die gilt, daß die Ansprungpunkte der auf-
gerufenen Entries tatsächlich nicht eindeutig sind.
*/
∀ l(nicht_eind_entry) lnee: ∃ l(nicht_eind_entry) l:
  l=lösche_eind_fälle (lnee)
  ∧ (ist_leer_lnee (l)=false
     ⇒ ∀ nicht_eind_entry n:
         ist_enth_lnee (n,l)=true
         ⇒ ∀ entry_mögl_entries e:
             ist_enth_leme (e,snee_2 (n))=true
             ⇒ anzahl_lnn (seme_2 (e))>1)

PRIVATE OPS :
lösche_eind_fälle_1:l(nicht_eind_entry)-->l(nicht_eind_entry)
lösche_eind_fälle_2:nicht_eind_entry-->nicht_eind_entry
lösche_eind_fälle_3:l(entry_mögl_entries)-->l(entry_mögl_entries)
op_vollst_nicht_eind_entries_1:l(name) l(dat)-->l(nicht_eind_entry)

DEFINE OPS :
op_vollst_nicht_eind_entries (ln,ld):=
  lösche_eind_fälle (op_vollst_nicht_eind_entries_1 (ln,ld));

op_vollst_nicht_eind_entries_1 (ln,ld):=
   case ln is
   *nil_ln-->nil_lnee
   *cons_ln (n,rest)-->
      cons_lnee (erz_nee (n,gen_l_entry_mögl_entries (
                           gen_l_aufr_ext_entries (erz_nd (n,
                                liefere_eind_modul (n,ld))),ld)),
                 op_vollst_nicht_eind_entries_1 (rest,ld))
   esac ;

gen_l_entry_mögl_entries (ln,ld):=
   case ln is
   *nil_ln-->nil_leme
   *cons_ln (e,rest)-->
      cons_leme (erz_eme (n,name_zur_datei (
                          gen_l_nicht_eind_entries (n,ld))),
                 gen_l_entry_mögl_entries (rest,ld))
   esac ;

name_zur_datei (lnd):=
   case lnd is
   *nil_lnd-->nil_lnn
   *cons_lnd (nd,rest)-->
     cons_lnn (erz_nn (snd_1 (nd), name_quelldatei (snd_2 (nd))),
               name_zur_datei (rest))
```

```
   esac ;

lösche_eind_fälle (lnee):=
   entferne_leere_elem (lösche_eind_fälle_1 (lnee));

lösche_eind_fälle_1 (lnee):=
   case lnee is
   *nil-lnee-->nil_lnee
   *cons_lnee (n,rest)-->
     cons_lnee (lösche_eind_fälle_2 (n), lösche_eind_fälle_1 (rest))
   esac ;

lösche_eind_fälle_2 (nee):=
   erz_nee (snee_1 (nee),lösche_eind_fälle_3 (snee_2 (nee));

lösche_eind_fälle_3 (leme):=
   case leme is
   *nil_leme-->nil_leme
   *cons_leme (e,rest)-->
      if ist_gt (anzahl_lnn (seme_2 (e)),succ (0))
        then cons_leme (e,lösche_eind_fälle_3 (rest)
        else lösche_eind_fälle_3 (rest)
      fi
   esac ;

entferne_leere_elem (lnee):=
   case lnee is
   *nil_lnee-->nil_lnee
   *cons_lnee (n,rest)-->
      if ist_leer_leme (snee_2 (n))
        then entferne_leere_elem (rest)
        else cons_lnee (n,entferne_leere_elem (rest))
      fi
   esac ;

ENDSPEC
```


3.2.1.5. Spezifikation Vollständigkeit der Komponenten

```
INSTANTIATE 4_Tupel to Vollst_d_Komp
ACTUALIZE : #x1 by L(Name)
            #x2 by L(Name_L(Name_Nat))
            #x3 by L(Name_L(Name))
            #x4 by L(Nicht_eind_entry)
            SORTS : #x1.elem by l(name)
                    #x2.elem by l(name_l(name_nat))
                    #x3.elem by l(name_l(name))
                    #x4.elem by l(nicht_eind_entry)
RENAME : SORTS : 4_tupel by vollst_d_komp
         OPS : erz_vier by erz_vollst_d_komp
END


SSPEC VOLLSTÄNDIGKEIT DER KOMPONENTEN

USE SSPECS : Vollständigkeit der Entries,
             Vollständigkeit der Externverweise,
             Vollständigkeit Secondary Entries,
             Vollständigkeit Nicht Eindeutige Entries,
             Vollst_d_Komp, L(Quelldatei), L(Name),
             Schnittstelle
PUBLIC OPS :
op_vollst_d_komponenten(ln,ld):l(name) l(dat)-->vollst_d_komp

PROPERTIES :
/*
op_vollst_d_komponenten (ln,ld)  erzeugt aus den Ergebnissen der
Teilauswertefunktionen ein Gesamtergebnis.
*/

DEFINE OPS :
op_vollst_d_komponenten(ln,ld):=
  erz_vollst_d_komp(op_vollst_d_entries(ln,ld),
                    op_vollst_externverweise(ln,ld),
                    op_vollst_sec_entries(ln,ld),
                    op_vollst_nicht_eind_entries(ln,ld));

ENDSPEC
```

3.2.2. Enhaltenstruktur

3.2.2.1. Beschreibung

Sinn dieser Auswertefunktion ist es, die interne Struktur eines Quellmoduls darzustellen. Die interne Struktur besteht aus der Schachtelung der internen Prozeduren und der Secondary Entries. Dabei enthalten Secondary Entries die Betrachtungsobjekte (int. Proz, Sec. Entries), die nach dem Secondary Entry deklariert sind.

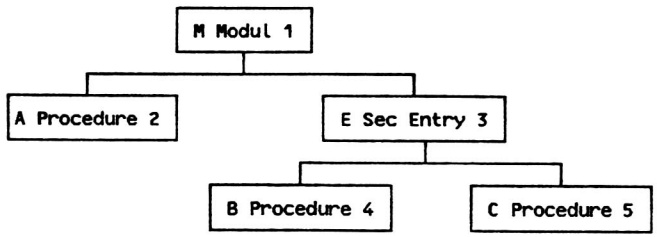
Die Ausgabe dieser Auswertefunktion ist eine Baumstruktur. Die Knoten dieses Baumes bestehen aus dem Namen der Betrachtungsobjekte, einer Kennzeichnung (int Proz oder Sec Entry) und einer Zahl, die der fortlaufenden Nummerierung der Knoten entspricht. Die Kanten des Baumes sind von oben nach unten als "ist deklariert in" zu lesen.

Da dieser Baum sehr umfangreich und damit unübersichtlich werden kann, besteht die Möglichkeit über die Angabe eines Fensters sich eine Teilhierarchie der Enthaltenstruktur erstellen zu lassen. Als Fenster kommt jedes Betrachtungsobjekt im Quellmodul in Frage.

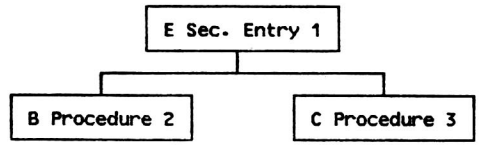
Beispiel:
a) Quellmodul M

```
M : PROCEDURE;
    A : PROCEDURE;
        ...
    END A;
E : ENTRY;
    B : PROCEDURE;
        ...
    END B;
    C : PROCEDURE;
        ...
    END C;
    ...
END M;
```

b) Enthaltenstruktur



c) Bei Angabe des Fensters "E Sec. Entry" reduziert sich die Enthaltenstruktur zu:



In der Beschreibung der ST wurde bereits festgestellt, daß in der ST für jede Prozedur ein Prozedurblock mit dem Namen der Prozedur existiert.

Sec. Entries, die durch das Entrystatement deklariert sind, finden sich in der ST im entsprechendem Gebiet als Eintrag wieder. Der Name des Sec. Entry ist im Namensfeld enthalten, während die Attributliste das Attribut "Secondary Entry" enthält.

3.2.2.2. Spezifikation Enthaltenstruktur

```
SSPEC FENSTER

USE SSPECS : Bool, Name

PUBLIC SORTS : fenster

PUBLIC OPS :
alles:-->fenster
name_intproz:name-->fenster
name_sec_entry:name-->fenster

CONSTRUCTORS :
*alles
*name_intproz
*name_sec_entry

ENDSPEC


INSTANTIATE 3_Tupel to Knoten_es
ACTUALIZE : #x1 by Name
            #x2 by Charstring
            #x3 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by charstring
                    #x3.elem by nat
RENAME : SORTS : 3_tupel by knoten_es
         OPS : erz_drei by erz_kes
END


INSTANTIATE Baum to Enth_Struktur_1
ACTUALIZE : #x by Knoten_es
            SORTS : #x.elem by knoten_es
RENAME : SORTS : baum by enth_struktur
         OPS : erz_baum by erz_es
               son by son_es
               anf by anf_es
END
```



```
SSPEC ENTHALTENSTRUKTUR

USE SSPECS : Bool, Nat, Fenster
             Schnittstelle, Enth_Struktur_1

PUBLIC OPS :
op_enthaltenstruktur:name fenster l(dat)-->enth_struktur
h_index:enth_struktur-->nat
anf_hier:l(knoten_st) st enth_struktur knoten_es-->enth_struktur

PROPERTIES :
/*
h_index (es)  liefert den größten Index aller in der Enthaltenstruktur
vorhandenen Knoten. Der Index ist die dritte Komponente eines Knotens.
gemeint.
*/
∀ enth_struktur es: ∃ nat n: n=h_index (es)
  ∧ ¬ (∃ knoten_es kes: ist_in_baum_es (kes,es)=true
       ∧ skes_3 (kes)>n)
  ∧ ∃ knoten_es k: ist_in_baum_es (k,es)=true
      ∧ skes_3 (k)=n

/*
liefere_knst  (n,lst)  liefert aus der Liste von Knoten der ST lst den
Knoten, der Prozedurblock des Namens n ist.
*/
∀ l(knoten_st) lst: ∀ name n: ∃ knoten_st k:
  k=liefere_knst (n,lst)
  ∧ ist_enth_lst (k,lst)=true
  ∧ ist_proz_block (k)=true
  ∧ blockname (k)=n

/*
liefere_kst_zu_sec_entry (e,lst)  liefert aus der Liste der Knoten der
ST  lst den Knoten,  der einem Gebiet entspricht,  das die Deklaration
des Sec. Entry e enthält.
*/
∀ name e: ∀ l(knoten_st) lst: ∃ knoten_st k:
  k=liefere_kst_zu_sec_entry (e,lst)
  ∧ ist_enth_lst (k,lst)=true
  ∧ ist_entry_enth (k)=true
  ∧ entryname (k)=e

/*
wechsel_kst_zu_kes (kst,es) liefert, sofern kst ein Knoten der ST ist,
der einen Prozedurblock darstellt, einen Knoten der Enthaltenstruktur,
der aus dem Prozedurnamen,  einer Kennzeichnung "Interne Prozedur" und
dem höchsten Index aus es besteht.
```

```
*/
∀ knoten_st kst: ∀ enth_struktur es: ∃ knoten_es k:
  k=wechsel_kst_zu_kes (kst,es)
  ∧ (k≠error
    ⇒ (skes_1 (k)=blockname (kst)
      ∧ skes_2 (k)=Interne Prozedur
      ∧ skes_3 (k)=h_index (es))

/*
restgebiet (kst,e)
Sofern der Knoten kst ein Gebiet darstellt, werden in dem Gebiet alle
Einträge einschließlich des Eintrags, der in der Attributliste das
Attribut "Sec. Entry" und im Namensfeld den Namen e enthält, gelöscht.
Das Ergebnis ist die Liste der verbleibenden Einträge.
*/
∀ knoten_st kst: ∀ name e: ∃ gebiet gb:
  gb=restgebiet (kst,e)
  ∧ (gb≠error
    ⇒ ∃ gebiet g: kst=erz_kst (blck_g (g),skst_2 (kst))
        ∧ (∃ eintrag etr: ist_enth_gb (etr,g)=true
            ∧ setr_1 (etr)=e
            ∧ ist_enth_attr (Sec. Entry,setr_2 (etr))=true)
        ∧ gb=l_rest_gb (etr,g))

/*
liefere_nachf (e,kst,st) liefert eine Liste l von Knoten der ST. kst
ist ein Gebiet, das die Deklaration des Sec. Entry e enthält. Das er-
ste Element der Liste l wird zum Teil durch die Operation restgebiet
(kst,e) geliefert. Der Rest der Liste l besteht aus allen rechten Brü-
dern des Knotens kst in der ST st.
*/
∀ name e: ∀ knoten_st kst: ∀ st st:
  ∃ l(knoten_st) l:
   l=liefere_nachf (e,kst,st)
   ∧ car_lst (l)=erz_kst (restgebiet (kst,e),skst_2 (kst))
   ∧ cdr_lst (l)=l_rest_lst (kst,lds_st (vater_st (kst,st),st))

/*
anf_hier (lst,st,es,kes) ist die zentrale Operation der Spezifikation.
lst ist die Liste von Knoten der ST st, die abzuarbeiten ist.es ist
die Enthaltenstruktur, soweit wie sie bereits erstellt worden ist. kes
ist der aktuelle Knoten der Enthaltenstruktur es, an den der nächste
aus lst zu ermittelnde Knoten, anzuhängen ist. Dieser anzuhängende
Knoten sei k. Ehe nun lst vollständig abgearbeitet wird, wird zunächst
die Teilhieratchie der Enthaltenstruktur für den Knoten k ermittelt
und angefügt (Erst Tiefe, dann Breite). Mit dieser geänderten Ent-
haltenstruktur wird dann der Rest der Liste lst abgearbeitet.
```


Die Operation op_enthalten_struktur (n,f,ld) ruft die Operation anf_hier mit den aktuellen Parametern auf. Diese aktuellen Parameter hängen in erster Linie von der Angabe des Fensters f ab. Aus dem Modulnamen n und der Liste der Quelldateien wird die entsprechende ST erzeugt.

Wird im Fenster "alles" angegeben, dann ist die Enthaltenstruktur für das gesamte Quellmodul zu erstellen. lst besteht aus der Liste der Söhne der Wurzel der ST. es besteht aus einem Knoten mit dem Modulnamen n. kes ist dieser Knoten und somit die Wurzel von es.

Wird im Fenster der Name einer internen Prozedur angegeben, so wird zunächst der entsprechende Prozedurblock in der ST gesucht. lst besteht dann aus der Liste der Söhne dieses Knotens. es besteht aus einem Knoten mit dem Namen der internen Prozedur. kes ist wiederum genau dieser Knoten.

Wird im Fenster der Name eines Sec. Entry angegeben, so wird in der ST der Knoten gesucht, der das Gebiet darstellt, das die Deklaration dieses Entry enthält. In diesem Gebiet sind weitere Sec. Entries zu suchen, so daß der Rest nach dem Eintrag zu einem neuen Knoten der ST wird und zusammen mit allen rechten Brüdern des Knotens die neue Liste lst bildet. es besteht aus dem Knoten mit Namen des Sec. Entry. Dies ist dann auch der aktuelle Knoten.

```
*/
∀ name n: ∀ fenster f: ∀ l(dat) ld: ∃ st st:
  st=lies_st (n,liefere_eind_modul (n,ld))
  ∧ ∃ enth_struktur es: es=op_enthalten_struktur (n,f,ld)
      ∧ ∀ knoten_es k: (ist_in_baum_es (k,es)=false
          ∨ wurzel_es (es)=k)
          ∨ (skes_2 (k)=procedure
            ∧ ∃ knoten_st kst: ist_in_baum_st (kst,st)=true
                ∧ ist_proz_block (kst)=true
                ∧ blockname (kst)=skes_1 (k)
                ∧ ∃ knoten_st ks: ist_in_baum_st (ks,st)=true
                   ∧ ((ist_proz_block (ks)=true
                     ∧ blockname (ks)=skes_1 (vater_es (k,es))
                     ∧ ist_in_baum_st (kst,t_baum_st (ks,st))=true)
                     ∨ (ist_enth_ln (skes_1 (vater_es (k,es)),
                                     entryliste (ks))=true
                       ∧ ist_enth_lst (kst,l_rest_lst (ks,
                           lds_st (vater_st (ks,st),st)))=true))
          ∨ skes_2 (k)=secondary entry
            ∧ ∃ knoten_st kn: ist_in_baum_st (kn,st)=true
                ∧ ist_enth_ln (skes_1 (k),entryliste (kn))=true
                ∧ ∃ knoten_st kn': ist_in_baum_st (kn',st)=true
                    ∧ (ist_proz_block (kn')=true
                        ∧ blockname (kn')=skes_1 (vater_es (k,es))
```

```
                        ∧ ist_in_baum_st (kn,t_baum_st (kn',st))
                                                    =true)
                      ∨ (ist_enth_ln (skes_1 (vater_es (k,es)),
                                      entryliste (kn'))=true
                         ∧ ist_enth_lst (kn,cons_lst (
                              restgebiet (kn',skes_1 (vater_es (
                               k,es)),l_rest_lst (kn',lds_st (
                                vater_st (kn',st)),st))=true)
     ∧ ∀ knoten_es kes: ist_in_baum_es (kes,es)=true
         ⇒ ¬ ∃ knoten_es kes': ist_in_baum_es (kes',es)=true
                ∧ kes≠kes' ∧ skes_3 (kes)=skes_3 (kes)

PRIVATE OPS :
liefere_knst:name l(knoten_st)-->knoten_st
liefere_nachf:name knoten_st st-->l(knoten_st)
restgebiet:knoten_st name-->gebiet
restgebiet_1:symtab name-->gebiet
restgebiet_2:gebiet name-->gebiet
wechsel_kst_zu_kes:knoten_st enth_struktur-->knoten_es
liefere_kst_zu_sec.entry:name l(knoten_st)-->knoten_st

DEFINE OPS :
op_enthaltenstruktur (n,f,ld):=
   let x=lies_st (n,liefere_eind_modul (n,ld)) in
    case f is
    *alles-->anf_hier (lds_st (wurzel_st (x),x),
                       x,
                       erz-es (erz_kes (n,modul,1)),
                       erz_kes (n,modul,1))
    *name_intproz (p)-->anf_hier (lds_st (liefere_knst (
                                          p,preorder_st (x)),x),
                                  x,
                                  erz_es (erz_kes (n,intproz,1)),
                                  erz_kes (p,intproz,1))
    *name_sec_entry (e)-->
      anf_hier (liefere_nachf (e,liefere_kst_zu_entry (e,
                                          preorder_st (x),x)),
                x,
                erz_es (erz_kes (e,sec. entry,1)),
                erz_kes (e,sec. entry,1))
   esac ;

anf_hier (lst,st,es,kes):=
   case lst is
   *nil_lst-->es
   *cons_lst (kst,rest)-->
      if ist_prozblock (kst)
        then anf_hier (rest,st,anf_hier (lds_st (kst),st,
```



```
                      anf_es (wechsel_kst_zu_kes (kst,es),kes,es),
                      wechsel_kst_zu_kes (kst,es)),kes)
        else if ist_beg_block (kst)
               then anf_hier (rest,st,
                      anf_hier (lds_st (kst,st),st,es,kes),kes)
               else
                if ist_entry_enth (kst)
                   then
                     anf_hier (cons_lst (erz_kst (blck_g (
                                restgebiet (kst,entryname (kst))),
                                skst_2 (kst)),nil_lst),st,
                     anf_es (erz_kes (entryname (kst),sec. entry,
                                      incr (h_index (es))),
                             kes,es),
                     erz_kes (entryname (kst),sec. entry,
                              incr (h_index (es))))
                   else anf_hier (rest,st,es,kes)
                fi
               fi
     fi
   esac ;

liefere_knst (n,lst):=
   case lst is
   *nil_lst-->error.knoten_st
   *cons_lst (l,rest)--> if ist_prozblock (l)
                         then if eq.name (blockname (l),n)
                                then l
                                else liefere_knst (n,rest)
                              fi
                         else lefere_knst (n,rest)
                        fi
   esac ;

restgebiet (kst,e):=restgebiet_1 (skst_1 (kst),e);

restgebiet_1 (s,e):=
   case s is
   *blck_b (bn)-->error.gebiet
   *blck_g (g)-->restgebiet_2 (g,e)
   esac ;

restgebiet_2 (g,e):=
   case g is
   *nil_gb-->error.gebiet
   *cons_gb (etr,rest)-->
      if ist_enth_attr (sec. entry,setr_2 (etr))
         and eq.name (setr_1 (etr),e)
```

```
        then rest
        else restgebiet_2 (rest,e)
     esac ;

h_index (es):=
   case es is
   *erz_es (kes)-->skes_3 (kes)
   *son_es (t1,t2)--> if ist_gt (h_index (t1),h_index (t2))
                       then h_index (t1)
                       else h_index (t2)
                      fi
   esac ;

liefere_nachf (e,kst,st):=
  cons_lst (erz_kst (restgebiet (kst,e),skst_2 (kst)),
            l_rest_lst (kst,lds_st (kst,st),st)));

wechsel_kst_zu_kes (kst,es):=
   erz_kes (blockname (kst),intproz,incr (h_index (es)));

liefere_kst_zu_sec.entry (e,lst):=
   case lst is
   *nil_lst-->error.knoten_st
   *cons_lst (l,rest)-->
      if ist_entry_enth (l) and eq.name (entryname (l),e)
        then l
        else liefere_kst_zu_sec.entry (e,rest)
      fi
   esac ;

ENDSPEC
```


3.2.3. Benutzthierarchie

3.2.3.1. Beschreibung

Es ist der Steuerungsfluß in einem Programmsystem in Form von Unterprogrammaufrufen darzustellen.

Betrachtungsobjekte sind: Quellmodule, externe Sec. Entries, interne Sec. Entries und interne Prozeduren.s1

Der Steuerungsfluß besteht aus einer Baumstruktur, dessen Knoten Darstellungen der Betrachtungsobjekte sind, und dessen Kanten von oben nach unten als "ruft auf" zu lesen sind. Zusätzlich ist jedem Knoten eine natürliche Zahl hinzuzufügen, aus der hervorgeht, wie oft das entsprechende Objekt aufgerufen wird.

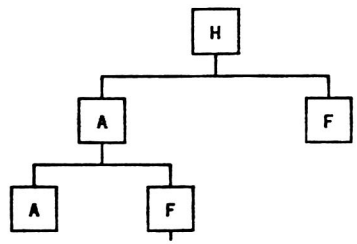
Die Ruft-auf-Hierarchie endet in einem Knoten, falls das diesem Knoten entsprechende Objekt selbst keine anderen Betrachtungsobjekte aufruft, falls das Objekt sich selbst aufruft (auch über mehrere Objekte hinweg) oder falls für das Objekt bereits die Hierarchie ermittelt worden ist.

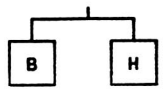
Analog zur Enthaltenstruktur ist es möglich, sich durch Angabe eines Fensters eine Teilhierarchie der Benutzthierarchie erstellen zu lassen.

Die Eingabe dieser Auswertefunktion besteht aus einem Quellmodulnamen, einem Fenster und der Liste der Quelldateien. Ausgabe ist die Ruft-auf-Hierarchie, d.i. die Benutzthierarchie.

Bemerkung: Sec. Entries erhalten die gleiche Teilhierarchie wie das Betrachtungsobjekt, in dem sie enthalten sind.

Beispiel: Das Quellmodul H rufe die interne Prozedur A und das externe Quellmodul F auf. A rufe ebenfalls das Quellmodul F und rekursiv sich selbst (A) auf. F rufe die interne Prozedur B und das Quellmodul H auf.





Für jeden Aufruf eines Betrachtungsobjekts gibt es im SB einen Stammsatz mit Satzart CALLEXT bzw. CALLEXTFCT für den Aufruf einer externen Prozedur oder Funktion, bzw. CALL oder CALLFCT für den Aufruf eines internen Objekts.

3.2.3.2. Spezifikation Benutzthierarchie

```
INSTANTIATE 5_Tupel to Callstmt
ACTUALIZE : #x1 by Name
            #x2 by Name
            #x3 by Charstring
            #x4 by Nat
            #x5 by Name
            SORTS : #x1.elem by name
                    #x2.elem by name
                    #x3.elem by charstring
                    #x4.elem by nat
                    #x5.elem by name
RENAME : SORTS : 5_tupel by callstmt
           OPS : erz_fünf by erz_cs
                 sfünf_1 by scs_1
END


INSTANTIATE 7_Tupel to Knoten_bh
ACTUALIZE : #x1 by Nat
            #x2 by Name
            #x3 by L(Name)
            #x4 by Charstring
            #x5 by Nat
            #x6 by Name
            #x7 by Nat
            SORTS : #x1.elem by nat
                    #x2.elem by name
                    #x3.elem by l(name)
                    #x4.elem by charstring
                    #x5.elem by nat
                    #x6.elem by name
                    #x7.elem by nat
END
```



```
INSTANTIATE Baum to Benutzthierarchie_1
ACTUALIZE : #x by Knoten_bh
            SORTS : #x.elem by knoten_bh
RENAME : SORTS : baum by benuztzhierarchie
           OPS : erz_baum by erz_bh
                 son by son_bh
END


SSPEC BENUTZTHIERARCHIE

USE SSPECS : Fenster, Benuzthierarchie_1, Callstmt,
             Schnittstelle, L(Name), L(Quelldatei)

PUBLIC OPS :
op_benutzthierarchie:name fenster l(dat)-->benutzthierarchie
en_l_entrypoints:stammsatz sb-->l(name)
l_aller_entrynamen:name datei-->l(name)
l_aller_sec.entrynamen:name name datei-->l(name)
best_äußerste_proz:sb-->stammsatz

PROPERTIES :
/*
Die nach außen angebotene Operation op_benutzt_hierarchie erstellt die
gewünschte Benutzthierarchie (BH). Aufgrund des angegebenen Fensters
wird die Operation anf_bhier mit unterschiedlichen Parametern aufgeru-
fen.
   Die Operation anf_bhier entspricht ihrem Aufbau nach der Operation
anf_hier der SSPEC Enthaltenstruktur.
Die formalen Parameter der Operation anf_bhier sind:
- Liste von Knoten der BH: Dies ist die Liste der Knoten, die evtl. an
  den aktuellen Knoten der BH anzuhängen sind.
- Die BH: Dies ist der bereits erstellte Baum der BH.
- Ein Knoten der BH: Dies ist der aktuelle Knoten, an den die nächsten
  Knoten aus der Liste der Knoten der BH anzuhängen sind.
- Liste der Quelldateien: Da in einem Quellmodul auch externe Entries
  aufgerufen werden können, muß diese Liste der Quellbibliotheken als
  Parameter übergeben werden, da die BH auch für die aufgerufenen ex-
  ternen Objekte zu erstellen ist.

Im Fenster werden drei Fälle unterschieden:

1.) - alles: alles bedeutet, es ist der gesamte Quellmodul auszuwer-
      ten.

   Die Operation lac (liste aller calls) ermittelt im SB der aktuellen
   Prozedur sämtliche Stammsätze, die einen Aufruf einer Prozedur oder
```

eines Sec. Entry enthalten. Diese Stammsätze werden in sogenannte Call-Stmts überführt, die die wesentliche Information der Stammsätze enthalten. D.i. der Modulname, der Name des aufgerufenen Objekts, das Aufrufattribut, d.i. die Satzart (call, callfct, callext, callextfct), die Identifikation des Stammsatzes und der Dateiname der Datei, die den Quellmodul enthält.

Die Operation lap (liste aller prozeduren) überführt die Liste der Call-Stmts in eine Liste von Knoten der BH, wobei jeder Knoten der aufgerufenen Prozedur entspricht. Diese Knoten enthalten an Information:
die Aufrufhäufigkeit,
den Modulnamen,
den Namen des Objekts,
die Kennzeichnung des Objekts, d.i. das Attribut Interne Prozedur, Sec. Entry, Modul, Ext. Prozedur oder Ext. Sec. Entry,
die Identifikation des Stammsatzes und
den Dateinamen.
lap ruft zunächst die Operation ermittle_uo auf, die zu einem Call-Stmt das zugehörige Objekt ermittelt und den entsprechenden Stammsatz in einen Knoten der BH überführt. Durch Zählen der übereinstimmenden Knoten in dieser Liste wird durch die Operation aufrufhäufigkeit die Aufrufhäufigkeit ermittelt.

2.) - name_int_proz (n): Im aktuellen SB wird der Teil-SB der angegebenen internen Prozedur ermittelt. Dieser Teilbaum wird an die Operation lac übergeben.

3.) - name_sec_entry (e): Nach SIEMENS-Funktionskatalog erhält ein Sec. Entry die gleiche Aufrufhierarchie wie die Prozedur, die den Entry enhält. Deshalb wird zunächst die innerste Prozedur gesucht, die den angegebenen Sec. Entry e enthält. lac wird mit dem Teilbaum des SB der gefundenen Prozedur aufgerufen.

Die Operation anf_bhier hängt die aufgerufenen Objekte des aktuellen Knotens der BH an den aktuellen Knoten an. Die Operation ruft sich rekursiv auf und endet, wenn ein Objekt kein anderes aufruft, oder wenn ein Objekt bereits abgearbeitet wurde.
Die Operation arbeitet wie folgt:
Ist die Liste der Knoten der BH leer, dann ist die als Parameter übergebene BH die gesuchte. Im anderen Fall wird überprüft, ob für den aktuellen Knoten, an den anzuhängen ist, bereits die BH ermittelt wurde. Dies leistet die boolsche Operation ist_bearbeitet. Falls ja, ist die übergebene BH wiederum die gesucht. Falls nein, wird jeder Knoten der Liste von Knoten der BH an den aktuellen Knoten angehängt, nachdem die BH des ersten Knotens dieser Liste durch die Operationsaufruffolge lap (ermittle_calls (kbh,ld),ld) ermittelt und angehängt worden ist.

Beispiel: A rufe B und C auf. B rufe D auf und C rufe E auf. An A wird zunächst B angehängt. Bevor nun C an A angefügt wird, wird erst noch D an B gehängt. Nachdem C an A angefügt wurde, wird auch E an C gehängt.

Die Operation ermittle_calls liefert mittels der Operation lac die Call-Stmts in der Prozedur, die dem Knoten der BH entspricht. An lac wird der SB übergeben, in dem die Aufrufe zu suchen sind.

Die Operation ermittle_uo, die von der Operation lap benutzt wird, liefert zu einem Aufruf das tatsächliche Objekt. Da nicht davon ausgegangen werden Kann, daß eindeutige Namen vergeben worden sind, ist das aufgerufene Objekt im jeweiligem Gültigkeitsbereich zu suchen.
ermittle_uo unterscheidet grundsätzlich zwischen Aufrufen von internen und Aufrufen von externen Objekten.

Interne Objekte werden durch die Operation suche_obj_zu_call ermittelt. suche_obj_zu_call wird mit dem vollqualifiziertem Stammsatz, in dem der Aufruf vorkommt, aufgerufen.
suche_obj_zu_call ruft die Operation su_obj_1 mit dem Stammsatz der innersten Prozedur oder des innersten Begin Blocks, in dem der Aufruf enthalten ist, auf.
su_obj_1 überprüft, sofern der innnerste Block eine Prozedur ist, ob es die zugehörige Prozedur zum Aufruf ist. In diesem Fall handelt es sich um einen rekursiven Prozeduraufruf. Aus dem Stammsatz wird dann der Knoten für die BH erzeugt. In allen anderen Fällen wird die Operation su_obj_2 mit den Argumenten Liste der Söhne des Stammsatzes des innersten Blocks des SB, dem Stammsatz des Aufrufs und dem Stammatz des innersten Blocks augerufen.
su_obj_2 durchläuft die Liste von Stammsätzen. Ist diese Liste leer, dann wird, sofern die Wurzel des SB noch nicht erreicht ist, su_obj_1 mit dem nächst äußeren Block des Stammsatzes des Aufrufs aufgerufen. Ist die Liste nicht leer, unterscheidet man nach den Satzarten der Liste.
Im Fall, daß ein Begin Block vorliegt, wird nur der Rest der Liste abgearbeitet, denn alle Prozeduren, die in diesem Begin Block deklariert sind, sind auch nur in diesem Begin Block bekannt.
Im Fall, daß ein Prozedurblock vorliegt, wird überprüft, ob es die Prozedur zum Aufruf ist, indem die Namen auf Gleichheit geprüft werden, d.h. da eine Prozedur bzw. ein Sec. Entry mehrere Entrynamen besitzen kann, die den gleichen Entrypunkt kennzeichnen, ist zu prüfen, ob der aufgerufene Name in der Liste der Entrynamen enthalten ist. Die Liste der Entrynamen zu einer Prozedur bzw. einem Sec. Entry liefert die Operation gen_l_entrypoints. Ist dies so, dann ist der gesuchte Stammsatz gefunden, und es wird hieraus der entsprechende Knoten der BH erstellt. Ist dies nicht so, wird überprüft, ob in der Prozedur ein Sec. Entry definiert ist, der zum Aufruf gehört. Falls ja, wird aus diesem Sec. Entry der Knoten der BH erstellt. Dies leisten die Operationen ist_call_eines_sec_entry und entry_zum_call. Im Fall,

daß kein Prozedurblock und kein Begin Block vorliegt, wird die Operation su_obj_2 mit der durch die Söhne dieses Stammsatzes verlängerten Liste, dem Stammsatz des Calls und dem Stammsatz des gerade untersucht wordenen Blocks rekursiv aufgerufen.
ist_call_eines_sec_entry ruft die Hilfsoperation ist_call_sec_entry_1 mit den Söhnen des Stammsatzes des Prozedurblocks auf. Diese Liste wird mit den Söhnen des gerade aktuellen Stammsatzes verlängert, sofern der Stammsatz nicht die Satzart "procedure" oder "executable stmt" hat. Bei Satzart "secondary entry" werden die Namen überprüft. Stimmen sie überein, so liefert die Operation true, ansonsten wird der Rest der Liste verarbeitet. Bei leerer Liste liefert die Operation false.
Analog zu dieser boolschen Operation liefert die Operation entry_zum_call den gesuchten Stammsatz und erzeugt daraus den Knoten für die BH.

Externe Objekte werden durch die Operation suche_ext_obj_zu_call ermittelt. Zu diesem Zweck benutzt die Operation die Operation liefere_eind_entry der SSPEC Schnittstelle.
*/

```
/*
gen_l_entrypoints (s,sb) liefert die Liste der Entrynamen zu dem durch
s gegebenen Prozedur-Stmt bzw. Entry-Stmt.
*/
∀ sb sb: ∀ stammsatz s: ∃ l(name) l:
  l=gen_l_entrypoints (s,sb)
  ∧ (l≠error
   ⇒ (ssl_6 (ssts_1 (s))≠proc stmt
    ∧ ssl_6 (ssts_1 (s))≠procedure
    ∧ ssl_6 (ssts_1 (s))≠entry stmt
    ∧ ssl_6 (ssts_1 (s))≠secondary entry
    ∧ (∀ name n: ist_enth_ln (n,l)=true
        ⇒ ∃ stammsatz s1: ist_in_baum_sb (s1,sb)=true
            ∧ ssl_7 (ssts_1 (s1))=ssl_7 (ssts_1 (s))
            ∧ ssl_6 (ssts_1 (s1))=label identifier
            ∧ sdt_3 (ssts_2 (s1))=n)
    ∧ ¬ ∃ stammsatz s1': ist_in_baum_sb (s1',sb)=true
          ∧ ssl_7 (ssts_1 (s1'))=ssl_7 (ssts_1 (s))
          ∧ ssl_6 (ssts_1 (s1'))=label identifier
          ∧ ist_enth_ln (sdt_3 (ssts_1 (s1')),l)=false))

/*
l_aller_entrynamen (n,d) liefert alle Entrynamen des Quellmoduls n der
Quelldatei d.
*/
∀ name n: ∀ datei d: ∃ l(name) l:
  l=l_aller_entrynamen (n,d)
```



```
  ∧ (l≠error
    ⇒ ∃ stammsatz s: ist_in_baum_sb (s,lies_sb (n,d))=true
        ∧ ssl_6 (ssts_1 (s))=procedure
        ∧ vater_sb (s,lies_sb (n,d))=wurzel_sb (lies_sb (n,d))
        ∧ l=gen_l_entrypoints (s,lies_sb (n,d)))

/*
l_aller_sec_entrynamen (n,e,d) liefert die Liste sämtlicher Entrynamen
des Secondary Entry mit Namen e im Quellmodul n der Datei d.
*/
∀ name n,e: ∀ datei d: ∃ l(name) l:
  l=l_aller_sec_entrynamen (n,e,d)
  ∧ (l≠error
    ⇒ ∃ stammsatz s: ist_in_baum_sb (s,lies_sb (n,d))=true
        ∧ (ssl_6 (ssts_1 (s))=secondary entry
           ∨ ssl_6 (ssts_1 (s))=entry stmt)
        ∧ vater_sb (enth_vater_proc (s,lies_sb (n,d)),
            lies_sb (n,d))=wurzel_sb (lies_sb (n,d))
        ∧ l=gen_l_entrypoints (s,lies_sb (n,d)))

/*
high_index (bh) liefert den größten Index aller Knoten der BH bh. Dieser Index (skbh_7 (kbh)) dient der eindeutigen Identifizierung eines
Knotens in der BH. Jedesmal wenn ein neuer Knoten in bh angefügt wird,
wird dieser Index um eins erhöht.
*/
∀ benutzthierarchie bh: ∃ nat n: n=high_index (bh)
  ∧ ∃ knoten_bh k: ist_in_baum_bh (k,bh)=true
      ∧ skbh_7 (k)=n
      ∧ ¬ ∃ knoten_bh k': ist_in_baum_bh (k',bh)=true
            ∧ skbh_7 (k')>n

/*
index_bh (k,bh) ermittelt in bh den größten Index und schreibt diesen
um eins vergrößerten Index in k.
*/

/*
ist_bearbeitet (k,bh) liefert true, falls es zu dem Knoten k aus bh
einen Knoten k' gibt, so daß k und k' in bestimmten Teilen übereinstimmen.
*/
∀ knoten_bh k: ∀ benutzthierarchie bh:
  ist_bearbeitet (k,bh)=true
  ⇒ ∃ knoten_bh k': k≠k'
      ∧ ist_in_baum_bh (k,bh)=true
      ∧ ist_eq_t_qual (k,k')=true
```

```
/*
ist_eq_t_qual (k,k') liefert true, falls die Knoten k und k' in den
Komponenten Modulname, Prozedurname, Attribut und Dateiname identisch
sind.
*/
∀ knoten_bh k,k': ist_eq_t_qual (k,k')=true
  ⇒ (skbh_2 (k)=skbh_2 (k')
     ∧ skbh_3 (k)=skbh_3 (k')
     ∧ skbh_4 (k)=skbh_4 (k')
     ∧ skbh_6 (k)=skbh_6 (k'))

/*
v_qual_id (kbh,sb) liefert den Stammsatz aus SB sb mit Satzart "pro-
cedure" oder "secondary entry", der in der Identifikationsnummer
(sdt_1 (ssts_2 (s))) mit der füften Komponente (skbh_5 (kbh)) des Kno-
tens kbh übereinstimmt.
*/
∀ knoten_bh kbh: ∀ sb sb: ∃ stammsatz s:
  s=v_qual_id (kbh,sb)
  ∧ (s≠error
    ⇒ (ist_in_baum_sb (s,sb)=true
       ∧ (ssl_6 (ssts_1 (s))=procedure
          ∨ ssl_6 (ssts_1 (s))=secondary entry)
       ∧ sdt_1 (ssts_2 (s))=skbh_5 (kbh)))

/*
gen_knoten_bh_aus_cs (cs,nd) liefert einen Knoten der BH. Dieser Kno-
ten identifiziert das Objekt, das zu dem Call-Stmt cs gehört. nd ist
das Tupel Modulname-Quelldatei, das den Modul charakterisiert, der den
aufgerufenen Entry enthält.
*/

/*
lac (sb,dn) liefert alle Aufrufe von Entries der äußersten Prozedur
des SB sb, in Form von Call-Stmts. Ein Call-Stmt besteht aus dem
Modulnamen, dem Entrynamen, einem Attribut, der Identifikationsummer
des Stammsatzes, der den Aufruf enthält und dem Dateinamen dn.
*/
∀sb sb: ∀ name dn: ∀ callstmt c: ist_enth_lc (c,lac (sb,dn))=true
  ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
     ∧ ssl_1 (ssts_1 (s))=scs_1 (c)
     ∧ sdt_3 (ssts_2 (s))=scs_2 (c)
     ∧ ssl_6 (ssts_1 (s))=scs_3 (c)
     ∧ sdt_1 (ssts_2 (s))=scs_4 (c)
     ∧ scs_5 (c)=dn
     ∧ enth_vater_proc (s,sb)=best_äußerste_proz (sb)
     ∧ ∀ stammsatz s': (ist_in_baum_sb (s',sb)=true
        ∧ ssl_6 (ssts_1 (s))=call
```



```
        ∧ ssl_6 (ssts_1 (s))=callfct
        ∧ ssl_6 (ssts_1 (s))=callext
        ∧ ssl_6 (ssts_1 (s))=callextfct
        ∧ enth_vater_proc (s,sb)=best_äußerste_proz (sb))
       ⇒ (∃ callstmt cs: ist_enth_lc (cs,lac (sb,sn))=true
          ∧ scs_4 (cs)=sdt_1 (ssts_2 (s'))
          ∧ scs_2 (cs)=sdt_3 (ssts_2 (s'))
          ∧ scs_3 (cs)=ssl_6 (ssts_1 (s'))
          ∧ scs_5 (cs)=dn)

/*
aufrufhäufigkeit (lkbh) liefert eine Liste von Knoten der BH, in der
jeweils alle teilweise identischen Knoten aus lkbh zu einem Knoten,
der dann die Anzahl dieser Knoten enthält, zusammengefaßt sind.
*/
∀ l(knoten_bh) lkbh: ∃ l(knoten_bh) l: l=aufrufhäufigkeit (lkbh)
  ∧ (∀ knoten_bh k: ist_enth_lkbh (k,l)=true
     ⇒ (¬ ∃ knoten_bh k': ist_enth_lkbh (k',l)=true
          ∧ ist_eq_t_qual (k,k')=true))
   ∧ anzahl_lkbh (l)≤anzahl_lkbh(lkbh)
   ∧ k=aufrufhäufifkeit_1 (k,lkbh))

/*
aufrufhäufigkeit_1 (k,l) ist eine Hilfsoperation der Operation aufruf-
häufigkeit (l). Diese Hilfsoperation addiert alle teilweise identi-
schen Knoten zu k aus l und schreibt diese Summe in die erste Kompo-
nente von k.
*/
∀ knoten_bh k: ∀ l(knoten_bh) l:
  ∃ knoten_bh k': k'=aufrufhäufigkeit_1 (k,l)
   ∧ ∃ l(knoten_bh) l': ∀ knoten_bh kbh:
      ist_enth_lkbh (kbh,l')=true
      ⇒ (ist_eq_t_qual (kbh,k)=true
         ∧ anzahl_lkbh (l')=skbh_1 (k)
         ∧ ist_enth_lkbh (kbh,l)=true
         ∧ ¬ ∃ knoten_bh kbh': ist_enth_lkbh (kbh',l)=true
                ∧ ist_eq_t_qual (k,kbh')=true
                ∧ ist_enth_lkbh (kbh',l')=false)

/*
suche_obj_zu_call (s,dn,ld) wird mit dem Stammsatz des Aufrufs des
Entry aufgerufen. Da ein interner Entry angesprungen wird, muß die
Prozedur bzw. der Sec. Entry gesucht werden, der zum Aufruf gehört.
Das Ergebnis dieser Operation ist ein Knoten der BH, der sämtliche In-
formation über das aufgerufene Objekt enthält.
*/
∀ stammsatz s: ∀ name dn: ∀ l(dat) ld:
  ∃ knoten_bh k: k=suche_obj_zu_call (s,dn,ld)
```

```
   ∧ (k≠error
     ⇒ k=su_obj_1 (s,enth_vater_proc_beg (s,
            lies_sb (ssl_1 (ssts_1 (s)),lies_dat (dn,ld))),
            lies_sb (ssl_1 (ssts_1 (s)),lies_dat (dn,ld)),dn))

/*
Die Operation su_obj_1 (sc,sv,sb,dn), Hilfsfunktion der Operation
suche_obj_zu_call, liefert, falls sv das zu sc gesuchte Objekt, ist
einen entsprechenden Knoten der BH, ansonsten erfolgt ein Aufruf der
Hilfsoperation su_obj_2. sc ist der Stammsatz des Calls. sv ist der
Stammsatz des innersten Blocks, der den Aufruf enthält. dn ist der Da-
teiname, der Datei, die den durch den SB sb gegebenen Quellmodul ent-
hält.
*/
∀ stammsatz sc,sv: ∀ sb sb: ∀ name dn:
  ∃ knoten_hb k: k=su_obj_1 (sc,sv,sb,dn)
   ∧ (( ssl_6 (ssts_1 (sv))=procedure
        ∧ ist_enth_ln (sdt_3 (ssts_2 (sc)),gen_l_entrypoints (sv,sb))
                                                          =true)
      ⇒ k=erz_kbh (1,ssl_1 (ssts_1 (sv)),gen_l_entrypoints (sv,sb),
                   intproz,sdt_1 (ssts_2 (sv)),dn,0))
     ∨ k=su_obj_2 (lds_sb (sv,sb),sc,sv,sb,sn)

/*
su_obj_2 (lsb,sc,sv,sb,dn) liefert einen Knoten der BH, der dem in sc
aufgerufenem Objekt entspricht.
*/
∀ l(stammsatz) lsb: ∀ stammsatz sc,sv: ∀ sb sb: ∀ name dn:
  ∃ knoten_bh k: k=su_obj_2 (lsb,sc,sv,sb,dn)
   ∧ (k≠error
      ⇒ ((∃ stammsatz s: ist_enth_lsb (s,lsb)=true
           ∧ ssl_6 (ssts_1 (s))=procedure
           ∧ ((ist_enth_ln (sdt_3 (ssts_2 (sc)),
                 gen_l_entrypoints (s,sb))=true
              ∧ k=erz_kbh (1,ssl_1 (ssts_1 (s)),
                   gen_l_entrypoints (sa,sb),intproz,
                   sdt_1 (ssts_2 (s)),sn,0))
            ∨ k=entry_zum_call (s,sc,sb,dn))
         ∨ (ssl_6 (ssts_1 (car_lsb (lsb)))=begin block
            ∧ k=su_obj_2 (cdr_lsb (lsb),sc,sv,sb,dn))
         ∨ (ssl_6 (ssts_1 (car_lsb (lsb)))≠begin block
            ∧ ssl_6 (ssts_1 (car_lsb (lsb)))≠procedure
            ∧ k=su_obj_2 (append_lsb (cdr_lsb (lsb),
                 lds_sb (car_lsb (lsb),sb),sc,sv,sb,dn))
         ∨ (ssl_6 (ssts_1 (car_lsb (lsb)))=procedure
            ∧ ist_enth_ln (sdt_3 (ssts_2 (sc)),
                gen_l_entrypoints (car_lsb (lsb)))=false
            ∧ k=su_obj_2 (cdr_lsb (lsb),sc,sv,sb,dn))
```



```
         ∨ k=su_obj_1 (sc,enth_vater_proc_beg (sv,sb),sb,dn)))

/*
entf_t_id (k,lkbh) entfernt aus lkbh alle zu k teilidentischen Knoten.
*/
∀ knoten_bh k: ∀ l(knoten_bh) lkbh :
  ∃ l(knoten_bh) l: l=entf_t_id (k,lkbh)
   ∧ ¬ ∃ knoten_bh k': ist_enth_lkbh (k',l)=true
         ∧ ist_eq_t_qual (k,k')=true
         ∧ k≠k'

/*
v_qual (c,ld) liefert zum Call-Stmt c den zugehörigen Stammmsatz des
Aufrufs im zugehörigen SB.
*/
∀ callstmt c: ∀ l(dat) ld: ∃ stammsatz s:
  s≠error
   ⇒ (ist_in_baum_sb (s,lies_sb (scs_1 (c),
          lies_dat (scs_5 (c),ld))=true
      ∧ scs_4 (c)=sdt_1 (ssts_2 (s))

/*
entry_zum_call (sc,sv,sb,dn) liefert einen Knoten der BH, der dem
Entry-Stmt zum Aufruf, gegeben durch sc, entspricht. Diese Operation
wird nur aufgerufen, wenn sichergestellt ist, daß es einen Sec. Entry
mit dem Namen des Aufrufs im Teil-SB mit Wurzel sv gibt.
*/
∀ stammsatz sc,sv: ∀ sb sb: ∀ name dn:
  ∃ knoten_bh k: k=entry_zum_call (sc,sv,sb,dn)
   ∧ (k≠error
    ⇒ ∃ stammsatz s: ist_in_baum_sb (s,t_baum_sb (sv,sb))=true
         ∧ ssl_6 (ssts_1 (s))=secondary entry
         ∧ ist_enth_ln (sdt_3 (ssts_2 (sc)),
              gen_l_entry_points (s,sb))=true
         ∧ k=erz_kbh (0,ssl_1 (ssts_1 (s)),
                   gen_l_entrypoints (s,sb),
                   sec. entry,
                   sdt_1 (ssts_2 (s)),dn,0))))

/*
ist_call_eines_sec_entry (sc,sv,sb) liefert true,falls es einen Sec.
Entry mit dem Namen des Aufrufs aus sc im Teil-SB mit Wurzel sv gibt.
Die Operation entspricht in analoger Weise der Operation
entry_zum_call.
*/
∀ stammsatz sc,sv: ∀ sb sb:
  ist_call_eines_sec_entry (sc,sv,sb)=true
   ⇒ ∃ stammsatz s: ist_in_baum_sb (s,t_baum_sb (sv,sb))=true
```

```
      ∧ ssl_6 (ssts_1 (s))=secondary entry
      ∧ ist_enth_ln (sdt_3 (ssts_2 (s)),
           gen_l_entrypoints (s,sb)=)=true

/*
Die Operation op_benutzthierarchie (m,f,ld) liefert die gewünschte BH.
*/
∀ name m: ∀ fenster f: ∀ l(dat) ld:
  ∃ benutzthierarchie bh:
   bh=op_benutzthierarchie (m,f,ld)
   ∧ ∀ knoten_bh k: (ist_in_baum_bh (k,bh)=tru ∧ k≠error)
       ⇒ ∃ knoten_bh k2: ist_in_baum_bh (k1,bh)=true
           ∧ vater_bh (k,bh)=k1
           ∧ ∃ sb sb: ist_enth_lkbh (k,
               lap (lac (sb,skbh_6 (k1),ld))=true
               ∨ ist_enth_lkbh (k,lap (
                    ermittle_calls (k1,ld),ld))=true

/*
best_äußerste_proz (sb) liefert den Stammsatz mit Satzart "procedure",
der in keiner anderen Prozedur enthalten ist.
*/
∀ sb sb: ∃ stammsatz s:   s=best_äußerste_proz (sb)
  ∧ ssl_6 (ssts_1 (s))=procedure
  ∧ ist_enth_in_proz (s,sb)=false

/*
ist_enth_in_proz  (s,sb) liefert true,  falls der Stammsatz s in einer
anderen Prozedur enthalten ist.
*/
∀ sb sb: ∀ stammsatz s:
  ist_enth_in_proz (s,sb)=true
   ⇒ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
       ∧ ssl_6 (ssts_1 (s'))=procedure
       ∧ ist_in_baum_sb (s,t_baum_sb (s',sb))=true

PRIVATE OPS :
anf_bhier:l(knoten_bh) benutzthierarchie knoten_bh
                                  l(dat)-->benutzthierarchie
anfügen_bh:l(knoten_bh) benutzthierarchie knoten_bh
                                    l(dat)-->benutzthierarchie
lap:l(callstmt) l(dat)-->l(knoten_bh)
lac: sb name-->l(callstmt)
gen_l_entrypoints_1:l(stammsatz)-->l(name)
l_aller_entrynamen_1l(stammsatz) sb-->l(name)
ist_bearbeitet:knoten_bh benutzthierarchie-->bool
anzahl_knoten:knoten_bh benutzthierarchie-->nat
anzahl_knoten_1:knoten_bh benutzthierarchie nat-->nat
```



```
ermittle_uo:callstmt l(dat)-->knoten_bh
suche_obj_zu_call:stammsatz name l(dat)-->knoten_bh
su_obj_1:stammsatz stammsatz sb name-->knoten_bh
su_obj_2:l(stammsatz) stammsatz stammsatz sb name-->knoten_bh
l_aller_sec.entrynamen_1:l(stammsatz) sb name-->l(name)
v_qual:callstmt l(dat)-->stammsatz
v_qual_1:callstmt l(stammsatz)-->stammsatz
ist_call_eines_sec.entry:stammsatz stammsatz sb-->bool
ist_call_sec.entry_1:stammsatz l(stammsatz) sb-->bool
ermittle_calls:knoten_bh l(dat)-->l(callstmt)
aufrufhäufigkeit:l(knoten_bh)-->l(knoten_bh)
aufrufhäufigkeit_1:knoten_bh l(knoten_bh)-->knoten_bh
ist_eq_t_qual:knoten_bh knoten_bh-->bool
lac_1:l(stammsatz) sb name-->l(callstmt)
suche_ext_obj_zu_call:callstmt l(dat)-->knoten_bh
entf_t_id:knoten_bh l(knoten_bh)-->l(knoten_bh)
entry_zum_call:stammsatz stammsatz sb name-->knoten_bh
entry_zum_call_1:stammsatz l(stammsatz) sb-->knoten_bh
gen_knoten_bh_aus_cs:callstmt name_dat-->knoten_bh
v_qual_id:knoten_bh sb-->stammsatz
v_qual_id_1:knoten_bh l(stammsatz)-->stammsatz
high_index:benutzthierarchie-->nat
index_bh:knoten_bh benutzthierarchie-->knoten_bh
best_äußerste_proz_1:l(stammsatz) sb-->stammsatz
ist_enth_in_proz:stammsatz sb-->bool

DEFINE OPS :
high_index (bh):=
   case bh is
   *erz_bh (k)-->skbh_7 (k)
   *son_bh (b1,b2)-->
      if ist_gt (high_index (b1),high_index (b2))
        then high_index (b1)
        else high_index (b2)
      fi
  esac ;

index_bh (k,bh):=pkbh_7 (incr (high_index (bh)),k);

op_benutzthierarchie (m,f,ld):=
   let x=lies_sb (m,liefere_eind_modul (m,ld)) in
   let y=name_quelldatei (liefere_eind_modul (m,ld)) in
     case f is
     *alles-->anf_bhier (lap (lac (x,y),ld),
                erz_bh (erz_khb (0,m,l_aller_entrynamen (m,
                                  liefere_eind_modul (m,ld)),modul,
                             sdt_1 (ssts_2 (wurzel_sb (x)),y,0)))))
                erz_kbh (0,m,l_aller_entrynamen (m,
```

```
                              liefere_eind_modul (m,ld)),modul,
                        sdt_1 (ssts_2 (wurzel_sb (x))),y,0))),ld)
    *name_intproz (n)-->
       let z=v_qual_sa (n,procedure,x) in
       let x1=erz_kbh (0,m,gen_l_entrypoints (z,x),
                       intproz,sdt_1 (ssts_2 (z))),y,0) in
       anf_bhier (lap (lac (t_baum_sb (z,x)),y),ld),
                  erz_bh (x1), x1,ld)
    *name_sec_entry (e)-->
       let y1=v_qual_sa (e,sec.entry,x) in
       let y2=erz_kb (0,m,gen_l_entrypoints (y1,x,sec.entry,
                          sdt_1 (ssts_2 (y1)),y,0) in
       anf_bhier (lap (lac (t_baum_sb (
                             enth_vater_proc (y1,x),y,ld),
                  erz_bh (y2),y2,ld)
    esac ;

lac (sb,dn):=lac_1 (preorder_sb (sb),sb,dn);

lac_1 (lsb,sb,dn):=
   case lsb is
   *nil_lsb-->nil_lcs
   *cons_lsb (s,rest)-->
      let x=ssl_6 (ssts_1 (s)) in
       if eq.stammsatz (enth_vater_proz (s,sb),
                        best_äußerste_proz (sb))
         and (eq.charstring (x,call),
              or eq.charstring (x,callext)
              or eq.charstring (x,callfct)
              or eq.charstring (x,callextfct))
         then cons_lc (erz_cs (
                      ssl_1 (ssts_1 (s)),sdt_3 (ssts_2 (s)),
                      x,sdt_3 (ssts_2 (s)),dn),nil_lc)
         else nil_lc
   esac ;

lap (lc,ld):=
   case lc is
   *nil_lc-->nil_lkbh
   *cons_lc (c,rest)-->
     aufrufhäufigkeit (
           cons_lkbh (ermittle_uo (c,ld),lap (rest,ld)))
   esac ;

ermittle_uo (cs,ld):=
   if eq.charstring (scs_3 (cs),call),
      or eq.charstring (scs_3 (cs),callfct)
     then suche_obj_zu_call (v_qual (cs,ld),scs_5 (cs),ld)
```



```
     else if eq.charstring (scs_3 (cs),callext)
             or eq.charstring (scs_3 (cs),callextfct)
            then suche_ext_obj_zu_call (cs,ld)
            else error.knoten_bh
          fi
   fi ;

suche_obj_zu_call (s,dn,ld):=
   let x=lies_sb (ssl_1 (ssts_1 (s)),lies_dat (dn,ld)) in
   let y=enth_vater_proz_beg (s,x) in
    if eq.charstring (ssl_6 (ssts_1 (y)),procedure)
       or eq.charstring (ssl_6 (ssts_1 (y)),begin block)
      then su_obj_1 (s,x,y,dn)
      else error.knoten_bh
   fi ;

su_obj_1 (sc,sv,sb,dn):=
   if eq.charstring (ssl_6 (ssts_1 (sv)),procedure)
      and ist_enth_ln (sdt_3 (ssts_2 (sc)),
                       gen_l_entrypoints (sv,sb)
     then erz_kbh (1,ssl_1 (ssts_1 (sv)),gen_l_entrypoints (sv,sb),
                   intproz,sdt_1 (ssts_2 (sv)),dn,0)
     else su_obj_2 (lds_sb (sv,sb),sc,sv,sb,dn)
   fi ;

su_obj_2 (lsb,sc,sv,sb,dn):=
   case lsb is
   *nil_lsb-->
      if not ist_enth_lsb (sv,lds_sb (wurzel_sb (sb),sb))
        then su_obj_1 (sc,enth_vater_proz_beg (sv,sb),sb,dn))
        else error.knoten_bh
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),begin block)
       then su_obj_2 (rest,sc,sv,sb,dn)
       else
        if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
         then
           if ist_enth_ln (sdt_3 (ssts_2 (sc)),
                           gen_l_entrypoints (s,sb))
             then erz_kbh (1,ssl_1 (ssts_1 (s)),
                           gen_l_entrypoints (s,sb))
                           intproz,sdt_1 (ssts_2 (s)),dn,0)
             else if ist_call_eines_sec.entry (s,sc,sb)
                   then entry_zum_call (s,sc,sb,dn)
                   else su_obj_2 (rest,sc,sv,sb,dn)
                  fi
           fi
         else su_obj_2 (append_lsb (rest,lds_sb (s,sb)),
```

```
                    sc,sv,sb,dn)
      fi
    fi
  esac ;

entf_t_id (k,lk):=
  case lk is
  *nil_lkbh-->nil_lkbh
  *cons_lkbh (k1,rest)-->
    if ist_eq_t_qual )k,k1)
      then entf_t_id (k,rest)
      else cons_lkbh (k1,entf_t_id (k,rest))
    fi
  esac ;

v_qual (cs,ld):=v_qual_1 (preorder_sb (scs_1 (cs),
                          lies_dat (scs_5 (cs),ld));
v_qual_1 (cs,lsb):=
  case lsb is
  *nil_lsb-->error.stammsatz
  *cons_lsb (s,rest)-->
    if eq.nat (scs_4 (cs),sdt_1 (ssts_2 (s))
      then s
      else v_qual_1 (cs,rest)
    fi
  esac ;

ist_call_eines_sec.entry (sp,sc,sb):=
  ist_call_sec.entry_1 (sc,lds_sb (sp,sb),sb);

ist_call_sec.entry_1 (sc,lsb,sb):=
  case lsb is
  *nil_lsb-->false
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),procedure),
       or eq.charstring (ssl_6 (ssts_1 (s)),executable stmt)
      then ist_call_sec.entry_1 (rest,sc)
      else if eq.charstring (ssl_6 (ssts_1 (s)), secondary entry)
           then if ist_enth_ln (sdt_3 (ssts_2 (sc)),
                                gen_l_entrypoints (s,sb))
                  then true
                  else ist_call_sec.entry_1 (rest,sc)
                fi
           else ist_call_sec.entry_1 (
                     rest,lds_sb (s,sb)),sc,sb)
   fi
  esac ;
```



```
entry_zum_call (sc,sp,sb,dn):=
  entry_zum_call_1 (sc,lds_sb (sp,sb),sb,dn);

entry_zum_call_1 (sc,lsb,sb,dn):=
  case lsb is
  *nil_lsb-->error.knoten_bh
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
       or eq.charstring (ssl_6 (ssts_1 (s)),executable stmt)
     then entry_zum_call_1 (sc,rest,sb)
     else
       if eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
         then
          if ist_enth_ln (sdt_3 (ssts_2 (sc)),
                                gen_l_entrypoints (s,sb))
            then erz_kbh (0,ssl_1 (ssts_1 (s)),
                          gen_l_entrypoints (s,sb),
                          sec.entry,sdt_1 (ssts_2 (s)),dn,0)
            else entry_zum_call_1 (sc,rest,sb,dn)
          fi
         else entry_zum_call_1 (sc,
                    append_lsb (rest,lds_sb),sb,dn)
       fi
    fi
  esac ;

anf_bhier (lk,bh,k,ld):=
  if ist_bearbeitet (k,bh):=
    then bh
    else anfügen_bh (lk,bh,k,ld)
  fi ;

anfügen_bh (lk,bh,k,ld):=
  case lk is
  *nil_lkbh-->bh
  *cons_lkbh (k1,rest)-->
    anfügen_bh (rest,anf_bhier (
                        lap (ermittle_calls (k1,ld),ld)
                        anf_bh (index_bh (k1,bh),bh),
                        index_bh (k1,bh),ld),
                k,ld)
  esac ;

ermittle_calls (k,ld):=
  let x=lies_sb (skbh_2 (k),lies_dat (skbh_6 (k),ld) in
  let y=v_qual_id (k,x) in
   if eq.charstring (skbh_4 (k),intproz)
      or eq.charstring (skbh_4 (k),sec.entry)
```

```
      then lac (t_baum_sb (y),skbh_6 (k))
      else if eq.charstring (skbh_4 (K),sec.entry)
             then lac (t_baum_sb (enth_vater_proz (y),skbh_6 (k))
             else
              if eq.charstring (skbh_4 (k),extproz)
                 or eq.charstring (skbh_4 (k),ext sec.entry)
                then lac (x,skbh_6 (k))
                else error.l (callstmt)
              fi
           fi
   fi ;

aufrufhäufigkeit (lk):=
   case lk is
   *nil_lkbh-->nil_lkbh
   *cons_lkbh (k,rest)-->
     cons_lkbh (aurufhäufigkeit_1 (k,rest),
                aufrufhäufigkeit (entf_t_id (k,rest)
   esac ;

aufrufhäufigkeit_1 (k,lk):=
   case lk is
   *nil_lkbh-->k
   *cons_lkbh (k1,rest)-->
      if ist_eq_t_qual (k,k1)
        then aufrufhäufigkeit_1 (
               pkbh_1 (incr (skbh_1 (k)),k),rest)
        else aufrufhüfigkeit_1 (k,rest)
      fi
   esac ;

ist_eq_t_qual (k1,k2):=
   if eq.name (skbh_2 (k1),skbh_2 (k2))
      and eq.l (name) (skbh_3 (k1),skbh_3 (k))
      and eq.charstring (skbh_4 (k1),skbh_4 (k))
      and eq.name (skbh_6 (k1),skbh_6 (k2))
     then true
     else false
   fi ;

gen_knoten_bh_aus_cs (cs,nd):=
  let x=l_aller_entrynamen (snd_1 (nd),snd_2 (nd)) in
  let y=l_aller_sec.entrynamen (snd_1 (nd),scs_2 (cs),snd_2 (nd)) in
  if ist_enth_ln (scs_2 (cs),x)
    then erz_kbh (0,snd_1 (nd),x,extproz,0,
                  name_quelldatei (snd_2 (nd)),0)
    else erz_kbh (0,snd_1 (nd),y,ext sec.entry,0,
                  name_quelldatei (snd_2 (nd)),0)
```



```
  fi ;

suche_ext_obj_zu_call (cs,ld):=
  gen_knoten_bh_aus_cs (cs,
    liefere_eind_entry (scs_1 (cs),scs_2 (cs),ld));

ist_bearbeitet (k,bh):=
   if ist_gt (anzahl_knoten (k,bh),succ (0))
     then true
     else false
   fi ;

anzahl_knoten (k,bh):=
   anzahl_knoten_1 (k,bh,0);

anzahl_knoten_1 (k,bh,n):=
   case bh is
   *erz_bh (k1)--> if ist_eq_t_qual (k1,k)
                     then incr (n)
                     else n
                   fi
   *son_bh (t1,t2)-->add (anzahl_knoten_1 (k,t1,n)
                          anzahl_knoten_1 (k,t2,n))
   esac ;

gen_l_entrypoints (s,sb):=
  if eq.charstring (ssl_6 (ssts_1 (s)),proc stmt)
     or eq.charstring (ssl_6 (ssts_1 (s)),entrystmt)
     or eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
     or eq.charstring (ssl_6 (ssts_1 (s)),procedure)
    then gen_l_entrypoints_1 (gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb)
    else error.l (name)
  fi ;

gen_l_entrypoints_1 (lsb):=
   case lsb is
   *nil_lsb-->nil_ln
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
       then cons_ln (sdt_3 (ssts_2 (s)),gen_l_entrypoints_1 (rest))
       else gen_l_entrypoints_1 (rest)
     fi
  esac ;

v_qual_id (k,sb):=
   v_qual_id_1 (k, preorder_sb (sb));

v_qual_id_1 (kbh,lsb):=
```

```
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
     if eq.nat (sdt_1 (ssts_2 (s)),skbh_5 (kbh)
        and (eq.charstring (ssl_6 (ssts_1 (s)),procedure)
              or eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
      then s
      else v_qual_id_1 (kbh,rest)
     fi
   esac ;

l_aller_entrynamen (n,d):=
   l_aller_entrynamen_1 (preorder_sb (lies_sb (n,d)),
                         lies_sb (n,d));

l_aller_entrynamen_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->error.l (name)
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
        and eq.stammsatz (vater_sb (s),wurzel_sb (sb))
      then gen_l_entrypoints (s,sb)
      else l_aller_entrynamen_1 (rest,sb)
     fi
   esac ;

l_aller_sec.entrynamen (n,e,d):=
   l_aller_sec.entrynamen_1 (preorder_sb (lies_sb (n,d)),
                             lies_sb (n,d),e);

l_aller_sec.entrynamen_1 (lsb,sb,e):=
   case lsb is
   *nil_lsb-->error.l (name)
   *cons_lsb (s,rest)-->
     if (eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
         or eq.charstring (ssl_6 (ssts_1 (s)),entry stmt))
        and eq.name (sdt_3 (ssts_2 (s)),e)
        and eq.stammsatz (vater_sb (enth_vater_proz (s,sb),sb)),
                          wurzel_sb (sb))
      then gen_l_entrypoints (s,sb)
      else l_aller_sec.entrynamen_1 (rest,sb)
     fi
   esac ;

best_äußerste_proz (sb):=
   best_äußerste_proz_1 (preorder_sb (sb),sb);

best_äußerste_proz_1 (lsb,sb):=
```



```
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
        and not ist_enth_in_proz (s,sb)
      then s
      else best_äußerste_proz_1 (rest,sb)
     fi
   esac ;

ist_enth_in_proz (s,sb):=
   if eq.stammsatz (s,wurzel_sb (sb))
     then false
     else if eq.stammsatz (vater_sb (s,sb),wurzel_sb (sb))
             and eq.charstring (ssl_6 (ssts_1 (s)),procedure)
           then true
           else if eq.stammsatz (vater_sb (s,sb),wurzel_sb (sb))
                   and not eq.charstring (ssl_6 (ssts_1 (
                                          wurzel_sb  (sb))),procedure)
                 then true
                 else if eq.charstring (ssl_6 (ssts_1 (
                                         vater_sb (s,sb),procedure)
                         then true
                         else ist_enth_in_proz (vater_sb (s,sb),sb)
                        fi
                fi
        fi
   fi ;

ENDSPEC
```

3.2.4. Einfache Statistik

Diese Auswertefunktion ist in sechs Teilfunktionen gegliedert. Die Spezifikation Einfache Statistik ist eine Zusammenfassung der Teilfunktionen zu einer Gesamtfunktion.

Der Zweck der Funktion besteht darin, dem Benutzer einfache Qualitäts- und Quantitätsaussagen über seine Quellprogramme zu liefern.

3.2.4.1. Allgemein

3.2.4.1.1. Beschreibung

Diese Unterfunktion analysiert ein gesamtes Quellmodul oder eine interne Prozedur eines Quellmoduls. Beides wird hier als Untersuchungsobjekt (UO) bezeichnet.

Folgende Informationen sind in der Ausgabe dieser Funktion zu finden:

1. Der Kopf der Ausgabe besteht aus dem Namen des UO und dessen Länge in Statements.
2. Die im UO aufgerufenen externen Prozeduren und/oder externen Sec. Entries werden namentlich gelistet. Die Anzahl der externen Prozeduren wird addiert und ausgegeben.
   Die Häufigkeit der externen Prozeduren ist definiert als der Prozentsatz der Anzahl der Aufrufe der externen Prozeduren im Verhältnis zur Anzahl der Statements im UO. Diese Zahl ist zu ermitteln und auszugeben.
3. Für jede im UO enthaltene interne Prozedur ist der Name, die Länge in Statements, die Länge in Zeilen, die Namen der Parameter und die Häufigkeit, hier definiert als der Prozentsatz der Anzahl der Statements der internen Prozedur im Verhältnis zur Anzahl der Statements des UO, auszugeben.

Bemerkung: Eine Prozedur kann mehrere Entrynamen haben. Aber es wird nur der äußerste berücksichtigt.

Beispiel: P1 : P2 : P3 : PROCEDURE;

P1 ist dann der Name dieser Prozedur.

Beispiel: Die Ausgabe hat somit folgenden Aufbau:

Name UO	Laenge in Statements	Kopf

Name1	Name2	.....	externe Prozeduren
Anzahl	Haeufigkeit		

Name1	Name2	.....	interne Prozeduren
Parameternamen			
Laenge in Zeilen			
Laenge in Stmts			
Haeufigkeit			
Anzahl			

3.2.4.1.2. Spezifikation Allgemein

```
INSTANTIATE Tupel to As_kopf
ACTUALIZE : #x1 by Name
            #x2 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by nat
RENAME : SORTS : tupel by as_kopf
           OPS : erz_tup by erz_ask
END


INSTANTIATE 3_Tupel to As_Ext_Proz
ACTUALIZE : #x1 by L(Name)
            #x2 by Nat
            #x3 by Nat
            SORTS : #x1.elem by l(name)
                    #x2.elem by nat
```

```
                #x3.elem by nat
RENAME : SORTS : 3_tupel by as_ext_proz
         OPS : erz_drei by erz_ase
END


INSTANTIATE 5_Tupel to Inf_Int_Proz
ACTUALIZE : #x1 by Name
            #x2 by L(Name)
            #x3, #x4, #x5 by Nat
            SORTS : #x1.elem by Name
                    #x2.elem by l(name)
                    #x3.elem, #x4.elem, #x5.elem by nat
RENAME : SORTS : 5_tupel by inf_int_proz
         OPS : erz_fünf by erz_ii
END


INSTANTIATE Liste to L(Inf_Int_proz)
ACTUALIZE : #x by Inf_Int_Proz
            SORTS : #x.elem by inf_int_proz
RENAME : SORTS : liste by l(inf_int_proz)
         OPS : nil by nil_lii
END


INSTANTIATE Tupel to As_Int_Proz
ACTUALIZE : #x1 by L(Inf_Int_Proz)
            #x2 by Nat
            SORTS : #x1.elem by l(inf_int_proz)
                    #x2.elem by nat
RENAME : SORTS : tupel by as_int_proz
         OPS : erz_tup by erz_asi
END


INSTANTIATE 3_Tupel to Allg_stat
ACTUALIZE : #x1 by As_Kopf
            #x2 by As_Ext_Proz
            #x3 by As_Int_Proz
            SORTS : #x1.elem by as_kopf
                    #x2.elem by as_ext_proz
                    #x3.elem by as_int_proz
RENAME : SORTS : 3_tupel by allg_stat
```



```
         OPS : erz_drei by erz_as
END


SSPEZ ALLGEMEIN

USE SSPECS : Schnittstelle, L(Name), Fenster,
             L(Name_Nat), Allg_Stat, Benutzthierarchie

PUBLIC OPS :
op_allgemein:name fenster l(dat)-->allg_stat
diff_zp:zahlenpaar-->nat
gen_l_name_form_par:name_nat sb-->l(name)
v_qual_proz_name_stmtnr:name_nat sb-->stammsatz
gen_l_int_proz:sb-->l(name_nat)

PROPERTIES :
/*
   Die nach außen angebotene Operation op_allgemein unterscheidet die
beiden Fälle, Analyse des gesamten Quellmoduls und Analyse einer in-
ternen Prozedur.
   Im ersten Fall wird die Operation allgemein_1 mit dem Namen des
Quellmoduls, der Liste der Quelldateien und einer initialisierten Aus-
gabe, die an Information nur den Namen des UO enthält, aufgerufen.
   Im zweiten Fall wird die Operation allgemein_2 mit dem Namen des
Quellmoduls, dem Namen der internen Prozedur, der Liste der
Quelldateien und analog zum ersten Fall einer initialisierten Ausgabe
aufgerufen.

   Die Operationen allgemein_1 und allgemein_2 rufen die gleichen
Operationen auf, allerdings mit unterschiedlichen aktuellen Parame-
tern. Die Operationen werden jeweils mit dem Teil des SB, der auch
tatsächlich zu analysieren ist, aufgerufen. Im ersten Fall ist dies
der vollständige SB, im zweiten Fall der Teil des SB, der der internen
Prozedur, die analysiert werden soll, entspricht.

   Dem Aufbau der Ausgabe entsprechend werden drei ineinander ge-
schachtelte Operationen (abs_int_proz, abs_ext_proz,abs_kopf (abs für
Abschnitt)) aufgerufen, die in ihrer Reihenfolge jeweils einen Ab-
schnitt der Ausgabe mit Information ausfüllen. Dabei ist diese Reihen-
folge wesentlich, da die äußeren Operationen auf die Information zu-
greifen, die durch die inneren Operationen geliefert werden.

   abs_kopf ermittelt aus dem SB die Stmtlänge des UO und trägt diese
Zahl in das entsprechende Feld der Ausgabe ein. Der Stmtbereich ist im
Stammsatz der Wurzel des SB vermerkt. Die Differenz der oberen und un-
teren Grenze ergibt die Stmtlänge.
```

abs_ext_proz füllt den zweiten Abschnitt der Ausgabe. Im übergebenen SB wird die Liste der Aufrufe von externen Prozeduren und externen Sec. Entries ermittelt. Aus der Anzahl der Elemente dieser Namensliste und der Stmtlänge des UO wird die Häufigkeit errechnet. Die Anzahl der unterschiedlichen externen Objekte ergibt sich, indem die doppelt vorkommenden Namen gestrichen werden und die Listenelemente dann gezählt werden. Die Operation namen_ext_proz trägt dann diese Liste in die Ausgabe ein.

abs_int_proz ermittelt die Informationen über die im UO enthaltenen internen Prozeduren. Zu beachten ist, daß das UO nicht selbst als interne Prozedur ausgewertet wird, da nur die im UO enthaltenen internn Prozeduren auszuwerten sind.
gen_l_int_proz liefert die Namen und Stmtnummern, der internen Prozeduren.
gen_l_form_par liefert die Liste der Namen der formalen Parameter einer Prozedur.
Sämtliche Informationen über eine interne Prozedur werden durch die Operation abs_int_proz_2 zusammengetragen.
abs_int_proz trägt die Liste der Informationen der internen Prozeduren in den entsprechenden Abschnitt der Ausgabe ein.
*/

```
/*
v_qual_proc_name_stmtnr (ns,sb) liefert zum Tupel Name-Stmtnr einer
Prozedur den vollqualifizierten Stammsatz aus dem SB sb.
*/
∀ name_nat ns: ∀ sb sb: ∃ stammsatz s:
  s=v_qual_proc-name_stmtnr (ns,sb)
  ∧ ist_in_baum_sb (s,sb)=true
  ∧ ssl_6 (ssts_1 (s))=procedure
  ∧ sdt_3 (ssts_2 (s))=sni_1 (ns)
  ∧ ssl_7 (ssts_1 (s))=sni_2 (ns)

/*
gen_l_int_proz (sb) liefert die Tupel Name-Stmtnr aller im SB sb
deklarierten Prozeduren mit Ausnahme der äußersten Prozedur.
*/
∀ sb sb: ∀ name_nat ns:
  ist_enth_lni (ns,gen_l_int_proz (sb))=true
  ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
      ∧ ssl_6 (ssts_1 (s))=procedure
      ∧ ssl_7 (ssts_1 (s))=sni_2 (ns)
      ∧ sdt_3 (ssts_2 (s))=sni_1 (ns)
      ∧ best_äußerste_proz (sb)≠s
      ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
            ∧ ssl_6 (ssts_1 (s'))=procedure
            ∧ s'≠best_äußerste_proz (sb)
```



```
            ∧ ist_enth_lni (erz_ni (sdt_3 (ssts_2 (s')),
                                    ssl_7 (ssts_1 (s'))),
                gen_l_int_proz (sb))=false

/*
gen_l_name_form_par (ns,sb) liefert die Namen der formalen Parameter,
derr durch das Tupel ns eindeutig gekennzeichneten Prozedur.
*/
∀ name_nat ns: ∀ sb sb: ∃ l(name) ln:
  ln=gen_l_name_form_par (ns,sb)
  ∀ name n: ist_enth_ln (n,ln)=true
   ⇒ ∃ stammsatzs1,s2: ist_in_baum_sb (s1,sb)=true
        ∧ ist_in_baum_sb (s2,sb)=true
        ∧ ssl_6 (ssts_1 (s1))=variable identifier
        ∧ n=sdt_3 (ssts_2 (s1)
        ∧ ssl_6 (ssts_1 (s2))=parameter list
        ∧ ssl_7 (ssts_1 (s2))=sni_2 (ns)
        ∧ ist_in_baum_sb (s1,t_baum_sb (s2,sb))=true

/*
l_ext_calls (sb) liefert alle Namen, der im SB sb aufgerufenen externen Entries.
*/
∀ sb sb: ∃ l(nmae) ln: ln=l_ext_calls (sb)
  ∧ ∀ name n: ist_enth_ln (n,ln)=true
      ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
           ∧ (ssl_6 (ssts_1 (s))=callext
              ∨ ssl_6 (ssts_1 (s))=callextfct )
           ∧ sdt_3 (ssts_2 (s))=n
           ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
                ∧ (ssl_6 (ssts_1 (s'))=callext
                  ∨ ssl_6 (ssts_1 (s'))=callextfct)
                ∧ ist_enth_ln (sdt_3 (ssts_2 (s')),ln)=false
```

PRIVATE OPS :

```
op_allgemein_1:name l(dat) allg_stat-->allg_stat
op_allgemein_2:name name l(dat) allg_stat-->allg_stat
abs_kopf: sb allg_stat-->allg_statallg_stat
abs_ext_proz:sb allg_stat-->allg_stat
abs_int_proz:sb allg_stat-->allg_stat
abs_int_proz_1:l(name_nat) sb allg_stat-->l(inf_int_proz)
abs_int_proz_2:name_nat sb allg_stat-->inf_int_proz
l_ext_calls:sb-->l(name)
l_ext_calls_1:l(stammsatz)-->l(name)
häufigkeit_ext_proz:l(name) allg_stat-->allg_stat
häufigkeit_ext_proz_1:nat allg_stat-->allg_stat
häufigkeit_ext_proz_2:nat nat-->allg_stat
anzahl_ext_proz:l(name) allg_stat-->allg_stat
```

```
namen_ext_proz:l(name) allg_stat-->allg_stat
gen_l_name_form_par_1:l(stammsatz) sb-->l(name)
gen_l_name_form_par_2:l(stammsatz)-->l(name)
v_qual_proz_name_stmtnr_1:name_nat l(stammsatz)-->stammsatz
gen_l_int_proz_1:l(stammsatz) sb-->l(name_nat)

DEFINE OPS :
op_allgemein (n,f,ld):=
   case f is
   *alles-->op_allgemein_1 (n,ld,erz_as (
               erz_ask (n,0),
               erz_ase (nil_ln,0,0),
               erz_asi (nil_lii,0)))
   *name_int_proz (p)-->op_allgemein_2 (n,p,ld,erz_as (
                        erz_ask (p,0),
                        erz_ase (nil_ln,0,0),
                        erz_asi (nil_lii,0)))
   *name_sec.entry (e)-->error.allg_stat
   esac ;

op_allgemein_1 (n,ld,as):=
   let x=lies_sb (n,liefere_eind_modul (n,ld)) in
      abs_int_proz (x,abs_ext_proz (x,abs_kopf (x,as)));

op_allgemein_2 (m,n,ld,as):=
   let x=lies_sb (m,liefere_eind_modul (m,ld)) in
   let y=t_baum_sb (v_qual_sa (n,procedure,x),x) in
       abs_int_proz (abs_ext_proz (y,abs_kopf (y,as)));

häufigkeit_ext_proz (ln,as):=
   häufigkeit_ext_proz_1 (anzahl_ln (ln,as));

häufigkeit_ext_proz_1 (i,as):=
  häufigkeit_ext_proz_2 (i,sask_2 (sas_1 (sa)),as);

häufigkeit_ext_proz_2 (i1,i2,as):=
   pas_2 (pase_3 (div (mult (i1,100),i2),sas_2),as);

anzahl_ext_proz (ln,as):=
   pas_2 (pase_3 (anzahl_ln (lösche_doppelte_elem_ln (ln)),
                  sas_2),as);

namen_ext_proz (ln,as):=
   pas_2 (pas_1 (lösche_doppelte_elem_ln (ln),sas_2),as);

abs_kopf (sb,as):=
   pas_1 (pas_2 (diff_zp (sdt_5 (ssts_2 (wurzel_sb (sb)))),
                 sas_1 (as)), as);
```



```
diff_zp (zp):=
   sub (szp_2 (zp),szp_1 (zp));

abs_ext_proz (sb,as):=
   let x=l_ext_calls (sb) in
  namen_ext_proz (x,anzahl_ext_proz (x,
                  häufigkeit_ext_proz (x,as)));

l_ext_calls (sb):=l_ext_calls (preorder_sb (sb));

l_ext_calls_1 (lsb):=
   case lsb is
   *nil_lsb-->nil_ln
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),callext)
        or eq.charstring (ssl_6 (ssts_1 (s)),callextfct)
       then cons_ln (sdt_3 (sts_2 (s)),l_ext_calls_1 (rest))
       else l_ext_calls_1 (rest)
     fi
   esac ;

abs_int_proz (sb,as):=
   let x=abs_int_proz_1 (gen_l_int_proz (sb),sb,as) in
     sas_3 (pasi_1 (x,pasi_2 (anzahl_lii (x sas_3 (as))),as);

abs_int_proz_1 (lni,sb,as):=
   case lni is
   *nil_lni-->nil_lii
   *cons_lni (n,rest)-->
     cons_lii (abs_int_proz_2 (n,sb,as),
               abs_int_proz_1 (rest,sb,as))
   esac ;

abs_int_proz_2 (ni,sb,as):=
   let x=v_qual_proz_name_stmtnr (sni_1 (ni),sni_2 (ni),sb) in
   let y=diff_zp (sdt_5 (ssts_2 (x)) in
   erz_ii (sni_1 (ni),
           gen_l_name_form_par (ni,sb),
           diff_zp (sdt_6 (ssts_2 (x)),
           div (mult (y,100),sask_2 (sas_1 (as))))
   esac ;

gen_l_name_form_par (ni,sb):=
   gen_l_name_form_par_1 (
       lds_sb (v_qual_proz_name_stmtnr (ni,sb),sb),sb));

gen_l_name_form_par_1 (lsb,sb):=
   case lsb is
```

```
   *nil_lsb-->nil_ln
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),parameter list)
        and eq.nat (ssl_7 (ssts_1 (s)),sni_2 (ni))
       then gen_l_name_form_par_2 (
              preorder_sb (t_baum_sb (s,sb)))
       else gen_l_name_form_par_1 (rest,sb)
     fi
   esac ;

gen_l_name_form_par_2 (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_ln
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),variable identifier)
       then cons_ln (sdt_3 (ssts_2 (s)),
                     gen_l_name_form_par_2 (rest))
       else gen_l_name_form_par_2 (rest)
     fi
   esac ;

v_qual_proz_name_stmtnr (ni,sb):=
   v_qual_proz_name_stmtnr_1 (ni,preorder_sb (sb));

v_qual_proz_name_stmtnr_1 (ni,lsb):=
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
        and eq.name (sdt_3 (ssts_2 (s)),sni_1 (ni))
        and eq.nat (ssl_7 (ssts_1 (s)),sni_2 (ni))
       then s
       else v_qual_proz_name_stmtnr_1 (ni,rest)
     fi
   esac ;

gen_l_int_proz (sb):=
   gen_l_int_proz_1 (preorder_sb (sb),sb);

gen_l_int_proz_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lni
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts1 (s)),procedure)
        and not eq.stammsatz (s,best_äußerste_proz (sb))
       then cons_lni (erz_ni (sdt_3 (ssts_2 (s)),
                              ssl_7 (ssts_1 (s))),
                      gen_l_int_proz_1 (rest,sb))
```



```
       else gen_l_int_proz_1 (rest,sb)
     fi
   esac ;

ENDSPEC
```

3.2.4.2. Anweisung

3.2.4.2.1. Beschreibung

Diese Teilfunktion analysiert einen Quellmodul (UO) bezüglich der vorkommenden Anweisungen. Die Anweisungen für SPL 3 sind wie folgt gruppiert:

Gruppe	Anweisung
Steuerung	Do, End, Goto, If, While
Blocksteuerung	Begin, Call, Entry, Code, Procedure, Return, Null
Zuweisung	Assignment

Der Benutzer kann Informationen über eine Gruppe oder eine einzelne Anweisung verlangen. Die Eingabe dieser Auswertefunktion besteht also entweder aus dem Gruppennamen oder einer einzelnen Anweisung. Bei Angabe einer Gruppe sind alle Anweisungen dieser Gruppe auszuwerten.

Die Unterfunktion Anweisung liefert an Information:

1. Name des UO
2. Anweisungsgruppe. Wird eine einzelne Anweisung vom Benutzer eingegeben, ist die Anweisungsgruppe dieser Anweisung zuzuordnen und hier einzutragen. Im anderen Fall wird die Anweisungsgruppe direkt eingetragen.
3. Die dritte Komponente der Ausgabe besteht aus einer Liste, die nur ein Element enthält, falls nur eine einzelne Anweisung auszuwerten ist, oder mehrere Elemente, wenn alle Anweisungen einer bestimmten Gruppe auszuwerten sind. Die Elemente dieser Liste haben folgenden Aufbau:
   1. Name der Anweisung
   2. Liste von jeweils vier Zahlen, die den jeweiligen Ort des Auftretens der Anweisung im UO kennzeichnen. Diese Zahlen entsprechen der Reihenfolge nach der Statementnummer, der Zeilennummer, der Includeaufrufnummer und der Includezeilennummer.
   3. Es wird der Prozentanteil der Anzahl der Anweisungen im Verhältnis zur Statementlänge des UO errechnet.
   4. Es wird der Prozentanteil der Anzahl der Anweisungen, die mit einem Label versehen sind, im Verhältnis der Gesamtzahl der Anweisungen ermittelt.

Beispiel: Die Ausgabe hat somit folgendes Aussehen:

<table>
<tr><td colspan="3">Name UO</td></tr>
<tr><td colspan="3">Anweisungsgruppe</td></tr>
<tr><td>Anweisung 1</td><td rowspan="4">Anweisung 2</td><td rowspan="4">.....</td></tr>
<tr><td>n11 n12 n13 n14
n21 n22 n23 n24
.....</td></tr>
<tr><td>% Anteil</td></tr>
<tr><td>% Anteil Label</td></tr>
</table>

3.2.4.2.2. Spezifikation Anweisung

```
SSPEC ANWEISUNG_ODER_GRUPPE

USE SSPECS : Bool

PUBLIC SORTS : anw_grp

PUBLIC OPS :
steuerung, blocksteuerung, zuweisung:-->anw_grp
do, end, goto, if, while:-->anw_grp
begin, call, entry, code, procedure, return, null:-->anw_grp
assignment:-->anw_grp

CONSTRUCTORS :
*steuerung, *blocksteuerung, *zuweisung
*do, *end, *goto, *if, *while
*begin, *call, *entry, *code, *procedure, *return, *null
*assignment

ENDSPEC


INSTANTIATE Liste to L(Nat)
ACTUALIZE : #x by Nat
            SORTS : #x.elem by nat
RENAME : SORTS : liste by l(nat)
          OPS : nil by nil_lnat
END


INSTANTIATE 4_Tupel to Vier_Zahlen
ACTUALIZE : #x1, #x2, #x3, #x4 by Nat
            SORTS : #x1.elem, #x2.elem by nat
                    #x3.elem, #x4.elem by nat
RENAME : SORTS : 4_tupel by vier_zahlen
          OPS : erz_vier by erz_vz
END


INSTANTIATE Liste to L(Vier_Zahlen)
ACTUALIZE : #x by Vier_Zahlen
            SORTS : #x.elem by vier_zahlen
RENAME : SORTS : liste by l(vier_zahlen)
          OPS : nil by nil_lvz
```



```
END


INSTANTIATE 4_Tupel to Anws_Inf
ACTUALIZE : #x1 by Anweisung_oder_Gruppe
            #x2 by L(Vier_Zahlen)
            #x3, #x4 by nat
            SORTS : #x1.elem by anw_grp
                    #x2.elem by l(vier_zahlen)
                    #x3.elem, #x4.elem by nat
RENAME : SORTS : 4_tupel by anws_inf
          OPS : erz_vier by erz_ai
END


INSTANTIATE Liste to L(Anws_Inf)
ACTUALIZE : #x by Anws_Inf
            SORTS : #x.elem by anws_inf
RENAME : SORTS : liste by l(anws_inf)
          OPS : nil by nil_lai
END


INSTANTIATE 3_Tupel to Anws_Stat
ACTUALIZE : #x1 by Name
            #x2 by Anweisung_Oder_Gruppe
            #x3 by L(Anws_Inf)
            SORTS : #x1.elem by name
                    #x2.elem by anw_grp
                    #x3.elem by l(anws_inf)
RENAME : SORTS : 3_tupel by anws_stat
          OPS : erz_drei by erz_anws
END


SSPEC ANWEISUNG

USE SSPECS : Schnittstelle, Anweisung_oder_Gruppe,
             Anws_Stat, Allgemein, L(Nat)


PUBLIC OPS :
op_anweisung:name l(dat) anw_grp-->anws_stat
```

select_ort:stammsatz-->vier_zahlen

PROPERTIES :
/*
Die nach außen angebotene Operation op_anweisung liefert die beschriebene Ausgabe mittels der Operationen zuordnung und gen_l_anws_inf.

zuordnung liefert für eine Anweisung die zugehörige Gruppe oder, falls als Argument eine Anweisungsgruppe vorkommt, diese Anweisungsgruppe.

gen_l_anws_inf erzeugt die Liste, deren Elemente aus den Informationen bestehen,die für jede einzelne Anweisung aus dem SB gewonnen wurden.

Im Falle einer einzelnen Anweisung wird die Operation op_anw_inf aufgerufen, die die verlangte Information für eine Anweisungsgruppe liefert. Das Ergebnis dieser Operation ist eine einelementige Liste.

Im Falle einer Anweisungsgruppe wird eine Liste erzeugt, deren Elemente Operationsaufrufe der Operation op_anw_inf mit den entsprechenden Anweisungen sind. Die Operation op_anw_inf erzeugt ein 4_Tupel mit den Komponenten:

1. Name der Anweisung
2. dem Ergebnis der Operation ort_des_auftretens. D.i. eine Liste, deren Elemente aus je vier Zahlen, der Stmtnr, der Linenr, der Includeaufrufnr und der Includelinenr besteht. Die Operation ort_des_auftretens durchsucht den SB nach Stammsätzen, die die Satzart der jeweiligen Anweisung enthalten. Aus jedem gefundenen Stammsatz werden mittels der Operation select_ort die vier gesuchten Zahlen entnommen.
   Die Operation change_to_charstring, die in der Operation ort_1, der Hilfsfunktion der Operation select_ort aufgerufen wird, wandelt ein Element der Sorte anw_grp in einen entsprechenden charstring um. Dies ist notwendig, da das Feld Satzart im SB von der Sorte charstring ist.
3. dem Ergebnis der Operation %angabe. Die Operation %angabe errechnet aus der Anzahl der Elemente der Operation ort_des_auftretens und der Stmtlänge des UO die Häufigkeit des Vorkommens.
4. Dem %-Anteil der Anweisungen, die mit einem Label versehen sind. Diese Zahl errechnet sich wie folgt.
   1. Es wird eine Liste der Stmtnr der Stmts angelegt, in denen die bestimmte Anweisung vorkommt. Dies leistet die Operation %_anteil_1.
   2. Im SB werden alle Stammsätze mit Satzart "label identifier" gesucht. Sofern die Stmtnr dieser Stammsätze in der oben angegebenen Liste der Stmtnr vorkommt, hat man ein Vorkommen einer bestimmten Anweisung, das mit einem Label versehen ist, gefunden. Diese Vorkommen werden gezählt.

   Da eine Anweisung mehrere Label haben kann, für die %-Angabe aber nur ein Label wesentlich ist, wird die Stmtnr dieser Anweisung aus der Liste der Stmtnrn entfernt. Dies leistet die Operation %_anteil_2.
3. Die Operation %_anteil_label benutzt die Operation anzahl_lvz zum Zählen der verschiedenen Vorkommen der Anweisungen. Der %-Anteil errechnet sich dann aus der Anzahl der mit einem Label versehenen Anweisungen und der Gesamtzahl der Vorkommen.

*/

```
/*
ort_d_auftretens (ag,sb) liefert die Liste mit den Orten des Auftretens der Anweisung ag im durch sb gegebenen Quellmodul.
*/
∀ anw_grp ag: ∀ sb sb: ∃ l(vier_zahlen) l:
  l=ort_d_auftretens (ag,ab)
  ∧ ∀ vier_zahlen v: ist_enth_lvz (v,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ ssl_6 (ssts_1 (s))=change_to_charstring (ag)
          ∧ v=select_ort (s)
          ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
                 ∧ ssl_6 (ssts_1 (s'))=change_to_charstring (ag)
                 ∧ ist_enth_lvz (select_ort (s'),l)=false

/*
%_anteil_1 (ag,lsb) liefert die Liste der Stmtnrn aller Stammsätze aus lsb, die die Satzart "ag" haben.
*/
∀ anw_grp ag: ∀ l(stammsatz) lsb: ∃ l(nat) l:
  l=%_anteil_1 (ag,lsb)
  ∧ ∀ nat n: ist_enth_lnat (n,l)=true
     ⇒ ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
          ∧ ssl_6 (ssts_1 (s))=change_to_charstring (ag)
          ∧ ssl_7 (ssts_1 (s))=n
          ∧ ¬ ∃ stammsatz s': ist_enth_lsb (s',lsb)=true
                 ∧ ssl_6 (ssts_1 (s'))=change_to_charstring (ag)
                 ∧ ist_enth_lnat (ssl_7 (ssts_1 (s')),l)=false

/*
%_anteil_2 (lsb,lnat,n) liefert die Anzahl der Stammsätze aus lsb, die mit einem Label versehen sind. Die Operation wird mit n=0 aufgerufen.
*/
∀ l(stammsatz)lsb: ∀ l(nat) lnat: ∀ nat n:
  ∃ nat n': n'=%_anteil_2 (lsb,lnat,n)
  ∧ ∃ l(stammsatz) l: anzahl_lsb (l)=n'
     ∧ ∀ stammsatz s: ist_enth_lsb (s,l)=true
        ⇒ (ssl_6 (ssts_1 (s))=label identifier
           ∧ ist_enth_lnat (ssl_7 (ssts_1 (s)),lnat)=true
```

```
            ∧ ¬ ∃ stammsatz s': ist_enth_lsb (s',l)=true
                  ∧ ssl_7 (ssts_1 (s))=ssl_7 (ssts_1 (s'))
                  ∧ s≠s')

PRIVATE OPS :
zuordnung:anw_grp-->anw_grp
gen_l_anws_inf:anw_grp sb-->l(anws_inf)
op_anws_inf:anw_grp sb-->anws_inf
ort_d_auftretens:anw_grp sb-->l(vier_zahlen)
%_angabe:anw_grp sb-->nat
change_to_charstring:anw_grp-->charstring
%_anteil_label:anw_grp sb-->nat
%_anteil_1:anw_grp l(stammsatz)-->l(nat)
%_anteil_2:l(stammsat) l(nat) nat-->nat
ort_d_auftr_1:anw_grp l(stammsatz)-->l(vier_zahlen)

DEFINE OPS :
op_anweisung (n,ld,ag):=
  let x=lies_sb (n,liefere_eind_modul (n,ld))) in
  erz_anws (n,zuordnung (ag),gen_l_anws_inf (ag,x));

zuordnung (ag):=
   case ag is
   *steuerung-->steuerung
   *blocksteuerung-->blocksteuerung
   *zuweisung-->zuweisung
   *do-->steuerung
   ..............
   *assignment-->zuweisung
   esac ;

gen_l_anws_inf (ag,sb):=
   case ag is
   *steuerung-->
     cons_lai (op_anw_inf (do,sb),
      cons_lai (op_anw_inf (end,sb),
       cons_lai (op_anw_inf (goto,sb),
        cons_lai (op_anw_inf (if,sb),
         cons_lai (op_anw_inf (while,sb),nil_lai)))))
   *blocksteuerung-->
     cons_lai (op_anw_inf (begin,sb),
      cons_lai (op_anw_inf (call,sb),
       cons_lai (op_anw_inf (entry,sb),
        cons_lai (op_anw_inf (code,sb),
         cons_lai (op_anw_inf (procedure,sb),
          cons_lai (op_anw_inf (return,sb),
           cons_lai (op_anw_inf (null,sb),nil_lai)))))))
   *zuweisung-->cons_lai (op_anw_inf (assignment,sb),nil_lai)
```



```
   otherwise -->cons_lai (op_anw_inf (ag,sb),nil_lai)
   esac ;

op_anw_inf (ag,sb):=
   erz_ai (ag,ort_des_auftretens (ag,sb),
           %_angabe (ag,sb),%_anteil_label (ag,sb));

ort_d_auftretens (ag,sb):=
   ort_d_auftr_1 (ag,preorder_sb (sb));

ort_d_auftr_1 (ag,lsb):=
   case lsb is
   *nil_lsb-->nil_lvz
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),
                       change_to_charstring (ag))
       then cons_lvz (select_ort (s),ort_d_auftr_1 (ag,rest))
       else ort_d_auftr_1 (ag,rest)
     fi
   esac ;

select_ort (s):=
   erz_vz (ssl_7 (ssts_1 (s)),
           szp_1 (ssl_8 (ssts_1 (s))),
           szp_2 (ssl_8 (ssts_1 (s))),
           szp_3 (ssl_8 (ssts_1 (s))));

change_to_charstring (ag):=
   case ag is
   *steuerung-->create (s,create (t,create (e,create (
                u,create (e,create (r,create (u,create (
                n,create (g,blank)))))))))
   ...........
   esac ;

%_angabe (ag,sb):=
   div (mult (anzahl_lvz (ort_d_auftretens (ag,sb)),100),
             diff_zp (sdt_5 (ssts_2 (wurzel_sb (sb)))));

%_anteil_label (ag,sb):=
   let x=%_anteil_2 (preorder_sb (%_anteil_1 (ag,
                                  preorder_sb (sb)),0) in
   let y=anzahl_lvz (ort_d_auftretens (ag,sb)) in
   div (mult (x,100),y);

%_anteil_1 (ag,lsb):=
   case lsb is
   *nil_lsb-->nil_lnat
```

```
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),
                       change_to_charstring (ag))
       then cons_lnat (ssl_7 (ssts_1 (s)),
                      %_anteil_1 (ag,rest))
       else %_anteil_1 (ag,rest)
     fi
   esac ;

%_anteil_2 (lsb,lnat,n):=
   case lsb is
   *nil_lsb-->n
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
        and ist_enth_lnat (ssl_7 (ssts_1 (s)),lnat)
       then %_anteil_2 (rest,entferne_lnat (
                              ssl_7 (ssts_1 (s)),lnat),incr (n))
       else %_anteil_2 (rest,lnat,n)
     fi
   esac ;

ENDSPEC
```


3.2.4.3. Datentyp

3.2.4.3.1. Beschreibung

Es ist ein Quellmodul (UO) bzgl. der vorkommenden Datentypen auszuwerten. Es werden folgende Datentypen berücksichtigt:
Area ,Bit, Character, Class, Entry, Fix-Binary, Integer, Label, Mode, Pointer, Structure, Set, Array.

Der Benutzer hat die Möglichkeit, sich einen Datentyp analysieren zu lassen, oder durch die Angabe "alle datentypen", sich sämtliche Datentypen analysieren zu lassen.

Die Ausgabe hat folgendes Bild:

1. Der Kopf der Ausgabe ist der Name des auszuwertenden Quellmoduls.
2. Der Rumpf der Ausgabe besteht aus einer Liste, deren Elemente folgende Information enthalten:
   1. Name des Datentyps
   2. Eine Liste der verschiedenen Variablen des Datentyps mit den Informationen:
      1. Name der Variable
      2. Attribute der Variable
      3. Ort des Auftretens (Stmtnr, Line_nr, Inc_nr, Inc_line_nr)
      4. Ort der Definition (Stmtnr, Line_nr, Inc_nr, Inc_line_nr)
   3. Anzahl der verschiedenen Variablen dieses Typs
   4. %_Anteil der Variablen des Datentyps im Verhältnis zur Gesamtzahl der Quellstmts des UO

<table>
<tr><td colspan="4">Name UO</td></tr>
<tr><td colspan="3">Area</td><td>Bit</td><td>.....</td></tr>
<tr><td>Name 1

Attribute

Ort des Auftretens

Ort der Definition</td><td>Name 2</td><td>...</td><td></td><td></td></tr>
<tr><td colspan="3">Anzahl</td><td></td></tr>
<tr><td colspan="3">%_Anteil</td><td></td></tr>
</table>

Jede Deklaration eines Datentyps ist in der Symboltabelle enthalten. Der Datentyp der Deklaration ist in der Attributliste des deklarierten Namens enthalten.

Für den Datentyp Entry gilt, daß zwischen drei verschiedenen Attributen unterschieden wird. Enthält die Attributliste das Attribut
- Main Entry, so handelt es sich um die Deklaration eines Entry durch ein Procedure Statement,
- Sec. Entry, so handelt es sich um die Deklaration eines Entry durch ein Entry Statement,
- Entry, so handelt es sich um die Deklaration einer Entry Variablen oder eines externen Entries.

3.2.4.3.2. Spezifikation Datentyp

```
SSPEC DATTYP

USE SSPECS : Bool

PUBLIC SORTS : dattyp

PUBLIC OPS :
alle_datentypen:-->dattyp
area, bit, character, class, entry, fix-binary:-->dattyp
integer, label, mode, pointer, structure, set:-->dattyp

CONSTRUCTORS :
*alle_datentypen, *area, *bit, *character, *class,
*entry, *fix-binary, *integer, *label, *mode,
*pointer, *structure, *set

ENDSPEC


INSTANTIATE 4_Tupel to Var_St
ACTUALIZE : #x1 by Dattyp, #x2 by Name
            #x3 by Attributliste, #x4 by Nat
            SORTS : #x1.elem by dattyp
                    #x2 by name
                    #x3 by attributliste
                    #x4 by nat
RENAME : SORTS : 4_tupel by var_st
         OPS : erz_vier by erz_vst
END


INSTANTIATE Liste to L(Var_St)
ACTUALIZE : #x by Var_St
            SORTS : #x.elem by var_st
RENAME : SORTS : liste by l(var_st)
         OPS : nil by nil_vst
END


INSTANTIATE 4_Tupel to Var_Inf
ACTUALIZE : #x1 by Name
            #x2 by Attributliste
```

```
            #x3,#x4 by L(Vier_Zahlen)
            SORTS : #x1.elem by name
                    #x2 by attributliste
                    #x3.elem, #x4.elem by l(vier_zahlen)
RENAME : SORTS : 4_tupel by var_inf
         OPS : erz_vier by erz_vi
END


INSTANTIATE Liste to L(Var_Inf)
ACTUALIZE : #x by Var_Inf
            SORTS : #x.elem by var_inf
RENAME : SORTS : liste by l(var_inf)
         OPS : nil by nil_lvi
END


INSTANTIATE 4_Tupel to Dat_Inf
ACTUALIZE : #x1 by Dattyp
            #x2 by L(Var_Inf)
            #x3,#x4 by Nat
            SORTS : #x1.elem by dattyp
#x2.elem by l(var_inf)
#x3.elem, #x4.elem by
RENAME : SORTS : 4_tupel by dat_inf
         OPS : erz_vier by erz_di
END


INSTANTIATE Liste to L(Dat_Inf)
ACTUALIZE : #x by Dat_Inf
            SORTS : #x.elem by dat_inf
RENAME : SORTS : liste by l(dat_inf)
         OPS : nil by nil_ldi)
END


INSTANTIATE Tupel to Datentyp_Stat
ACTUALIZE : #x1 by Name
            #x2 by L(Dat_Inf)
            SORTS : #x1.elem by name
                    #x2.elem by l(dat_inf)
RENAME : SORTS : tupel by datentyp_stat
         OPS : erz_tup by erz_dts
```



```
END


SSPEC DATENTYP

USE SSPECS : Schnittstelle, Dattyp, L(Var_St), L(Dat_Inf),
             Datentyp_Stat, Allgemein, Anweisung

PUBLIC OPS :
op_datentyp:name dattyp l(dat)-->datentyp_stat

PROPERTIES
/*
   Die Operationsaufruffolge des Datentyps orientiert sich an der zu
erstellenden Ausgabe. Die verschiedenen Kästchen der Ausgabe werden
durch die Ergebnisse der entsprechenden Operationen beschrieben.
   Die nach außen zur Verfügung gestellte Operation op_datentyp, die
die verlangte Ausgabe liefert, ruft die private Operation datentyp_1
mit den für die weitere Auswertung benötigten Parametern SB und ST
auf.
In der Operation datentyp_1 werden nun zwei Fälle unterschieden:
1. Der Benutzer möchte Informationen über alle im Quellprogramm vor-
   kommenden Datentypen. Dieser Fall wird durch die Angabe
   "alle_datentypen" charakterisiert. Es wird dann eine Liste er-
   stellt, deren Elemente jeweils Informationen über einen Datentyp
   enthalten.
2. Der Benutzer möchte Informationen über einen Dastentyp. Der Benutz-
   ter gibt dann diesen Datentyp direkt an. Es wird dann eine Liste
   erstellt, die nur aus einem Element besteht. Dieses Element enthält
   die Informationen über Variable des Datentyps.

   Informationen über einen Datentyp liefert die Operation op_dat_inf.
Diese Operation trägt in den Teil der Ausgabe, der die Information
über einen Datentyp enthält, den Datentyp, die Liste der Variablen
dieses Typs durch die Operation gen_l_var_inf, die Anzahl der Variab-
len und die %-Angabe ein.
   Die Anzahl der Variablen ergibt sich durch die Anzahl der Elemente
der Liste der Variablen.
   Mittels dieser Anzahl und der Stmtlänge des UO wird der %-Anteil
ausgerechnet.

Die Operation gen_l_var_inf benötigt folgende Parameter:
1. Die Liste aller Variablen eines Datentyps.
      Diese Liste wird durch die Operation gen_l_var_st erstellt.
   gen_l_var_st durchsucht die ST systematisch. In jedem Gebiet der ST
   werden die Einträge untersucht. Enthält die Attributliste eines
   Eintrags den gesuchten Datentyp, so wird ein 4-Tupel (Var_St) er-
```

zeugt, das aus dem Datentyp, dem Namen des Eintrags, der Attributliste des Eintrags und der Blocknummer des Blocks, der diese Deklaration enthält, besteht. Aus all diesen 4-Tupeln wird eine Liste erzeugt.

Die Blocknummer ist notwendig, um im SB die entsprechende Deklaration zu finden.

Für jedes Element der Liste von 4-Tupeln wird dann aus dem SB weitere Information entnommen.

2. Die Liste aller Variablen aller Datentypen.
Diese Liste wird durch die Operation gen_l_all_var_st erstellt, indem für jeden möglichen Datentyp, die obige Liste durch die Operation gen_l_var_st erstellt wird und alle Listen anschließend zusammengefügt werden.

Diese Liste wird benötigt, um sicherzustellen, daß im SB das Auftreten einer Variablen in den richtigen Blöcken gesucht wird.

Beispiel:
```
A : PROCEDURE;
      DECLARE X INTEGER;
      ...
      BEGIN
         DECLARE X CHARACTER;
         ...
      END;
    END A;
```

Ein Auftreten von X vom Typ integer darf nicht in dem Begin Block gesucht werden, da X in diesem Block als Charaktervariable neu definiert wurde.

3. Den SB.

gen_l_var_inf arbeitet die Liste der Variablen eines Datentyps sequentiell ab. In die Ausgabe kann direkt der Name der Variablen und die Attributliste eingetragen werden. Die Orte des Auftretens werden durch die Operation best_ort_auftr und die Orte der Definition durch die Operation bestimme_ort_def bestimmt.

Der Ort der Definition wird so bestimmt:
Es sind zwei Fälle zu unterscheiden.
1. Die Variable ist mit dem Scope Attribut "external" verbunden.
2. Die Variable ist lokal.

Im ersten Fall kann es mehrere Blöcke im Quellmodul geben, in denen die Variable ebenfalls als external deklariert ist.

Eine Deklaration eines Identifiers mit external Attribut bezeichnet man als externe Deklaration dieses Identifiers. Alle externen Deklarationen eines Identifiers werden verbunden. Die Umgebung einer solchen Variablen ist die Vereinigung der Umgebungen aller externen Deklara-

tionen. Jede externe Deklaration ist somit auch Ort der Definition der Deklaration der Variablen.

Die Operation gen_l_ext_attr erzeugt eine Liste, die alle externen Deklarationen einer Variablen enthält. Für jedes einzelne Element wird dann der Ort der Definition bestimmt und sämtliche Ortsdefinitionen werden dann zu einer Liste zusammengefügt. Die Suche nach dem Ort der Definition im Fall 1 wird also auf den Fall 2 zurückgeführt.

Den Ort der Definition einer Variablen bestimmt man so:
Aus der Blocknummer des 4-Tupels geht eindeutig hervor, in welchem Block des SB die Deklaration der Variablen zu suchen ist. Der SB, der dem Deklarationsteil dieses Blocks entspricht, wird erzeugt und durchsucht. Hat ein Stammsatz aus diesem Teil des SB die Satzart "declare identifier" und entsprechen sich die Namen aus dem Inhalt des Stammsatzes und dem Namensfeld des 4-Tupels, so ist der gesuchte Stammsatz gefunden und die Ortsangaben können aus dem Stammsatz entnommen werden.

Sind sämtliche Stammsätze durchsucht, ohne daß eine passende Deklaration gefunden wurde, liegt einer der folgenden Fälle vor:
1. Es handelt sich um ein Label, das nicht explizit deklariert wurde, sondern dadurch, daß es ein Prefix eines Stmts ist, das kein Procedure- oder Entry-Stmt ist;
2. Es handelt sich um den Datentyp Entry, wobei der Entryname durch ein Entry-Stmt oder ein Procedure-Stmt als Entry deklariert ist.

In beiden Fällen wird der Ort der Definition durch die Operation best_ort_def_label_entry bestimmt.

In SPL können folgende Teile eines Programms mit einem Label versehen sein:
ending, if-clause, beg-stmt, non-group-stmt und do-stmt.

Für Entrynamen gilt, daß es im SB ein Procedure-Stmt oder ein Entry-Stmt gibt, das mit einem Label beginnt.

Im Block des SB, in dem der Label oder der Entry deklariert ist, werden alle Söhne von Stammsätzen mit Satzart "ending" (if clause, beg stmt, non group stmt, do stmt, entry stmt, proc stmt) betrachtet. Diese Stammsätze werden auf Satzart "label identifier" überprüft. Stimmt der Name des Label Identifier mit dem Namen des gesuchten Labels oder Entries überein, so ist der Ort der Definition gefunden, und es werden die entsprechenden Informationen aus dem Stammsatz entnommen.

Die Orte des Auftretens einer Variablen bestimmt man folgendermaßen:
Handelt es sich um eine Deklaration mit external Attribut, so ist das angewandte Auftreten in allen Blöcken zu suchen, in denen die Variable deklariert ist. Die Orte des Auftretens in einem Block werden durch die Operation best_ort_auftr_1 gesucht. Zu berücksichtigen ist, daß der Datentyp Entry, sofern es keine Entry Variable ist, in dem die Deklaration umgebenden Block bekannt ist. Die Operation

best_ort_auftr_2 wird mit den Söhnen der Prozedur oder des Blocks aufgerufen, in dem die Variable deklariert ist. Diese Stammsätze werden der Reihe nach durchsucht.
- Handelt es sich um einen Prozedur- oder Begin-Block, dann wird entschieden, ob dies ein Block ist, in dem die Variable neu bzw. umdefiniert wurde. Falle ja, darf in diesem Block nicht nach einem Auftreten gesucht werden. Falls nein, sind die Söhne des Blocks an die Liste der noch zu untersuchenden Stammsätze anzuhängen.
- Handelt es sich um einen Stammsatz mit Satzart "identifier", dann ist zu prüfen, ob der Name des Identifiers und der Name der Variablen übereinstimmen. Falls ja, wird der Ort des Auftretens entnommen. Falls nein, wird der nächste Stammsatz untersucht.
- Handelt es sich um irgendeinen anderen Stammsatz, dann werden die Söhne dieses Stammsatzes an die Liste der zu untersuchenden Stammsätze angehängt.

Zu beachten ist, daß der Ort der Definition nicht auch Ort des Auftretens ist.
*/

```
/*
enth_vater_proc_beg_st (k,st) liefert zum Knoten k aus ST den Prozedurblock bzw. dem Begin Block der Knoten k als Sohn hat.
*/
∀ knoten_st k: ∀ st st: ∃ knoten_st k':
  k'=enth_vater_proc_beg_st (k,st)
  ∧ (k'≠error
   ⇒ ((ist_proz_block (k')=true
        ∨ ist_begin_block (k')=true)
       ∧ k'=vater_st (k)))

/*
v_qual_sa_bnr (sa,n,lsb) liefert den ersten Stammsatz aus lsb, der in Satzart und Blocknummer mit sa und n übereinstimmt.
*/
∀ charstring sa: ∀ nat n: ∀ l(stammsatz) lsb:
  ∃ stammsatz s: s=v_qual_sa_bnr (sa,n,lsb)
  ∧ (s≠error
    ⇒ (ist_enth_lsb (s,lsb)=true
       ∧ ssl_6 (ssts_1 (s))=sa
       ∧ sdt_10 (ssts_2 (s))=n))

/*
v_qual_sa_block_bnr (n,lsb) liefert den ersten Stammsatz aus lsb mit Satzart "procedure" oder "begin block" und Blocknummer "n".
*/
∀ l(stammsatz) lsb: ∀ nat n:
```



```
  ∃ stammsatz s: s=v_qual_sa_block_bnr (n,lsb)
   ∧ (s≠error
     ⇒ ( ist_enth_lsb (s,lsb)=true
         ∧ (ssl_6 (ssts_1 (s))=procedure
            ∨ ssl_6 (ssts_1 (s))=begin block)
         ∧ sdt_10 (ssts_2 (s))=n))

/*
lösche_ext (v,lvst) löscht in lvst alle Elemente, die im Namen mit dem Namen aus v übereinstimmen und die in ihrer Attributliste das External-Attribut haben.
*/
∀ var_st v: ∀ l(var_st) lvst: ∃ l(var_st) l:
  l=lösche_ext (v,lvst)
  ∧ ∀ varst v': ist_enth_lvst (v',l)=true
      ⇒ (svst_2 (v')≠svst_2 (v)
          ∧ ist_enth_attr (external,svst_3 (v'))=false)

/*
bestimme_ort_auftr (v,lvst,sb,lvstall) liefert alle Auftreten des Variablennamens aus v im SB sb. Dabei wird der Gültigkeitsbereich des Namens berücksichtigt.
*/
∀ var_st v: ∀ l(var_st) lvst:
  ∀ sb sb: ∀ l(var_st) lvstall: ∃ l(vier_zahlen) l:
   l=bestimme_ort_auftr (v,lvst,sb,lvstall)
   ∧ ∀ vier_zahlen vz: ist_enth_lvz (vz,l)=true
        ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
             ∧ ssl_6 (ssts_1 (s))=identifier
             ∧ sdt_3 (ssts_2 (s))=svst_2 (a)
             ∧ vz=select_ort (s)
             ∧ ist_enth_lnat (sdt_10 (ssts_2 (s)),
                              gen_l_neu_def (v,lvstall))=false
             ∧ ist_enth_lnat (vz, bestimme_ort_def (v,lvst,sb))
                                                    =false
/*
gen_l_neu_def (v,lvstall) liefert die Blocknummern aller Blöcke, in denen die Variable mit dem Namen aus v neu definiert ist, indem die Liste der Deklarationen sämtlicher Variablen durchsucht wird und die Blocknummern der Variablen notiert werden, die im Namen mit dem Namen aus v übereinstimmen.
*/
∀ l(var_st) lvstall: ∀ var_st v: ∃ l(nat) l:
  l=gen_l_neu_def (v,lvstall)
  ∀ nat n: ist_enth_lnat (n,l)=true
   ⇒ ∃ var_st v': ist_enth_lvst (v',lvstall)=true
       ∧ n=svst_4 (v')
       ∧ n≠svst_4 (v)
```

```
      ∧ svst_2(v)=svst_2 (v')

/*
gen_l_var_st  (dtyp,st)  liefert für alle deklarierten Datentypen  ein
4-Tupel,  bestehend aus dem Datentyp, dem Namen, der Attributliste und
der Blocknummer des Blocks der Deklaration, in der die Deklaration be-
kannt ist.
*/
∀ st st: ∀ dattyp dtyp:
  ∃ l(var_st) l: l=gen_l_var_st (dtyp,st)
   ∧ ∀ var_st v: ist_enth_lvst (v,l)=true
      ⇒ ∃ knoten_st k: ist_in_baum_st (k,st)=true
           ∧ ∃ gegiet gb: gb=skst_1 (k)
              ∧ ∃ eintrag e: ist_enth_gb (e,gb)=true
                  ∧ svst_2 (v)=setr_1 (e)
                  ∧ svst_4 (v)=give_blocknr (
                                enth_vater_proc_beg_st (k,st))
                  ∧ svst_3 (v)=setr_2 (e)
                  ∧ svst_1 (v)=dtyp
                  ∧ (ist_enth_attr (
                      change_dt_to_charstring (dtyp),setr_2 (e))=true
                      ∨ ist_enth_attr (main entry,setr_2 (e))=true
                      ∨ ist_enth_attr (
                                secondary entry,setr_2 (e))=true)

/*
bestimme_ort_def  (v,lvst,sb)  liefert für das Element v den  Ort  der
Definition im Quellmodul.
*/
∀ sb sb: ∀ l(var_st) l:
  ∃ l(vier_zahlen) l: l=bestimme_ort_def (v,lvst,sb)
   ∧ ∀ vier_zahlen vz: ist_enth_lvz (vz,l)=true
       ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
           ∧ sdt_3 (ssts_2 (s))=svst_2 (v)
           ∧ sdt_10 (ssts_2 (s))=svst_4 (v)
           ∧ (ssl_6 (ssts_1 (s))=declare identifier
              ∨ ssl_6 (ssts_1 (s))=proc stmt
              ∨ ssl_6 (ssts_1 (s))=secondary entry
              ∨ ssl_6 (ssts_1 (s))=label identifier)

PRIVATE OPS :
op_datentyp_1:name datentyp sb st-->datentyp_stat
gen_dat_inf:dattyp sb st-->dat_inf
gen_l_var_inf:l(var_st) l(var_st) sb-->l(var_inf)
gen_l_var_st:dattyp st-->l(var_st)
gen_l_var_st_1:dattyp l(knoten_st) st-->l(var_st)
gen_l_var_st_2:dattyp symtab knoten_st-->lvar_st)
gen_l_var_st_3:dattyp gebiet knoten_st st-->l(var_st)
```



```
bestimme_ort_def:var_st l(var_st) sb-->l(vier_zahlen)
best_ort_def_1:var_st sb-->l(vier_zahlen)
best_ort_def_2:var_st l(stammsatz)-->l(vier_zahlen)
best_ort_def_ext:l(var_st) sb-->l(vier_zahlen)
gen_l_ext_attr:var_st l(var_st) sb-->l(var_st)
best_ort_def_label_entry_:var_st sb-->l(vier_zahlen)
best_ort_def_label_entry_1:var_st l(stammmsatz)-->l(vier_zahlen)
best_ort_def_label_entry_2:var_st l(stammsatz)-->l(vier_zahlen)
ist_ort_def_label_entry:var_st l(stammsatz)-->bool
gen_l_all_var_st:st-->l(var_st)
change_dt_to_charstring:dattyp-->charstring
gen_l_neu_def:var_st l(var_st)-->l(nat)
best_ort_auftreten:var_st l(var_st) sb l(var_st)-->l(vier_zahlen)
best_ort_auftr_1:var_st l(nat) sb l(var_st)-->l(vier_zahlen)
best_ort_auftr_2:var_st l(stammmsatz) l(nat) l(var_st)
                                                  -->l(vier_zahlen)
best_ort_auftr_ext:l(var_st) sb l(var_st) l(var_st)
                                                  -->l(vier_zahlen)
enth_vater_proz_beg_st:knoten_st st -->knoten_st
enth_vater_proz_st:knoten_st st-->knoten_st
lösche_ext:var_st l(var_st)-->l(var_st)
v_qual_sa_bnr:charstring nat l(stammsatz)-->stammsatz
v_qual_sa_block_bnr:nat l(stammsatz)-->stammsatz

DEFINE OPS :
op_datentyp (n,dt,ld):=
  let x=liefere_eind_modul (n,ld) in
   op_datentyp_1 (n,dt,lies_sb (n,x),lies_st (n,x));

op_datentyp_1 (n,dt,sb,st):=
  case dt is
  *alle_datentypen-->
    erz_dts (n,
      cons_ldi (gen_dat_inf (area,sb,st),
       cons_ldi (gen_dat_inf (bit,sb,st),
        cons_ldi (gen_dat_inf (character,sb,st),
         cons_ldi (gen_dat_inf (class,sb,st),
          cons_ldi (gen_dat_inf (entry,sb,st),
           cons_ldi (gen_dat_inf (fix-binary,sb,st),
            cons_ldi (gen_dat_inf (integer,sb,st),
             cons_ldi (gen_dat_inf (label,sb,st),
              cons_ldi (gen_dat_inf (mode,sb,st),
               cons_ldi (gen_dat_inf (pointer,sb,st),
                cons_ldi (gen_dat_inf (structure,sb,st),
                 cons_ldi (gen_dat_inf (set,sb,st))))))))))))))
   otherwise -->
    erz_dts (n,cons_ldi (gen_dat_inf (dt,sb,st), nil_ldi))
  esac ;
```

```
gen_dat_inf (dt,sb,st):=
  let x=gen_l_var_inf (gen_l_var_st (dt,st),
                       gen_l_all_var_st (st) ,sb) in
   erz_di (dt,gen_l_var_inf (x),anzahl_lvi (x),
          div (mult (anzahl_lvi (x),100),
               diff_zp (sdt_5 (ssts_2 (wurzel_sb (sb))));

gen_l_var_inf (l1,l2,sb):=
   case l1 is
   *nil_lvst-->nil_lvi
   *cons_lvst (v,rest)-->
     cons_lvi (erz_vi (svst_2 (v),
                       svst_3 (v),
                       best_ort_auftr (v,l1,sb,l2),
                       best_ort_def (v,l1,sb)),
               gen_l_var_inf (lösche_ext (v,rest),l2,sb))
   esac ;

gen_l_var_st (dt,st):=
   gen_l_var_st_1 (dt,preorder_st (st),st);

gen_l_var_st_1 (dt,lst,st):=
   case lst is
   nil_lst-->nil_lvst
   *cons_lst (ket,rest)-->
     if ist_proz_block (kst)
        or ist_begin_block (kst)
       then gen_l_var_st_1 (dt,rest)
       else append_lvst (gen_l_var_st_1 (dt,skst_1 (kst),kst,st),
                         gen_l_var_st_1 (dt,rest,st)
    fi
   esac ;

gen_l_var_st_2 (dt,stab,kst,st):=
   case stab is
   *blck_b (b)-->error.l(var_st)
   *blck_g (gb)-->gen_l_var_st_3 (dt,gb,kst,st)
   esac ;

gen_l_var_st_3 (dt,gb,kst,st):=
   case gb is
   *nil_gb-->nil_lvst
   *cons_gb (e,rest)-->
     if ist_enth_attr (entry,setr_2 (e))
        or ist_enth_attr (main entry,setr_2 (e))
        or ist_enth_attr (sec. entry,setr_2 (e))
        or ist_enth_attr (change_dt_to_charstring (dt),setr_2 (e))
       then cons_lvst (
```



```
            erz_vst (dt,setr_1 (e),
              setr_2 (e),
              give_blocknr (enth_vater_proz_beg_st (kst,st)),
              gen_l_var_st_3 (dt,rest,kst,st)),
            gen_l_var_st_3 (dt,rest,kst,st)
       else gen_l_var_st_2 (dt,rest,kst,st)
     fi
   esac ;

bestimme_ort_def (v,l,sb):=
   if ist_enth_attr (external,svst_3 (v))
     then best_ort_def_ext (gen_l_ext_attr (v,l),sb)
     else cons_lvz (best_ort_def_1 (v,sb),nil_lvz)
   fi ;

best_ort_def_1 (v,sb):=
   best_ort_def_2 (v,preorder_sb( t_baum_sb (
     v_qual_sa_bnr (declaration part, svst_4 (v),
       preorder_sb (sb)),sb)),sb);

best_ort_def_ext (lv,sb):=
   case lv is
   *nil_lvst-->nil_lvz
   *cons_lvst (v,rest)-->
     cons_lvz (best_ort_def_1 (v,sb),
               best_ort_def_ext (rest,sb))
   esac ;

gen_l_ext_attr (v,lv):=
   case lv is
   *nil_lvst-->nil_lvst
   *cons_lvst (v1,rest)-->
     if eq.name (svst_2 (v),svst_2 (v1)
        and ist_enth_attr (external, svst_3 (v1))
       then cons_lvst (v1,gen_l_ext_attr (v,rest))
       else gen_l_ext_attr (v,rest)
     fi
   esac ;

best_ort_def_2 (v,lsb,sb):=
   case lsb is
   *nil_lsb-->
     if eq.dattyp (svst_1 (v),label)
        or eq.dattyp (svst_1 (v),entry)
       then best_ort_def_label_entry (v,sb)
       else error.l(vier_zahlen)
     fi
   *cons_lsb (s,rest)-->
```

```
     if eq.charstring (ssl_6 (ssts_1 (s)),declare identifier)
        and eq.name (sdt_3 (ssts_2 (s)),svst_2 (v))
       then cons_lvz (select_ort (s),nil_lvz)
       else best_ort_def_2 (v,rest,sb)
      fi
   esac ;

best_ort_def_label_entry (v,sb):=
   best_ort_def_label_entry_1 (v,
     preorder_sb (t_baum_sb (v_qual_sa_blocknr (
                 svst_4 (v),preorder_sb (sb), sb);

best_ort_def_label_entry_1 (v,lsb,sb):=
   case lsb is
   *nil_lsb-->error.l(vierzahlen)
   *cons_lsb (s,rest)-->
       let x=ssl_6 (ssts_1 (s)) in
         if (eq.charstring (x,proc stmt)
             or eq.charstring (x,secondary entry)
             or eq.charstring (x,ending)
             or eq.charstring (x,if clause)
             or eq.charstring (x,beg stmt)
             or eq.charstring (x,non group stmt)
             or eq.charstring (x,do stmt))
            and ist_ort_def_label_entry (v,lds_sb (s,sb))
           then best_ort_def_label_entry_2 (v,lds_sb (s,sb))
           else best_ort_def_label_entry_1 (v,rest,sb)
         fi
   esac ;

best_ort_def_label_entry_2 (v,lsb):=
   case lsb is
   *nil_lsb-->error.l(vier_zahlen)
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
        and eq.name (sdt_3 (ssts_2 (s)),svst_2 (v))
       then cons_lvz (select_ort (s),nil_lvz)
       else best_ort_def_label_entry_2 (v,rest)
     fi
   esac ;

ist_ort_def_label_entry (v,lsb):=
   case lsb is
   *nil_lsb-->false
   *cons_lsb (s,rest)-->
     if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
        and eq.charstring (sdt_3 (ssts_2 (s)),svst_2 (v))
       then true
```



```
       else ist_ort_def_label_entry (v,rest)
     fi
   esac ;

gen_l_all_var_st (st):=
  append_lvst (gen_l_var_st (area,st),
   append_lvst (gen_l_var_st (bit,st),
    append_lvst (gen_l_var_st (character,st),
     append_lvst (gen_l_var_st (class,st),
      append_lvst (gen_l_var_st (entry,st),
       append_lvst (gen_l_var_st (fix-binary,st),
        append_lvst (gen_l_var_st (integer,st),
         append_lvst (gen_l_var_st (label,st),
          append_lvst (gen_l_var_st (mode,st),
           append_lvst (gen_l_var_st (pointer,st),
            append_lvst (gen_l_var_st (structure,st),
              append_lvst (gen_l_var_st (set,st),
               append_lvst (gen_l_var_st (array,st))))))))))));

change_dt_to_charstring (dt):=
   case dt is
   *area-->create (a, create (r, create (e, create (a,blank))))
   .....
   esac ;

gen_l_neu_def (v,lv):=
   case lv is
   *nil_lvst-->nil_lnat
   *cons_lvst (v1,rest)-->
     if eq.name (svst_2 (v),svst_2 (v1))
        and not eq.nat (svst_4 (v),svst_4 (v1))
       then cons_lnat (svst_4 (v1),gen_l_neu_def (v,rest))
       else gen_l_neu_def (v,rest)
     fi
   esac ;

best_ort_auftr (v,lv,sb,lvall):=
   if ist_enth_attr (external,svst_3 (v))
     then best_ort_auftr_ext (
           gen_l_ext_attr (v,lv),sb,lvall)
     else best_ort_auftr_1 (v,
           gen_l_neu_def (v,lvall),sb,lv)
   fi ;

best_ort_auftr_1 (v,lnat,sb,lv):=
   if eq.dattyp (svst_1 (v),entry)
      and not ist_enth_attr (external,svst_3 (v))
      and not ist_enth_attr (variable,svst_3 (v))
```

```
      then best_ort_auftr_2 (v,lds_sb (
            enth_vater_proz_beg (
               v_qual_sa_block_bnr (svst_4 (v),
                  preorder_sb (sb)),sb),sb),lnat,lv)
      else best_ort_auftr_2 (v,lds_sb (
            v_qual_sa_block_bnr (svst_4 (v),
               preorder_sb (sb)),sb),lnat,lv)
   fi ;

best_ort_auftr_2 (v,lsb,lnat,lv):=
   case lsb is
   *nil_lsb-->nil_lvz
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),procedure)
          or eq.charstring (ssl_6 (ssts_1 (s)),begin block))
         and ist_enth_lnat (sdt_10 (ssts_2 (s)),lnat)
        then best_ort_auftr_2 (v,rest,lnat,lvst)
        else
         if eq.charstring (ssl_6 (ssts_1 (s)),identifier)
            and eq.name (sdt_3 (ssts_2 (s)),svst_2 (v))
            and not ist_enth_lvz (select_ort (s),
                                  best_ort_def_2 (v,lv,sb))
           then cons_lvz (select_ort (s)),
                         best_ort_auftr_2 (v,rest,lnat,lv))
           else best_ort_auftr_2 (v,
                append_lsb (lds_sb (s,sb),rest),lnat,sb,lv)
         fi
      fi
   esac ;

best_ort_auftr_ext (lv1,sb,lvall,lv2):=
   case lv1 is
   *nil_lvst-->nil_lvz
   *cons_lvst (v,rest)-->
      append_lvz (best_ort_auftr_1 (v,
                     gen_l_neu_def (v,lvall),sb,lv2),
                     best_ort_auftr_ext (rest,sb,lvall,lv2))
   esac ;

enth_vater_proz_beg_st (kst,st):=
   if eq.knoten_st (wurzel_st (st),kst)
     then error.knoten_st
     else if ist_proz_block (vater_st (kst,st))
             or ist_begin_block (vater_st (kst),st))
            then vater_st (kst,st)
            else error.knoten_st
          fi
   fi ;
```



```
enth_vater_proz_st (kst,st):=
   if eq.knoten_st (wurzel_st (st),kst)
     then error.knoten_st
     else if ist_proz_block (vater_st (kst,st))
             then vater_st (kst,st)
             else enth_vater_proz_st (vater_st (kst,st))
          fi
   fi ;

lösche_ext (v,lv):=
   case lv is
   *nil_lvst-->nil_lvst
   *cons_lvst (v,rest)-->
      if ist_enth_attr (external,svst_3 (v))
         and eq.name (svst_2 (v),svst_2 (v1))
        then lösche (v,rest)
        else cons_lvst (v1,lösche_ext (v,rest))
      fi
   esac ;

v_qual_sa_bnr (sa,n,lsb):=
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),sa)
         and eq.nat (sdt_10 (ssts_2 (s)),n)
        then s
        else v_qual_sa_bnr (sa,n,rest)
      fi
   esac ;

v_qual_sa_block_bnr (n,lsb):=
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),procedure)
          or eq.charstring (ssl_6 (ssts_1 (s)),begin block))
         and eq.nat (sdt_10 (ssts_2 (s)),n)
        then s
        else v_qual_sa_block_bnr (n,rest)
      fi
   esac ;

ENDSPEC
```

3.2.4.4. Konvertierung

3.2.4.4.1. Beschreibung

Unter Datenkonvertierung versteht man die Umwandlung eines gegebenen Datentyps in einen verlangten Datentyp.

Diese Auswertefunktion liefert dem Benutzer Informationen, wo in seinem Quellmodul Konvertierungen und wieviele Konvertierungen an dieser Stelle auftreten. Überprüft werden Konvertierungen in Ausdrücken, die selbst nicht Teil eines anderen Ausdrucks sind, und Konvertierungen in Zuweisungsstatements.

Die Ausgabe der Funktion sieht wie folgt aus:

Name UO	
Stmtnr	Anzahl der Konvertierungen
.....	.....
Stmtnr	Anzahl der Konvertierungen
Gesamtzahl aller Konvertierungen	

Außer in Zuweisungen können in SPL in den Statements IF-Statements, RETURN-Statements und WHILE-Statements, und in LENGTH-Functions und MODULO-Functions, und als Argumente eines CALL-Statements Ausdrücke auftreten.
Folgende zwei Funktionen werden als spezifiziert vorausgesetzt:
anzahl_convert_expr (t_sb,sb), die für den Teilstrukturbaum eines expression die Anzahl der Konvertierungen ermittelt und analog anzahl_convert_assignment (t_sb,sb), die die Anzahl der Konvertierungen einer Zuweisung liefert.

3.2.4.4.2. Spezifikation Konvertierung

```
INSTANTIATE Tupel to Nat_Nat
ACTUALIZE : #x1, #x2 by Nat
            SORTS : #x1.elem, #x2.elem by nat
RENAME : SORTS : tupel to nat_nat
         OPS : erz_tup by erz_jj
END


INSTANTIATE Liste to L(Nat_Nat)
ACTUALIZE :#x by Nat_Nat
            SORTS : #x.elem by nat_nat
RENAME : SORTS : liste by l(nat_nat)
         OPS : nil by nil_jj
END


INSTANTIATE 3_Tupel to Konv_Stat
ACTUALIZE : #x1 by Name
            #x2 by L(Nat_Nat)
            #x3 by Nat
            SORTS : #x1.elem by name
                    #x2 by.elem by l(nat_nat)
                    #x3.elem by nat
RENAME : SORTS : 3_tupel by konv_stat
         OPS : erz_drei by erz_cv
END


SSPEC KONVERTIERUNG

USE SSPECS : Bool, Schnittstelle, Konv_Stat

PUBLIC OPS :
op_konvertierungen:name l(dat)-->konv_stat

PROPERTIES :
/*
op_konvertierungen  (m,ld)  liefert für den Quellmodul m aus der Liste
der Quelldateien die verlangte Ausgabe.
*/
∀ name m: ∀ l(dat) ld:
  ∃ konv_stat c: c=op_konvertierungen (m,ld)
```

```
  ∧ (c≠error
     ⇒ (scv_1 (c)=m
       ∧ ∃ sb sb: sb=lies_sb (m,liefere_eind_modul (m,ld))
          ∧ (sb≠error
             ⇒ ∀ nat_nat n: (ist_enth_ljj (n,scv_2 (c))=true
                ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
                  ∧ ((ssl_6 (ssts_1 (s))=expression
                    ∧ ssl_7 (ssts_1 (s))=sjj_1 (n)
                    ∧ sjj_2 (n)=anzahl_convert_expr (
                                t_baum_sb (s,sb),sb)
                    ∧ ¬ ∃ stammsatz s':
                          ist_in_baum_sb (s',sb)=true
                          ∧ s≠s'
                          ∧ ssl_6 (ssts_1 (s'))=expression
                          ∧ ssl_7 (ssts_1 (s'))=sjj_1 (n)
                          ∧ ist_in_baum_sb (s,t_baum_sb (s,sb))
                                                  =true)
                   ∨ (ssl_6 (ssts_1 (s))=assignment stmt
                       ∧ ssl_7 (ssts_1 (s))=sjj_1 (n)
                       ∧ sjj_2 (n)=anzahl_convert_assignment (
                                   t_baum_sb (s,sb),sb))
                   ∨ (ssl_6 (ssts_1 (s))= mod argumente
                       ∧ ∃ stammsatz s1,s2:
                           ist_in_baum_sb (s1,sb)=true
                          ∧ ist_in_baum_sb (s2,sb)=true
                          ∧ s1≠s2
                          ∧ ist_enth_lsb (s1,lds_sb (s1,sb))=true
                          ∧ ist_enth_lsb (s2,lds_sb (s2,sb))=true
                          ∧ ssl_6 (ssts_1 (s1))=expression
                          ∧ ssl_6 (ssts_1 (s2))=expression
                          ∧ sjj_2 (n)=add (anzahl_convert_expr (
                             (s1,sb),anzahl_convert_expr (s2,sb)))
                     ∨ (ssl_6 (ssts_1 (s))=call stmt
                          ∧ sjj_2 (n)=op_cv_callstmt)
                   ∧ sjj_2 (n)≤0

/*
op_cv_callstmt (s,sb)  liefert die Anzahl der Konvertierungen in einem
Call-Stmt,  indem  die Operation anzahl_convert_expr auf die Ausdrücke
angewandt wird, die die Argumente bilden.
*/
∀ stammsatz s: ∀ sb sb: ∃ nat n:
  n=op_cv_callstmt
  ∧ (n≠error
     ⇒ ∃ l(stammsatz) l: ∀ stammsatz s':
          ist_enth_lsb (s',l)=true
          ⇒ (ssl_6 (ssts_1 (s'))=expression
              ∧ ssl_6 (ssts_1 (vater_sb (s',sb)))=argument
```



```
              ∧ istin_baum_sb (s',t_baum_sb (s,sb))=true
              ∧ n=op_cv_mod_expr (l,sb)

/*
gesamtzahl  (lnatnat)  liefert die Summe der  natürlichen  Zahlen  der
zweiten Komponente von lnatnat.
*/
∀ l(nat_nat) l:
  gesamtzahl (l)=add (sjj_2 (car_ljj (l)),
                      gesamtzahl (cdr_ljj (l)))
  ∧ gesamtzahl (nil_ljj)=0

/*
lösche_null_komp (l) löscht aus l alle Elemente,deren zweite Komponen-
te Null ist.
*/
∀ l(nat_nat) l: ∃ l(nat_nat) l':
  l'=lösche_null_komp (l)
  ∀ nat_nat n: ist_enth_ljj (n,l')=true
   ⇒ (sjj_2 (n)≠0
       ∧ ist_enth_ljj (n,l)=true)

PRIVATE OPS :
anzahl_convert_expr:sb sb-->nat
anzahl_convert_assignment:sb sb-->nat
op_konvertierungen_1:name l(stammsatz) sb-->konv_stat
op_convert:l(stammsatz) sb-->l(nat_nat)
op_cv_mod_expr:l(stammsatz) sb-->nat
op_cv_expr:l(stammsatz) sb-->nat
op_cv_callstmt:sb sb-->nat
op_cv_callstmt_1:l(stammsatz) sb-->nat
lösche_null_komp:l(nat_nat)-->l(nat_nat)
gesamtzahl:l(nat_nat)-->nat

DEFINE OPS :
op_konvertierungen (n,ld):=
   let x=lies-.sb (n,liefere_eind_modul (n,ld)) in
    op_konvertierungen_1 (n,preorder_sb (x),x);

op_konvertierungen_1 (n,lsb,sb):=
  erz_cv (n,
          lösche_null_komp (op_convert (lsb,sb)),
          gesamtzahl (op_convert (lsb,sb));

op_convert (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_ljj
   *cons_lsb (s,rest)-->
```

```
      if eq.charstring (ssl_6 (ssts_1 (s)),assignment statement)
        then cons_ljj (
             erz_jj (ssl_6 (ssts_1 (s)),
                     anzaahl_convert_assignment (t_baum_sb (s,sb),
                                                 op_convert (rest,sb))
        else
         if eq.charstring (ssl_6 (ssts_1 (s)),call statement)
           then cons_ljj (
                  erz_jj (ssl_7 (ssts_1 (s)),
                          op_cv_callstmt (
                          t_baum_sb (s,sb),sb),
                  op_convert (rest,sb))
           else
           if eq.charstring (ssl_6 (ssts_1 (s)),return value)
              or eq.charstring (ssl_6 (ssts_1 (s)),if clause)
              or eq.charstring (ssl_6 (ssts_1 (s)),while option)
              or eq.charstring (ssl_6 (ssts_1 (s)),length argument)
            then op_cv_expr (lds_sb (s,sb),sb)
            else if eq.charstring (ssl_6 (ssts_1 (s)),mod argumente)
                   then cons_ljj (
                          erz_jj (ssl_z (ssts_1 (s)),
                                  op_cv_mod_expr (
                                  lds_sb (s,sb),sb),
                          op_convert (rest,sb)
                   else op_convert (rest,sb)
                 fi
           fi
         fi
        fi
   esac ;

op_cv_mod_expr (lsb,sb):=
   case lsb is
   *nil_lsb-->0
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),expression)
        then add (anzahl_convert_expr (t_baum_sb (s,sb),sb),
                  op_cv_mod_expr (rest,sb))
        else op_cv_mod_expr (rest,sb)
      fi
   esac ;

gesamtzahl (l):=
   case l is
   *nil_ljj-->o
   *cons_ljj (l1,rest)-->add (sjj_2 (l1),gesamtzahl (rest))
   esac ;
```



```
op_cv_expr (lsb,sb):=
   case lsb is
   *nil_lsb-->error.nat
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),expression)
        then anzahl_convert_expr (t_baum_sb (s,sb))
        else op_cv_expr (rest,sb)
      fi
   esac ;

op_cv_callstmt (tsb,sb):=
   op_cv_callstmt_1 (preorder_sb (tsb),sb);

op_cv_callstmt_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->0
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),argument)
        then add (op_cv_expr (lds_sb (s,sb),
                  op_cv_callstmt_1 (rest,sb))
        else op_cv_callstmt_1 (rest,sb)
      fi
   esac ;

lösche_null_komp (l):=
   case l is
   *nil_ljj-->nil_ljj
   *cons_ljj (l1,rest)-->
      if eq.nat (sjj_2 (l1),0)
        then lösche_null_komp (rest)
        else cons_ljj (l1,lösche_null_komp (rest)
      fi
   esac ;

ENDSPEC
```

3.2.4.5. Externe Prozeduren

3.2.4.5.1. Beschreibung

Diese Funktion liefert Informationen über die im Quellmodul verwendeten externen Prozeduren. Die zu liefernde Information besteht aus:

1. dem Modulnamen
2. der Statementlänge des Moduls
3. der Liste mit Informationen über die im Quellmodul verwendeten externen Prozeduren, bestehend aus:
   1. dem Namen des Entry
   2. dem Typ des R ckgabewertes bei Funktionen
   3. der Liste der Typen der formalen Parameter der Deklarationen der externen Prozeduren
   4. dem Ort der Definition, gegeben durch die Statementnummer, bzw. bei mehrfacher Deklaration, der Liste dieser Statementnummern
   5. dem Ort des Auftretens der externen Prozedur, d.h. die Statementnummer aller Statements, in denen die Prozedur aufgerufen wird
   6. der Anzahl der Aufrufe
4. der Häufigkeit der ext. Prozeduren, die definiert ist als das Verhältnis Anzahl der Aufrufe zur Statementlänge des Moduls

Aufbau der Ausgabe:

<table>
<tr><td colspan="3">Modulname</td></tr>
<tr><td colspan="3">Statementlaenge Modul</td></tr>
<tr><td>Name Ext. Proz. 1</td><td>Name 2</td><td rowspan="8">.....</td></tr>
<tr><td>Return Attribut</td><td rowspan="7">.....</td></tr>
<tr><td>Liste (Typ der formalen Parameter)</td></tr>
<tr><td>Ort der Definition</td></tr>
<tr><td>Ort des Auftretens</td></tr>
<tr><td>Anzahl der Aufrufe</td></tr>
<tr><td>Haeufigkeit</td></tr>
</table>

3.2.4.5.2. Spezifikation Externe Prozeduren

```
INSTANTIATE Tupel to Param_Type
ACTUALIZE : #x1, #x2 by Charstring
            SORTS : #x1.elem, #x2.elem by charstring
RENAME : SORTS : tupel by param_type
         OPS : erz_tup by erz_pt
END


INSTANTIATE Liste to L(Param_Type)
ACTUALIZE : #x by Param_Type
            SORTS : #x.elem by param_type
RENAME : SORTS : tupel by l(param_type)
         OPS : nil_by_nil_lpt
END


INSTANTIATE 6_Tupel to Stat_Ext_Proz
ACTUALIZE : #x1 by Name
            #x2 by Param_Type
            #x3 by L(Param_Type)
            #x4, #x5 by L(Nat)
            #x6 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by param_type
                    #x3.elem by l(param_type)
                    #x4.elem, #x5.elem by l(nat)
                    #x6.elem by naz
RENAME : SORTS : 6_tupel by stat_ext_proz
'0 erz_sechs by erz sep
END


INSTANTIATE Liste to L(Stat_Ext_Proz)
ACTUALIZE : #x by Stat_Ext_Proz
            SORTS : #x.elem by stat_ext_proz
RENAME : SORTS : liste by l(stat_ext_proz)
         OPS : nil by nil_lsep
END


INSTANTIATE 4_Tupel to Ext_Prozeduren_Stat
```

```
ACTUALIZE : #x1 by Name
            #x2 by Nat
            #x3 by L(Stat_Ext_Proz)
            #x4 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by nat
                    #x3.elem by l(stat_ext_proz)
                    #x4.elem by nat
RENAME : SORTS : 4_tupel by ext_prozeduren_stat
         OPS : erz_vier by erz_extproz
END
```

SSPEC EXTERNE PROZEDUREN

USE SSPECS : Schnittstelle, Ext_Prozedur_Stat,
Vollständigkeit der Komponenten, Allgemein

PUBLIC OPS :

```
op_externe_prozeduren:name l(dat)-->ext_prozeduren_stat
op_parameter_attr: name_nat sb-->l(param_type)
stmtlänge_uo:sb-->nat
change_name_to_charstring:name-->charstring
op_längen_attribut:l(stammsatz) sb-->charstring
op_return_attribut:name_nat sb-->param_type
```

PROPERTIES :

Die Operation op_return_attribut (ns,sb) liefert für das durch ns gekennzeichnete Declare-Stmt eines externen Entries, sofern es sich um eine Funktionsprozedur handelt, den Typ des Rückgabeparameters einschließlich Längenspezifikation. Bei einer eigentlichen Prozedur wird der Hinweis geliefert, daß keine Funktion vorliegt. Die Längenspezifikation ist leer, wenn der Rückgabeparameter ein Label ist.

Beispiele für das RETURNS Attribut:

1. DECLARE F ENTRY RETURNS (INTEGER);
2. FP : PROCEDURE RETURNS (CHARACTER (8));

Das Returns Attribut kann nicht vom Typ Entry sein.

```
*/
∀ name_nat ns: ∀ sb sb: ∃ param_type pt:
  pt=op_return_attribut (ns,sb)
  ∧ (pt=erz_pt (keine Funktion,leer)
   ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
       ∧ ssl_6 (ssts_1 (s))=returns attr
       ∧ ssl_7 (ssts_1 (s))=sni_2 (ns)
       ∧ pt=erz_pt (op_type_attr (lds_sb (s,sb),sb),
```



```
                    op_längen_attr (lds_sb (s,sb),sb)))

/*
ort_d_aufrufs_ep (ns,sb) liefert die Liste der Stmtnrn aller Stammsätze im SB, in denen der durch ns gekennzeichnete Entry aufgerufen wird.
*/
∀ name_nat ns: ∀ sb sb: ∃ l(nat) l:
  l=ort_d_aufrufs_ep (ns,sb)
  ∧ ∀ nat n: ist_enth_lnat (n,l)=true
      ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ sdt_3 (ssts_2 (s))=sni_1 (ns)
          ∧ ssl_7 (ssts_1 (s))=n
          ∧ (ssl_6 (ssts_1 (s))=callext
             ∨ ssl_6 (ssts_1 (s))=callextfct)
          ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
                ∧ sdt_3 (ssts_2 (s'))=sni_1 (ns)
                ∧ (ssl_6 (ssts_1 (s'))=callext
                   ∨ ssl_6 (ssts_1 (s'))=callextfct)
                ∧ ist_enth_lnat (ssl_7 (ssts_1 (s')),l)=false

/*
op_parameter_attr (ns,sb) liefert für den durch ns gekennzeichneten externen Entry die Liste der Typen der Parameter einschließlich der Längenspezifikation.

Beispiel: DECLARE E ENTRY (POINTER,1) RETURNS (FIXED BINARY);
*/
∀ sb sb: ∀ name_nat ns:
  ∃ l(param_type) l: l=op_parameter_attr (ns,sb)
  ∧ (ist_leer_lpt (l)=true
     ⇒ ∀ param_type pt: ist_enth_lpt (pt,l)=true
         ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
             ∧ ssl_7 (ssts_1 (s))=sni_2 (ns)
             ∧ ssl_6 (ssts_1 (s))=parameter type
             ∧ pt=erz_pt (param_attr_2 (lds_sb (s,sb),sb),
                          op_längen_attr (lds_sb (s,sb),sb)))

/*
param_attr_2 (lsb,sb) liefert den Typ des Parameters, aufgrund der angebenen Liste von Stammsätzen lsb. Dies ist die Liste der Söhne des Stammsatzes mit Satzart "parameter type".
*/
∀ sb sb: ∀ l(stammsatz) lsb: ∃ charstring pa:
  pa=param_attr_2 (lsb,sb)
  ∧ (pa≠error
     ⇒ ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
         ∧ ((sdt_3 (ssts_2 (s))=pa
             ∧ ssl_6 (ssts_1 (s))=unsigned integer constant)
```

```
            v (ssl_6 (ssts_1 (s))=type attr
             ∧ pa=op_bin_fix (lds_sb (s,sb)))))

/*
op_bin_fix (lsb) liefert als Ergebnis ein Typ Attribut.
*/
∀ l(stammsatz) lsb: ∃ charstring pa: pa=op_bin_fix (lsb)
  ∧ (pa=no type attribut
     v (pa=binary fixed
        ∧ (ssl_6 (ssts_1 (car_lsb (lsb)))=fixed
           v ssl_6 (ssts_1 (car_lsb (lsb)))=binary fixed))
     v pa=ssl_6 (ssts_1 (car_lsb (lsb))))

/*
op_type_attribut (lsb,sb) liefert zu dem in lsb existierenden Stamm-
satz mit Satzart "type attribut" den Datentyp, der mit diesem Stamm-
satz verbunden ist.
*/
∀ l(stammsatz) lsb: ∀ sb sb:
  ∃ param_attribut pa: pa=op_type_attribut (lsb,sb)
   ∧ (pa≠error
      ⇒ ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
           ∧ ssl_6 (ssts_1 (s))=type attribut
           ∧ pa=op_bin_fix (lds_sb (s,sb)))

/*
ort_d_def_ep (ns,lns) liefert die Liste der Stmtnrn der Elemente aus
lns, die mit dem Entrynamen aus ns übereinstimmen.
*/
∀ name_nat ns: ∀ l(name_nat) lns:
  ∃ l(nat) l: l=ort_d_def_ep (ns,lns)
   ∧ ∀ nat n: ist_enth_lnat (n,l)=true
        ⇒ (sni_2 (ns)=n
           ∧ ¬ ∃ name_nat ni: ist_enth_lni (ni,lns)=true
                ∧ sni_1 (ni)=sni_1 (ns)
                ∧ ist_enth_lnat (sni_2 (ni),ln)=false)

/*
längen_attr_3 (lsb) liefert die Konkatenation aller Inhalte aller
Stammsätze aus lsb mit Satzart "declare identifier" getrennt durch ei-
nen Punkt bei Satzart ".".
*/
∀ stammsatz lsb: ∃ längen_attr la: la=längen_attr_3 (lsb)
  ∧ (la=blank
     v (la=concat (change_name_to_charstring (sdt_3 (ssts_2 (
           car_lsb (lsb)),längen_attr_3 (cdr_lsb (lsb))
        ∧ ssl_6 (ssts_1 (car_lsb (lsb)))=declare identifier)
     v (ssl_6 (ssts_1 (car_lsb (lsb)))=.
```



```
        ∧ la=concat (.,längen_attr_3 (cdr_lsb (lsb))))
     v la=error)

/*
op_längen_attribut (lsb,sb) liefert die Längenspezifikation, die zum
Stammsatz aus lsb mit Satzart "type attribut" gehört.
*/
∀ sb sb: ∀ l(stammsatz) lsb: ∃ längen_attribut la:
  la=op_längen_attribut (lsb,sb)
  ∧ (la≠error
     ⇒ ∃ stammsatz s: ist_enth_lsb (s,sb)=true
          ∧ ssl_6 (ssts_1 (s))=type attribut
          ∧ ∃ stammsatz s1: ist_in_baum_sb (s1,
                              t_baum_sb (s,sb))=true
               ∧ ((ssl_6 (ssts_1 (s1))=* ∧ la=*)
                  v (ssl_6 (ssts_1 (s1))=
                      unsigned integer constant
                    ∧ la=change_name_to_charstring (
                           sdt_3 (ssts_2 (s1)))
                  v (ssl_6 (ssts_1 (s1))=unsubscripted reference
                    ∧ la=längen_attr_3 (blätter_sb (
                           t_baum_sb (s1,sb))))))

PRIVATE OPS :
ort_d_def_ep:name_nat l(name_nat)-->l(nat)
ort_d_aufrufs_ep:name_nat sb-->l(nat)
ort_d_aufrufs_ep_1:name_nat l(stammsatz)-->l(nat)
gen_l_stat_ext_proz:sb-->l(stat_ext_proz)
gen_l_stat_ext_proz_1:l(stammsatz) sb-->l(stat_ext_proz)
op_stat_ext_proz:name_nat l(name_nat) sb-->stat_ext_proz
lösche_id_ext_entries:name_nat l(name_nat)-->l(name_nat)
op_externe_prozeduren_1:name sb-->ext_prozeduren_stat
return_attr:l(stammsatz) sb-->param_type
op_type_attr:l(stammsatz) sb-->charstring
op_bin_fix:l(stammsatz) sb-->charstring
längen_attr_1:l(stammsatz) sb-->charstring
längen_attr_2:l(stammsatz) sb-->charstring
längen_attr_3:l(stammsatz)-->charstring
param_attr_1:l(stammsatz) sb-->l(param_type)
param_attr_2:l(stammsatz) sb-->charstring

DEFINE OPS :
op_externe_prozeduren (n,ld):=
  op_externe_prozeduren_1     (lies_sb (n,liefere_eind_modul (n,ld));

op_externe_prozeduren_1 (n,sb):=
   erz_extproz (n,
                stmtlänge_uo (sb),
```

```
                gen_l_stat_ext_proz (sb),
                div (mult (anzahl_lep (gen_l_stat_ext_proz(sb)),100),
                     stmtlänge_uo (sb)));

stmtlänge_uo (sb):=
   diff_zp (sdt_5 (ssts_2 (wurzel_sb (sb))));

gen_l_stat_ext_proz (sb):=
   gen_l_stat_ext_proz_1 (gen_l_dcl_ext_entries (sb),sb);

gen_l_stat_ext_proz_1 (l,sb):=
   case l is
   *nil_lni-->nil_lep
   *cons_lni (l1,rest)-->
     cons_lep (op_stat_ext_proz (l1,rest,sb),
               gen_l_stat_ext_proz_1 (
                 lösche_id_ext_entries (l1,rest),sb)
   esac ;

lösche_id_ext_entries (ns,lns):=
   case lns is
   *nil_lni-->nil_lni
   *cons_lni (l1,rest)-->
      if eq.name (sni_1 (ns), sni_1 (l1))
        then lösche_id_ext_entries (ns,rest)
        else cons_lni (l1,lösche_id_ext_entries (ns,rest))
      fi
   esac ;

ort_d_aufrufs_ep (ns,sb):=
   ort_d_aufrufs_ep_1 (ns,preorder_sb (sb));

ort_d_aufrufs_ep_1 (ns,lsb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_1 (s)),callext)
          or eq.charstring (ssl_6 (ssts_1 (s)),callextfct))
         and eq.name (sdt_3 (ssts_2 (s)),sni_1 (ns))
        then cons_lnat (ssl_7 (ssts_1 (s)),
                        ort_d_aufrufs_ep_1 (ns,rest))
        else ort_d_aufrufs_ep_1 (ns,rest)
      fi
   esac ;

change_name_to_charstring (n):=
   case n is
   *bl-->blank
```



```
   *cr (x,rest)-->create (x,change_name_to_charstring (rest))
   esac ;

op_stat_ext_proz (ns,lns,sb):=
   erz_sep (sni_1 (ns),
            op_return_attribut (ns,sb),
            op_param_attr (ns,sb),
            ort_d_def_ep (ns,lns),
            ort_d_aufrufs_ep (ns,sb),
            anzahl_lnat (ort_d_aufrufs_ep (ns,sb)));

op_param_attr (ns,sb):=
   param_attr_1 (gen_l_stmt_nr (sni_2 (ns),preorder_sb (sb)),sb);

param_attr_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lpt
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),parameter type)
        then cons_lpt (
               erz_pt (param_attr_2 (lds_sb (s,sb),sb),
                       op_längen_att (lds_sb (s,sb),sb)),
               param_attr_1 (rest,sb)
        else param_attr_1 (rest,sb)
      fi
   esac ;

param_attr_2 (lsb,sb):=
   case lsb is
   *nil_lsb-->error.charstring
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),type attr)
        then op_bin_fix (lds_sb (s,sb))
        else if eq.charstring (ssl_6 (ssts_1 (s),
                                      unsigned integer constant)
               then change_name_to_charstring (sdt_3 (ssts_2 (s)))
               else param_attr_2 (rest,sb)
             fi
    fi
   esac ;

op_bin_fix (lsb):=
   case lsb is
   *nil_lsb-->no type attribut
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),binary fixed)
         or eq.charstring (ssl_6 (ssts_1 (s)),fixed)
        then binary fixed
```

```
      else ssl_6 (ssts_1 (s))
    fi
  esac ;

op_längen_attr (lsb,sb):=
  case lsb is
  *nil_lsb-->error.charstring
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),type attribut)
      then längen_attr_1 (lds_sb (s,sb),sb)
      else op_längen_attr (rest,sb)
    fi
  esac ;

längen_attr_1 (lsb,sb):=
  case lsb is
  *nil_lsb-->leer
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),size/length)
      or eq.charstring (ssl_6 (ssts_1 (s)),
                        size/length specification)
      or eq.charstring (ssl_6 (ssts_1 (s)),
                        precision specification)
      then längen_attr_2 (lds_sb (s,sb))
      else längen_attr_1 (rest,sb)
    fi
  esac ;

ort_d_def_ep (ns,lns):=
  case lns is
  *nil_lni-->cons_lnat (sni_1 (ns),nil_lnat)
  *cons_lni (ns1,rest)-->
    if eq.name (sni_1 (ns),sni_1 (ns1))
      then cons_lnat (sni_2 (ns1),ort_d_fef_ep (ns,rest))
      else ort_d_def_ep (ns,rest)
    fi
  esac ;

op_return_attribut (ns,sb):=
  return_attr (gen_l_stmtnr (sni_2 (ns),sb),sb);

return_attr (lsb,sb):=
  case lsb is
  *nil_lsb-->erz_pt (keine Funktion,leer)
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),returns attr)
      then erz_pt (op_type_attr (lds_sb (s,sb),sb),
                   op_längen_attr (lds_sb (s,sb),sb))
```



```
      else return_attr (rest,sb)
    fi
  esac ;

op_type_attr (lsb,sb):=
  case lsb is
  *nil_lsb-->error.charstring
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),type attribut)
      then op_bin_fix (lds_sb (s,sb))
      else op_type_attr (rest,sb)
    fi
  esac ;

längen_attr_2 (lsb,sb):=
  case lsb is
  *nil_lsb-->default
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),
                      unsigned integer constant)
      then sdt_3 (ssts_2 (s))
      else if eq.charstring (ssl_6 (ssts_1 (s)),*)
             then *
             else if eq.charstring (ssl_6 (ssts_1 (s)),
                                    unsubscripted reference)
                    then längen_attr_3 (blätter_sb (
                                        t_baum_sb (s,sb)))
                    else if eq.charstring (ssl_6 (ssts_1 (s)),
                           then längen_attr_2 (
                                lds_sb (s,sb),sb)
                           else längen_attr_2 (rest,sb)
                         fi
                  fi
           fi
    fi
  esac ;

längen_attr_3 (lsb):=
  case lsb is
  *nil_lsb-->blank
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),declare identifier)
      then concat (change_name_to_charstring (
                   sdt_3 (ssts_2 (s)),längen_attr_3 (rest))
      else if eq.charstring (ssl_6 (ssts_1 (s)),.)
             then concat (.,längen_attr_3 (rest))
             else error.charstring
           fi
```

```
      fi
   esac ;

ENDSPEC
```


3.2.4.6. Interne Prozeduren

3.2.4.6.1. Beschreibung

Dem Anwender dieser Funktion werden konkrete Informationen über seine im Quellprogramm verwendeten internen Prozeduren geliefert.

Diese Information besteht aus:

1. dem Quellmodulnamen
2. der Statementlänge des Moduls
3. der Liste mit Informationen über die enthaltenen internen Prozeduren, bestehend aus:
   1. der Liste der Namen der int. Prozedur
   2. der Länge der Prozedur in Statements
   3. dem Typ des Returns bei Funktionen
   4. der Liste der Typen der formalen Parameter
   5. dem Ort der Definition (Statementnummer)
   6. den Orten der Aufrufe (Statementnummern)
4. der Häufigkeit der internen Prozeduren im Verhältnis zur Statementlänge des Moduls

Aufbau der Ausgabe:

<table>
<tr><td colspan="3">Modulname</td></tr>
<tr><td colspan="3">Stmtlaenge Modul</td></tr>
<tr><td>L(Name int.Proz.1)</td><td>L(Name 2)</td><td rowspan="8">.....</td></tr>
<tr><td>Laenge in Stmts</td><td rowspan="7">.....</td></tr>
<tr><td>Return Attribut</td></tr>
<tr><td>Liste (Typ der form. Parameter)</td></tr>
<tr><td>Ort der Definition</td></tr>
<tr><td>Orte des Aufrufs</td></tr>
<tr><td>Anzahl der Aufrufe</td></tr>
<tr><td colspan="3">Haeufigkeit der internen Prozeduren</td></tr>
</table>

3.2.4.6.2. Spezifikation Interne Prozeduren

```
INSTANTIATE 7_Tupel to Stat_Int_Proz
ACTUALIZE : #x1 by L(Name
            #x2 by Nat
            #x3 by Param_Type
            #x4 by L(Param_Type)
            #x5 by Nat
            #x6 by L(Nat)
            #x7 by Nat
            SORTS : #x1.elem by l(name)
                    #x2.elem by nat
                    #x3.elem by param_type
                    #x4.elem by l(param_type)
                    #x5.elem by nat
                    #x6.elem by l(nat)
                    #x7.elem by nat
RENAME : SORTS : 7_tupel by stat_int_proz
         OPS : erz_sieben by erz_sip
END


INSTANTIATE Liste to L(Stat_Int_Proz)
ACTUALIZE : #x by Stat_Int_Proz
            SORTS : #x.elem by stat_int_proz
RENAME : SORTS : liste by l(stat_int_proz)
         OPS : nil by nil_lsip
END


INSTANTIATE 4_Tupel to Int_Prozeduren_Stat
ACTUALIZE : #x1 by Name
            #x2 by Nat
            #x3 by L(Stat_Int_Proz)
            #x4 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by nat
                    #x3.elem by l(stat_int_proz)
                    #x4.elem by nat
RENAME : SORTS : 4_tupel by int_prozeduren_stat
         OPS : erz_vier by erz_intproz
END
```


SSPEC INTERNE PROZEDUREN

USE SSPECS : Bool, Schnittstelle, Allgemein, Benutzthierarchie, Externe Prozeduren, Vollständigkeit der Externverweise, Int_Prozeduren_Stat, L(Nat)

PUBLIC OPS :
op_interne_prozeduren:name l(dat)-->int_prozeduren_stat
gen_l_stat_int_proz:sb-->l(stat-int_proz)
op_stat_int_proz:name_nat sb-->stat_int_proz
gen_l_typ_param:l(name) l(stammsatz) sb-->l(param_type)
ort_d_aufrufs_ip:l(name) sb name_nat-->l(nat)
gen_l_blöcke_mit_neuer_def:name_nat sb-->l(nat)

PROPERTIES :
/*

Die Operation gen_l_blöcke_mit_neuer_def (ns,sb) liefert alle Blocknummern der Blöcke im SB, die im Block mit der Prozedurdeklaration enthalten sind, in denen der Prozedurname eine neue Definition als Entry Constante oder als Entry Variable erhält.

Die Hilfsfunktion gen_l_nat_neue_def_entry_constante (ns,lsb,sb) liefert die Liste der Blocknummern der Blöcke, in denen der Entryname aus ns als Entry Constante neu definiert ist. Beim Aufruf des Entries in diesen Blöcken handelt es sich nicht, um einen Aufruf des durch ns gekennzeichneten Entries. lsb ist die Liste der Stammsätze des Teils des SB, der dem Begin Block bzw. dem Prozedurblock entspricht, der die Deklaration des durch ns gegebenen Entries enthält. In diesem Block ist nach einem Aufruf zu suchen.

Die Hilfsfunktion gen_l_nat_neue_def_entry_variable (ns,lsb,sb) liefert die Liste der Blocknummern der Blöcke, in denen der durch ns gegebene Entry in eine Entry Variable umdefiniert wird.

Beispiel:

```
M : PROCEDURE;
    X : PROCEDURE;
        ...
    END X;
    CALL X;
    A : PROCEDURE;
        ...
        X : PROCEDURE (...);
            ...
        END X;
        CALL X (...);
        ...
    END A;
END M;
```

```
M : PROCEDURE;
    X : PROCEDURE;
        ...
    END X;
    CALL X;
    A : PROCEDURE;
        ...
        DCL X ENTRY VARIABLE;
        ...
    END A;
    ...
END M;
```

In beiden Beispielen hat X in Prozedurblock M eine andere Bedeutung als in Prozedurblock A.
*/
```
V sb sb: V name_nat ns: V nat n:
  ist_enth_lnat (n,gen_l_blöcke_mit_neuer_def (ns,sb))=true
  ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
       ∧ ((sdt_10 (ssts_2 (s))=n
           ∧ ist_gleicher_entry (sni_1 (ns),
               gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb)=true
           ∧ ssl_6 (ssts_1 (s))=variable)
          v ((ssl_6 (ssts_1 (s))=proc stmt
              v ssl_6 (ssts_1 (s))=secondary entry)
           ∧ (sdt_10 (ssts_2 (enth_vater_proc_beg (
                enth_vater_proc_beg (s,sb),sb)))=n
              v sdt_10 (ssts_2 (s))=n)))
       ∧ ist_enth_lnat (block_zu_stmtnr (sni_2 (ns),sb),
            gen_l_blöcke_mit_neuer_def (ns,sb))=false
```

/*
ist_gleicher_entry (e,lsb) liefert true, falls es unter den Stammsätzen der Liste lsb einen Stammsatz mit Satzart "declare identifier" gibt, der im Namen mit e übereinstimmt.
*/
```
V name e: V l(stammsatz) lsb:
  ist_gleicher_entry (e,lsb)=true
  ⇒ ∃ stammsatz s: ist_enth_lsb (s,sb)=true
       ∧ ssl_6 (ssts_1 (s))=declare identifier
       ∧ sdt_3 (ssts_2 (s))=e
```

/*
ort_d_aufrufs_ip (l,sb,ns) liefert die Liste der Stmtnummern der Statements, in denen die Prozedur, die durch die Namensliste l (Liste sämtlicher Entrynamen) und dem Tupel ns (äußerste Entryname und Statementnummer) gekennzeichnet ist, aufgerufen wird. Zu beachten ist, daß ein Prozedurname nicht in einem inneren Block eine andere Prozedur kennzeichnet.
```
V l(name) l: V sb sb: ∃ l(nat) ln:
  ln=ort_d_aufrufs_ip (l,sb,ns)
  ∧ (ist_leer_lnat (ln)=true
     v V nat n: ist_enth_lnat (n,ln)=true
       ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ (ssl_6 (ssts_1 (s))=call
             v ssl_6 (ssts_1 (s))=callfct)
          ∧ ssl_7 (ssts_1 (s))=n
          ∧ ist_enth_ln (sdt_3 (ssts_2 (s)),l)=true
          ∧ ist_enth_lnat (sdt_10 (ssts_2 (s)),
              gen_l_blöcke_mit_neuer_def (erz_ni (
               sdt_3 (ssts_2 (s)),sni_2 (ns),
                t_baum_sb (enth_vater_proc (
```



```
                v_qual_proc_name_stmtnr (ns,sb),sb),sb)))=false)))
        ∧ ¬ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
             ∧ (ssl_6 (ssts_1 (s'))=call
                v ssl_6 (ssts_1 (s'))=callfct)
             ∧ ssl_7 (ssts_1 (s))≠n
             ∧ ist_enth_lnat (sdt_10 (ssts_2 (s')),
                 gen_l_blöcke_mit_neuer_def (erz_ni (
                  sdt_3 (ssts_2 (s)),sni_2 (ns),
                   v_qual_proc_name_stmtnr (ns,sb),sb),sb)))
                                                  =false)
```

/*
Die Operation gen_l_typ_param (ln,lsb,sb) wird mit den aktuellen Parametern ln (Liste der Namen der formalen Parameter der zu untersuchenden Prozedur), lsb (Liste der Stammsätze der Söhne des Stammsatzes der Satzart "procedure" der aktuellen Prozedur) und dem SB sb aufgerufen.

Im zur Prozedur gehörenden Declaration Part wird zu jedem Namen aus ln das Declare-Statement gesucht. Aus diesem Declare-Statement wird der Parametertyp einschließlich Längenspezifikation ermittelt, indem alle Stammsätze mit zum Declare-Statement identischer Statementnummer durchsucht werden. Abhängig von den Satzarten der Stammsätze läßt sich der Parametertyp angeben.
*/
```
V l(name) ln: V l(stammsatz) lsb: V sb sb:
  ∃ l(param_type) lpt: lpt=gen_l_typ_param (ln,lsb,sb)
   ∧ (lpt=error v ist_leer_lpt (lpt)=true
      v V param_type pt: ist_enth_lpt (pt,lpt)=true
          ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
               ∧ ((pt=erz_pt (Struktur,leer)
                  ∧ ssl_6 (ssts_1 (s))=structure declaration)
                 v (pt=erz_pt (array,leer)
                  ∧ ssl_6 (ssts_1 (s))=dimension attribut)
                 v (pt=erz_pt (set,leer)
                    ∧ ssl_6 (ssts_1 (s))=set declaration)
                 v pt=erz_pt (op_type_attr (s,sb),
                                op_längen_attr (s,sb)))
            ∧ ∃ stammsatz s1: ist_in_baum_sb (s1,sb)=true
               ∧ ssl_7 (ssts_1 (s1))=ssl_7 (ssts_1 (s))
               ∧ ssl_6 (ssts_1 (s1))=declare identifier
               ∧ (∃ name n: ist_enth_ln (n,ln)=true
                  ∧ sdt_s (ssts_2 (s1))=n)
               ∧ (∃ stammsatz s2: ist_enth_lsb (s2,lsb)=true
                  ∧ ssl_6 (ssts_1 (s2))=declaration part
                  ∧ ist_in_baum_sb (s1,t_baum_sb (s2,sb))=true)
```

PRIVATE OPS :
```
op_interne_prozedur_1:name sb-->int_proz_stat
gen_l_stat_int_proz_1:l(name_nat) sb-->stat_int_proz
```

```
gen_l_typ_param_1:l(name) l(stammsatz) sb-->l(param_type)
gen_typ_param:name l(stammsatz) sb-->param_type
gen_typ_spez:l(stammsatz) sb-->param_type
gen_typ_spez_1:l(stammsatz) sb-->param_type
ort_d_aufrufs_ip_1:name_nat sb-->l(nat)
ort_d_aufrufs_ip_2:name_nat l(stammsatz) l(nat)-->l(nat)
gen_l_nat_neue_def_entry_constante:name_nat l(stammsatz)
                                                  sb-->l(nat)
gen_l_nat_neue_def_entry_variable,
gen_l_nat_neue_def_entry_variable_1,
gen_l_nat_neue_def_entry_variable_2:
                              name_nat l(stammsatz) sb-->l(nat)
ist_gleicher_entry:name l(stammsatz)-->bool

DEFINE OPS :
op_interne_prozeduren (n,ld):=
   op_interne_prozedur_1 (n,lies_sb (n,
                              liefere_eind_modul (n,ld)));

op_interne_prozedur_1 (n,sb):=
   erz_int_proz (n,
                 stmtlänge_uo (sb),
                 gen_l_stat_int_proz (sb),
                 div (mult (anzahl_lip (
                             gen_l_stat_int_proz (sb)),100),
                      stmtlänge_uo (sb));

gen_l_stat_int_proz (sb):=
   gen_l_stat_int_proz_1 (gen_l_int_proz (sb),sb));

op_stat_int_proz (ns,sb):=
   let x=v_qual_proz_name_stmtnr (snid_1 (ns),snid_2(ns),sb) in
    erz_ip (gen_l_entrypoints (x,sb),
            stmtlänge_uo (t_baum_sb (x,sb)),
            op_return_attribut (ns,sb),
            gen_l_typ_param (
              gen_l_name_form_par (ns,sb), lds_sb (x,sb),sb),
            sni_2 (ns),
            ort_d_aufrufs_ip (
              gen_l_entrypoints (x,sb),sb,ns),
            anzahl_lnat (ort_d_aufrufs_ip (
                          gen_l_entrypoints (x,sb),sb,ns)));

gen_l_stat_int_proz_1 (lns,sb):=
   case lns is
   *nil_lni-->nil_lip
   *cons_lni (ns1,rest)-->
      cons_lip (op_stat_int_proz (ns1,sb),
```



```
                gen_l_stat_int_proz_2 (rest,sb))
   esac ;

gen_l_blöcke_mit_neuer_def (ns,sb):=
   let x=v_qual_sts_proz_beg_zu_blocknr (
        blocknr_zu_stmtnr (sni_2 (ns),sb)),sb) in
   let y=preorder_sb (t_baum_sb (x,sb)) in
    append_lnat (
       gen_l_nat_neue_def_entry_constante (ns,y,sb),
       gen_l_nat_neue_def_entry_variable (ns,y,sb));

gen_l_nat_neue_def_entry_constante (ns,lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_2 (s)),proc stmt)
          or eq.charstring (ssl_6 (ssts_2 (s)),secondary entry))
        and ist_enth_ln (sdt_3 (ssts_2 (s)),
                         gen_l_entrypoints (s,sb))
        and not eq.nat (ssl_7 (ssts_1 (s)),sni_2 (ns))
       then cons_lnat (sdt_10 (ssts_2 (s)),
                cons_lnat (sdt_10 (ssts_2 (
                  enth_vater_proc_beg (
                    enth_vater_proc_beg (s,sb),sb))),
                gen_l_nat_neue_def_entry_constante (ns,rest,sb)))
       else gen_l_nat_neue_def_entry_constante (ns,rest,sb)
      fi
   esac ;

gen_l_nat_neue_def_entry_variable (ns,lsb,sb):=
   gen_l_nat_neue_def_entry_variable_1 (ns,
     gen_l_sts_sa_dcl_part (lsb),sb);

gen_l_nat_neue_def_entry_variable_1 (ns,lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
      append_lnat (
        gen_l_nat_neue_def_entry_varible_2 (ns,
           preorder_sb (t_baum_sb (s,sb)),sb),
           gen_l_nat_neue_def_entry_variable_1 (ns,rest,sb)
   esac ;

gen_l_nat_neue_def_entry_variable_2 (ns,lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lnat
  *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_2 (s)),variable)
```

```
        and ist_gleicher_entry (sni_1 (ns),
                                gen_l_stmt_nr (ssl_7 (ssts1 (s)),sb))
      then cons_lnat (sdt_10 (ssts_2 (s)),
           gen_l_nat_neue_def_entry_variable_2 (ns,rest,sb)
      else gen_l_nat_neue_def_entry_variable_2 (ns,rest,sb)
    fi
  esac ;

ist_gleicher_entry (e,lsb):=
  case lsb is
  *nil_lsb-->false
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_2 (s)),declare identifier)
       and eq.name (sdt_3 (ssts_2 (s)),e)
      then true
      else ist_gleicher_entry (e,rest)
    fi
  esac ;

ort_d_aufrufs_ip (ln,sb,ns):=
  case ln is
  *nil_ln-->nil_lnat
  *cons_ln (n,rest)-->
    append_lnat (ort_d_aufrufs_ip_1 (erz_ni (sni_2 (ns)),
                 t_baum_sb (enth_vater_proz_beg (
                   v_qual_proz_name_stmt_nr (ns,sb),sb),sb)),
                 ort_d_aufrufs_ip (rest,sb,ns))
  esac ;

ort_d_aufrufs_ip_1 (ns,sb):=
  ort_d_aufrufs_ip_2 (sni_1 (ns),preorder_sb (sb),
                      gen_l_blöcke_mit_neuer_def (ns,sb));

ort_d_aufrufs_ip_2 (n,lsb,lnat):=
  case lsb is
  *nil_lsb-->nil_lnat
  *cons_lsb (s,rest)-->
    if (eq.charstring (ssl_6 (ssts_2 (s)),call)
        or eq.charstring (ssl_6 (ssts_2 (s)),callfct))
       and eq.name (sdt_3 (ssts_2 (s)),n)
       and not ist_enth_lnat (sdt_10 (ssts_2 (s)),lnat)
      then cons_lnat (ssl_7 (ssts_1 (s)),
                      ort_d_aufrufs_ip_2 (n,rest,lnat))
      else ort_d_aufrufs_ip_2 (n,rest,lnat)
    fi
  esac ;

gen_l_typ_param (ln,lsb,sb):=
```



```
  case lsb is
  *nil_lsb-->error.l(param_type)
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_2 (s)),declaration part)
      then gen_l_typ_param_1 (ln,preorder_sb (t_baum_sb (s,sb)),sb)
      else gen_l_typ_param_1 (ln,rest,sb)
    fi
  esac ;

gen_l_typ_param_1 (ln,lsb,sb):=
  case lsb is
  *nil_lsb-->nil_lpt
  *cons_lsb (s,rest)-->
    cons_lpt (gen_typ_param (n,lsb,sb),
              gen_l_typ_param_1 (rest,lsb,sb))
  esac ;

gen_typ_param (n,lsb,sb):=
  case lsb is
  *nil_lsb-->error.param_type
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_2 (s)),declare identifier)
       and eq.name (sdt_3 (ssts_2 (s)),n)
      then gen_typ_spez (gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb)),sb)
      else gen_typ_param (n,rest,sb)
    fi
  esac ;

gen_typ_spez (lsb,sb):=
  case lsb is
  *nil_lsb-->gen_typ_spez_1 (lsb,sb)
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_2 (s)),structure declaration)
      then erz_pt (Struktur,leer)
      else gen_typ_spez (rest,sb)
    fi
  esac ;

gen_typ_spez_1 (lsb,sb):=
  case lsb is
  *nil_lsb-->erz_pt (op_type_attr (lsb,sb),
                     op_längen_attr (lsb,sb))
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_2 (s)),set declaration)
      then erz_pt (Set,leer)
      else if eq.charstring (ssl_6 (ssts_2 (s)),dimension attribut)
              and not ist_leer_lsb (lds_sb (s,sb))
             then erz_pt (Array,leer)
```

```
            else gen_typ_spez (rest,sb)
          fi
    fi
  esac ;

ENDSPEC
```


3.2.4.7. Spezifikation Einfache Statistik

```
INSTANTIATE 6_Tupel to Einf_Statistik
ACTUALIZE : #x1 by Allg_Stat
            #x2 by Anw_Stat
            #x3 by Datentyp_Stat
            #x4 by Konvertierungen_Stat
            #x5 by Externe_Prozeduren_Stat
            #x6 by Interne_Prozeduren_Stat
            SORTS : #x1.elem by allg_stat
                    #x2.elem by anw_stat
                    #x3.elem by datentyp_stat
                    #x4.elem by konvertierungen_stat
                    #x5.elem by externe_prozeduren_stat
                    #x6.elem by interne_prozeduren._stat
RENAME : SORTS : 6_tupel by einf_statistik
           OPS : erz_sechs by erz_einf_stat
END


SSPEC EINFACHE_STATISTIK

USE SSPECS : Einf_Stat

PUBLIC OPS :
op_einfache_statistik:name fenster l(dat) anw_grp dattyp
                                                  -->einf_statistik

DEFINE OPS :
op_einfache_statistik (n,f,ld,a_grp,d_typ):=
  erz_einf_stat (op_allgemein (n,f,ld),
                 op_anweisung (n,ld,a_grp),
                 op_datentyp (n,d_typ,ld),
                 op_konvertierungen (n,ld),
                 op_externe_prozeduren (n,ld),
                 op_interne_prozeduren (n,ld));

ENDSPEC
```

3.2.5. Statischer Schnittstellentest

3.2.5.1. Beschreibung

Ziel der Funktion ist es, Fehler durch Nichtübereinstimmen von Schnittstellen zu erkennen. Es wird die Schnittstelle zwischen einem Quellprogramm (UO) und den im UO aufgerufenen externen Entries untersucht. Das Ergebnis dieser Auswertefunktion ist eine Übersicht über die Daten der Schnittstelle.

Die Ausgabe beinhaltet im einzelnen folgende Information:

1. den Namen des UO
2. den Namen der Quelldatei, die das UO enthält
3. eine Liste mit Informationen über die im UO deklarierten externen Entries, bestehend aus:
   1. Name des externen Entry
   2. Name der Quelldatei, die die externe Prozedur enthält
   3. eine Liste bestehend aus Typ- und Längenspezifikation sämtlicher Parameter des externen Entry
   4. eine Liste bestehend aus Typ- und Längenspezifikation sämtlicher Parameter der Deklaration des externen Entry im aufrufenden UO
   5. Typ- und Längenspezifikation des Rückgabeparameters des externen Entries (gegebenenfalls)
   6. Typ- und Längenspezifikation des Rückgabeparameters der Deklaration des externen Entry im aufrufenden UO
4. Eine Liste, die sämtliche Namen der im UO deklarierten Daten, die mit dem Scope Attribut "external" versehen sind, zusammen mit Typ- und Längenspezifikation, sowie sämtliche Attribute, die aus der ST zu entnehmen sind, enthält
5. eine Liste, die die obige Information für die external Daten der externen Prozeduren enthält

Beispiel: A und Y seien die folgenden Prozeduren:

```
A : PROCEDURE;
    DCL M CHARACTER (15) EXTERNAL;
    DCL N BINARY FIXED (31);
    DCL (P,O) BINARY FIXED (15);
    DCL X ENTRY (CHARACTER (15),BINARY FIXED (31));
    DCL Y ENTRY (BINARY FIXED (15),BINARY FIXED (15));
    ...
    CALL X (M,N);
    CALL Y (O,P);
    ...
END A;
```



```
Y : PROCEDURE (D,E);
    DCL (D,E) BINARY FIXED (15);
    DCL F CHARACTER (16);
    DCL G BINARY FIXED (15);
    ...
X : ENTRY (F,G);
    DCL H CHARACTER (16) EXTERNAL;
    ...
END Y;
```

Ausgabe der Auswertefunktion Statische Schnittstelle für das obige Beispiel:

A	
Bib 1	
X	Y
Character 16 Binary Fixed 15	Binary Fixed 15 Binary Fixed 15
Character 15 Binary Fixed 31	Binary Fixed 15 Binary Fixed 15
Keine Funktion	Keine Funktion
Keine Funktion	Keine Funktion
M Character 15 External	
H Character 16 External	Character 16 External

3.2.5.2. Spezifikation Statische Schnittstelle

```
INSTANTIATE 3_Tupel to Name_Nat_Nat
ACTUALIZE : #x1 by Name
            #x2, #x3 by Nat
            SORTS : #x1.elem by name
                    #x2.elem, #x3.elem by nat
RENAME : SORTS : 3_tupel by name_nat_nat
           OPS : erz_drei by erz_t
END


INSTANTIATE Liste to L(Name_Nat_Nat)
ACTUALIZE : #x by Name_Nat_Nat
            SORTS : #x.elem by name_nat_nat
RENAME : SORTS : liste by l(name_nat_nat)
           OPS : nil by nil_lt
END


INSTANTIATE 3_Tupel to Ext_Dat
ACTUALIZE : #x1 by Name
            #x2 by Param_Type
            #x3 by Attributliste
            SORTS : #x1.elem by name
                    #x2.elem by param_type
                    #x3.elem by attributliste
RENAME : SORTS : 3_tupel by ext_dat
           OPS : erz_drei by erz_ed
END


INSTANTIATE Liste to L(Ext_Dat)
ACTUALIZE : #x by Ext_Dat
            SORTS : #x.elem by ext_dat
RENAME : SORTS : liste by l(ext_dat)
           OPS : nil by nil_led
END


INSTANTIATE Liste to L(L(Ext_Dat))
ACTUALIZE : #x by L(Ext_Dat)
            SORTS : #x.elem by by l(ext_dat)
```



```
RENAME : SORTS : liste to l(l(ext_dat))
           OPS : nil by nil_lled
END


INSTANTIATE 6_Tupel to Inf_Ext_Obj
ACTUALIZE : #x1, #x2 by Name
            #x3, #x4 by L(Param_Type)
            #x5, #x6 by Param_Type
            SORTS : #x1.elem, #x2.elem by name
                    #x3.elem, #x4.elem by l(param_type)
                    #x5.elem, #x6.elem by param_type
RENAME : SORTS : 6_tupel by inf_ext_obj
           OPS : erz_sechs by erz_ieo
END


INSTANTIATE Liste to L(Inf_Ext_Obj)
ACTUALIZE : #x by Inf_Ext_Obj
            SORTS : #x.elem by inf_ext_obj
RENAME : SORTS : liste by l(inf_ext_obj)
           OPS : nil by nil_lieo
END


INSTANTIATE 5_Tupel to Schnittstelle_Stat
ACTUALIZE : #x1, #x2 by Name
            #x3 by L(Inf_Ext_Obj)
            #x4 by L(Ext_Dat)
            #x5 by L(L(Ext_Dat))
            SORTS : #x1.elem, #x2.elem by name
                    #x3.elem by l(inf_ext_obj)
                    #x4.elem by l(ext_dat)
                    #x5.elem by l(l(ext_dat))
RENAME : SORTS : 5_tupel by schnittstalle_stat
           OPS : erz_fünf by erz_ss
END
```

SSPEC STATISCHER SCHNITTSTELLENTEST

USE SSPECS : Bool, Schnittstelle, Vollständigkeit der Externverweise, Benutzthierarchie, Externe Prozeduren, Interne Prozeduren, Schnittstelle_Stat, L(Name_Nat_Nat)

PUBLIC OPS :
```
op_statischer_schnittstellentest: name l(dat)-->schnittstelle_dat
lösche_doppelt_dekl_entries:l(name_nat)-->l(name_nat)
gen_l_inf_ext_obj:name l(name_nat) sb l(dat)-->l(inf_ext_obj)
op_param_attr_dcl:name sb-->l(param_type)
op_ret_attr_dcl:name sb-->l(param_type)
gen_l_external_daten:name l(name_nat) l(dat)-->l(ext_dat)
op_external_daten:sb st-->l(ext_dat)
gen_l_sts_t_baum_dcl_part:sb-->l(stammsatz)
gen_l_stmtnr_d_externals:l(stammsatz)-->l)nat)
gen_l_externals:l(nat) l(stammsatz)-->l(name_nat_nat)
ergänze_attr_liste:name_nat_nat param_type l(knoten_st)
                                                st-->ext_dat
gen_l_externals_1:nat l(stammsatz)-->l(name_nat_nat)
lösche_external:name_nat_nat l(name_nat_nat)-->l(name_nat_nat)
```

PROPERTIES :
```
/*
Die Operation op_statische_schnittstelle (m,ld) liefert die gewünschte
Information durch Aufruf der verschiedenen Hilfsoperationen.
Die Operation gen_l_inf_ext_obj  (m,lns,sb,ld)  ist ebenfalls eine Zu-
sammenfassung von Operationsaufrufen.
*/

/*
lösche_doppelt_dekl_entries (lns) bewirkt, daß in der Liste lns, die
aus den Tupeln Name-Statementnummer der Deklarationen der im UO dekla-
rierten externen Entries besteht, jeder deklarierte externe Entry nur
einmal vorkommt.
*/
∀ l(name_nat) lns: ∃ l(name_nat) l:
  l=lösche_doppelt_dekl_entries (lns)
  ∧ ∀ name_nat n: ist_enth_lni (n,l)=true
     ⇒ ¬ ∃ name_nat n': ist_enth_lni (n',l)=true
            ∧ n≠n' ∧ sni_1 (n)=sni_1 (n')
            ∧ ist_enth_lni (n,lns)=true
            ∧ ¬ ∃ name_nat nt: ist_enth_lni (nt,lns)=true
                  ∧ sni_1 (nt)≠sni_1 (n)

/*
Die Operation op_param_attr_dcl  (e,sb)  liefert die Liste der Para-
metertypen des Procedure- oder Entry-Statements mit Namen e.
*/
∀ sb sb: ∀ name e: ∃ l(param_type) lpt:
  lpt=op_param_attr_dcl (e,sb)
  ∧ (lpt≠error
    ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
          ∧ sdt_3 (ssts_2 (s))=e
          ∧ sdt_10 ssts_2 (s))=sdt_10 (ssts_2 (
                                best_äußerste_proz (sb)))
          ∧ (ssl_6 (ssts_1 (s))=proc stmt
             ∨ ssl_6 (ssts_1 (s))=secondary entry)
          ∧ lpt=gen_l_typ_param (gen_l_name_form_par (
                erz_ni (e,ssl_7 (ssts_1 (s))),sb),
                lds_sb (enth_vater_proc (s,sb),sb)))

/*
Die Operation op_return_attr_dcl  (e,sb)  liefert das Return Attribut
des Procedure- bzw. Entry-Statements mit Entrynamen e.

Beispiel:
   E : PROCEDURE (P1,P2) RETURNS (FIXED BINARY);
   F : ENTRY (P1) RETURNS (FIXED BINARY);
*/
∀ name e: ∀ sb sb: ∃ param_type pt:
  pt=op_return_attr_dcl (e,sb)
  ∧ (pt≠error
  ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
       ∧ (ssl_6 (ssts_1 (s))=proc stmt
          ∨ ssl_6 (ssts_1 (s))=secondary entry)
       ∧ sdt_3 (ssts_2 (s))=e
       ∧ sdt_10 (ssts_2 (s))=
             sdt_10 (ssts_2 (best_äußerste_proz (sb)))
       ∧ pt=op_return_attribut (erz_ni (e,ssl_7 (sts_1 (s)),sb))

/*
Die Operation gen_l_external_daten (m,lns,ld) ruft die Operation
op_external_daten mit dem zum externen Entry gehörenden SB und ST auf
und liefert somit zu jedem externen Entry, die in dem Modul deklarier-
ten external Daten.
*/

/*
Die Operation gen_l_sts_t_baum_dcl_part  (sb)  liefert eine Liste von
Stammsätzen. Jeder Stammsatz dieser Liste hat als Vorgänger einen
Stammsatz mit Satzart "declaration part".
*/
∀ sb sb: ∃ l(stammsatz) lsb:
  lsb=gen_l_sts_t_baum_dcl_part (sb)
  ∧ (ist_leer_lsb (lsb)=true
```

```
    v (V stammsatz s: ist_enth_lsb (s,lsb)=true
        ⇒ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
             ∧ ssl_6 (ssts_1 (s'))=declaration part
             ∧ ist_in_baum_sb (s,t_baum_sb (s',sb))=true))
  ∧ ¬ ∃ stammsatz s1,s2: ist_in_baum_sb (s1,sb)=true
       ∧ ist_in_baum_sb (s2,sb)=true
       ∧ ist_in_baum_sb (s1,t_baum_sb (s2,sb))=true
       ∧ ssl_6 (ssts_1 (s2))=declaration part
       ∧ ist_enth_lsb (s1,lsb)=false
       ∧ ist_enth_lsb (s2,lsb)=false

/*
Die Operation gen_l_stmtnr_d_externals (lsb) liefert eine Liste von
natürlichen Zahlen, die den Statementnummern der Stammsätze aus lsb
entsprechen, die die Satzart "external" haben. Die Operation wird mit
dem Ergebnis der Operation gen_l_sts_t_baum_dcl_part aufgerufen.
*/
V l(stammsatz) lsb: ∃ l(nat) l:   l=gen_l_stmtnr_d_externals (lsb)
  ∧ (ist_leer_lnat (l)=true
    v ∀ nat n: ist_enth_lnat (n,l)=true
     ⇒ ∃ stammsatz s:ist_enth_lsb (s,lsb)=true
        ∧ ssl_6 (ssts_1 (s))=external
        ∧ ssl_7 (ssts_1 (s))=n)
    ∧ ¬ ∃ stammatz s': ist_enth_lsb (s',lsb)=true
          ∧ ssl_6 (ssts_1 (s'))=external
          ∧ ist_enth_lnat (ssl_7 (ssts_1 (s')),lsb)=false

/*
Die Operation gen_l_externals (l,lsb) liefert eine Liste. Jedes Ele-
ment der Liste ist ein 3-Tupel bestehend aus dem Namen der als exter-
nal vereinbarten Variable, der Statementnummer des Declarestatements
und der Blocknummer des Blocks, der die Deklaration enthält.
Die Liste der Statementnummern l ist das Ergebnis der Operation
gen_l_stmtnr_d_externals. Die Liste von Stammsätzen lsb ist das Ergeb-
nis der Operation gen_l_sts_t_baum_dcl_part.
*/
V l(nat) l: V l(stammsatz)lsb: ∃ l(name_nat_nat) lt:
  lt=gen_l_externals (l,lsb)
  ∧ (V name_nat_nat t: ist_enth_lt (t,lt)=true
     ⇒ ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
          ∧ ssl_6 (ssts_1 (s))=declare identifier
          ∧ ssl_7 (ssts_1 (s))=st_2 (t)
          ∧ sdt_3 (ssts_2 (s))=st_1 (t)
          ∧ sdt_10 (ssts_2 (s))=st_3 (t)
          ∧ ist_enth_lnat (ssl_7 (ssts_1 (s)),ln)=true)
   ∧ ¬ ∃ stammsatz s': ist_enth_lsb (s',lsb)=true
          ∧ ssl_6 (ssts_1 (s'))=declare identifier
          ∧ ist_enth_lnat (ssl_7 (ssts_ 1(s')),ln)=true
```



```
          ∧ ist_enth_lt (erz_t (sdt_3 (ssts_2 (s')),
                                ssl_7 (ssts_1 (s')),
                                sdt_10 (ssts_2 (s')),lt)=false

/*
Die Operation lösche_mehrf_dekl_externals (lt) löscht in der Liste lt
alle Elemente, die in der Namenskomponente mit dem aktuellen Element
übereinstimmen.
lt ist das Ergebnis der Operation gen_l_externals.
*/
V l(name_nat_nat) lt: ∃ l(name_nat_nat) l:
  l=lösche_mehrf_dekl_externals (lt)
  ∧ V name_nat_nat t: ist_enth_lt (t,l)=true
     ⇒ ((¬ ∃ name_nat_nat t':
              ist_enth_lt (t',entferne_lt (t,l))=true
              ∧ st_1 (t)=st_1 (t'))
       ∧ ist_enth_lt (t,lt)=true
       ∧ ¬ ∃ name_nat_nat n: ist_enth_lt (n,lt)=true
           ∧ st_1 (n)≠st_1 (t))

/*
Die Operation ergänze_attr_l (t,pt,lst,st) erzeugt ein Tripel
bestehend aus dem Namen der Variablen, dem Datentyp und der Attribut-
liste aus der ST.
t kennzeichnet die aktuelle Variable. pt ist der Datentyp einschließ-
lich Längenspezifikation der Variablen. lst ist die Liste der Knoten
der ST. st ist die ST des Quellmoduls.
*/
V name_nat_nat t: V param_type pt:
  V l(knoten_st) lst: V st st:
   ∃ ext_dat ed: ed=ergänze_attr_l (t,pt,lst,st)
    ∧ (ed≠error
      ⇒ ((∃ knoten_st k: ist_in_baum_st (k,st)=true
           ∧ give_blocknr (vater_st (k,st))=st_3 (t)
           ∧ ∃ gebiet gb: k=blck_g (gb)
              ∧ ∃ eintrag etr: ist_enth_gb (etr,gb)=true
                 ∧ setr_1 (etr)=st_1 (t)
                 ∧ setr_2 (etr)=sed_3 (ed))
         ∧ sed_1 (ed)=st_1 (t)
         ∧ sed_2 (ed)=pt))

/*
Die Operation op_external (sb,st) ist eine Operationsaufruffolge, die
sämtliche Information über die im durch sb und st gegebenen Quellmodul
deklarierten Daten mit External Attribut liefert.
*/
PRIVATE OPS :
op_ret_attr_dcl_1:name l(stammsatz) sb-->param_type
```

```
op_param_attr_dcl_1:name l(stammsatz) sb-->l(param_type)
gen_l_sts_t_baum_dcl_part_1:l(stammsatz) sb-->l(stammsatz)
lösche_mehrf_dekl_externals:l(name_nat_nat) sb st
                                  -->l(name_nat_nat)
op_external_daten_1:l(name_nat_nat) sb st-->l(ext_dat)
ergänze_attr_liste_1:l(knoten_st) name_nat_nat
                            param_type-->ext_dat
ergänze_attr_liste_2:symtab l(knoten_st) name_nat_nat
                                  param_type-->ext_dat
ergänze_attr_liste_3:name gebiet param_type-->ext_dat

DEFINE OPS :
op_statischer_schnittstellentest (n,ld):=
  let x=liefere_eind_modul (n,ld) in
  let sb=lies_sb (n,x) in
  let st=lies_st (n,x) in
  let y=lösche_doppelt_dekl_entries (
             gen_l_dcl_ext_entries (sb)) in
    erz_ss (n,
            name_quelldatei (x),
            gen_l_inf_ext_obj (n,y,sb,ld),
            op_external_daten (sb,st),
            gen_l_external_daten (n,y,ld));

lösche_doppel_dekl_entries (lns):=
   case lns is
   *nil_lni-->nil_lni
   *cons_lni (n,rest)-->
       cons_lni (n, lösche_doppelt_dekl_entries (
                    lösche_id_ext_entries (n,rest)))
   esac ;

gen_l_inf_ext_obj (n,lns,sb,ld):=
   case lns is
   *nil_lni-->nil_lieo
   *cons_lni (ns,rest)-->
    let x=liefere_eind_entry (n,sni_1 (ns),ld) in
    cons_lieo (erz_ieo (sni_1 (ns),
                        name_quelldatei (snd_2 (x)),
                        op_param_attr_dcl (sni_1 (ns),
                           lies_sb (snd_1 (x),snd_2(x))),
                        op_parameter_attribut (ns,sb),
                        op_ret_attr_dcl (sni_1 (ns),
                          lies_sb (snd_1 (x),snd_2 (x))),
                        op_return_attribut (ns,sb)),
              gen_l_inf_ext_obj (n,rest,sb,ld))
   esac ;
```



```
op_ret_attr_dcl (n,sb):=
   op_ret_attr_dcl_1 (n,preorder_sb (sb),sb);

op_ret_attr_dcl_1 (n,lsb,sb):=
   case lsb is
   *nil_lsb-->error.param_type
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_2 (s)),proc stmt)
          or eq.charstring (ssl_6 (ssts_2 (s)),secondary entry)
        and eq.name (sdt_3 (ssts_2 (s)),n)
        and eq.nat (sdt_10 (ssts_2 (s)),
                    sdt_10 (ssts_2 (best_äußerste_proz (sb))))
        then op_return_attribut (erz_ni (n,ssl_7 (ssts_1(s))),sb)
        else op_ret_attr_dcl_1 (n,rest,sb)
      fi
   esac ;

op_param_attr_dcl (n,sb):=
   op_param_attr_dcl_1 (n,preorder_sb (sb),sb);

op_param_attr_dcl_1 (n,lsb,sb):=
   case lsb is
   *nil_lsb-->error.l(param_type)
   *cons_lsb (s,rest)-->
      if (eq.charstring (ssl_6 (ssts_2 (s)),proc stmt)
          or eq.charstring (ssl_6 (ssts_2 (s)),secondary entry))
        and eq.name (sdt_3 (ssts_2 (s)),n)
        and eq.nat (sdt_10 (ssts_2)),
                    sdt_10 (ssts_2 (best_äußerste_proz (sb)))
        then gen_l_typ_param (
             gen_l_name_form_par (
               erz_ni (n,ssl_7 (ssts_1 (s))),sb),
               lds_sb (enth_vater_proz (s,sb),sb),sb)
        else op_param_attr_dcl_1 (n,rest,sb)
      fi
   esac ;

gen_l_external_daten (n,lns,ld):=
   case lns is
   *nil_lni-->nil_lled
   *cons_lni (ns,rest)-->
    let x=liefere_eind_entry (n,sni_1 (ns),ld) in
    cons_lled (op_external_daten (lies_sb (snd_1 (x),snd_2 (x)),
                                  lies_st (snd_1 (x),snd_2 (x))),
               gen_l_external_daten (n,rest,ld))
   esac ;

gen_l_sts_t_baum_dcl_part (sb):=
```

```
   gen_l_sts_t_baum_dcl_part_1 (preorder_sb (sb),sb);

gen_l_sts_t_baum_dcl_part_1 (lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lsb
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_2 (s)),declaration part)
        then append_lsb (preorder_sb (t_baum_sb (s,sb)),
                         gen_l_sts_t_baum_dcl_part_1 (rest,sb))
        else gen_l_sts_t_baum_dcl_part_1 (rest,sb)
      fi
   esac ;

gen_l_stmtnr_d_externals (lsb):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_2 (s)),external)
        then cons_lnat (ssl_7 (ssts_1 (s)),
                        gen_l_stmtnr_d_externals (rest))
        else gen_l_stmtnr_d_externals (rest)
      fi
   esac ;

gen_l_externals (l,lsb):=
   case l is
   *nil_lnat-->nil_lt
   *cons_lnat (n,rest)-->
     append_lnat (gen_l_externals_1 (n,lsb),
                  gen_l_externals (rest,sb))
   esac ;

gen_l_externals_1 (n,lsb):=
   case lsb is
   *nil_lsb-->nil_lt
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_2 (s)),declare identifier)
         and eq.nat (ssl_7 (ssts_1 (s)),n)
        then cons_lt (erz_t (sdt_3 (ssts_2 (s)),n,
                             sdt_10 (ssts_2 (s))),
                      gen_l_externals_1 (n,rest))
        else gen_l_externals_1 (n,rest)
      fi
   esac ;

lösche_mehrf_dekl_externals (lt):=
   case lt is
   *nil_lt-->nil_lt
```



```
   *cons_lt (t,rest)-->
      cons_lt (t,lösche_mehrf_dekl_externals (
                      lösche_external (t,rest))
   esac ;

lösche_external (t,lt):=
   case lt is
   *nil_lt-->nil_lt
   *cons_lt (t1,rest)-->
      if eq.name (st_1 (t1),st_1 (t))
        then lösche_external (t1,rest)
        else cons_lt (t1,lösche_external (t,rest))
      fi
   esac ;

op_external_daten_1 (lt,sb,st):=
   case lt is
   *nil_lt-->nil_led
   *cons_led (t,rest)-->
     cons_led (ergänze_attr_liste (t,
                 gen_typ_spez (gen_l_stmtnr (st_2 (t),sb),sb),
                 preorder_st (st),st),
               op_external-daten_1 (rest,sb,st))
   esac ;

ergänze_attr_liste (t,pt,lst,st):=
   case lst is
   *nil_lst-->error.extdat
   *cons_lst (k,rest)-->
      if (ist_proz_block (k) or ist_begin_block (k))
         and eq.nat (give_blocknr (k),st_3 (t))
        then ergänze_attr_liste_1 (lds_st (s,st),t,pt)
        else ergänze_attr_liste_1 (t,pt,rest,st)
      fi
   esac ;

ergänze_attr_liste_1 (lst,t,pt):=
   case lst is
   *nil_lst-->error.extdat
   *cons_lst (k,rest)-->
     ergänze_attr_liste_2 (skst_1 (k),rest,t,pt)
   esac ;

ergänze_attr_liste_2 (s,lst,t,pt):=
   case s is
   *blck_b(b)-->ergänze_attr_liste_1 (lst,t,pt)
   *blck_g (gb)--> if ist_dekl_in (st_1 (t),gb)
                   then ergänze_attr_liste_3 (st_1 (t),gb,pt)
```

```
                    else ergänze_attr_liste_1 (lst,t,pt)
                  fi
   esac ;

ergänze_attr_liste_3 (n,gb,pt):=
   case gb is
   *nil_gb-->error.extdat
   *cons_gb (e,rest)--> if eq.name (setr_1 (e),n)
                        then erz_ed (n,pt,setr_2 (e))
                        else ergänze_attr_liste_3 (n,rest,pt)
                       fi
   esac ;

op_external_daten (sb,st):=
   let x=gen_l_sts_t_baum_dcl_part (sb) in
   op_external_daten_1 (lösche_mehrf_dekl_externals (
     gen_l_externals (gen_l_stmtnr_d_externals (x),x),sb,st);

ENDSPEC
```


3.2.6. Verweisstruktur

3.2.6.1. Beschreibung

Zweck der Funktion ist es, die intermodularen Beziehungen, der sich gegenseitig aufrufenden Quellmodule zu durchleuchten. Diese Funktion ist eine Veränderung und eine Erweiterung der Auswertefunktion Benutzthierarchie. Betrachtungsobjekte sind: Quellmodule, externe Sec. Entries, interne Sec. Entries ung interne Prozeduren. Die Ausgabe besteht wiederum in einer Baumstruktur, dessen Knoten Darstellungen der Betrachtungsobjekte sind und dessen Kanten von oben nach unten als "ruft auf" zu lesen sind. In Abänderung der Benutzthierarchie erscheint jeder Aufruf eines Betrachtungsobjekts in der Baumstruktur.

Die Hierarchie endet in einem Knoten, falls das diesem Knoten entsprechende Objekt sich selbst aufruft (auch über mehrere Objekte hinweg) oder falls für das Objekt bereits die Hierarchie ermittel worden ist. Analog zur Enthaltenstruktur ist es möglich, sich durch Angabe eines Fensters eine Teilhierarchie erstellen zu lassen.

Für Sec. Entries gilt, daß sie die gleiche Hierarchie enthalten, wie das Betrachtungsobjekt, in dem sie enthalten sind.

In Erweiterung der Benutzthierarchie erhält jeder Knoten die folgende Information:

1. den Namen des Moduls
2. sämtliche Entrynamen
3. eine Kennzeichnung des Betrachtungsobjekts
4. den Quelldateinamen
5. die Statementnummer des Entry-Stmts bzw. des Procedure-Stmts
6. eine Liste der aktuellen Parameter mit denen das Betrachungsobjekt aufgerufen wird
7. eine Liste mit den Namen der formalen Parameter und den Orten des Auftretens im Betrachtungsobjekt
8. eine Liste sämtlicher möglicher Return-Werte bei Funktionen
9. eine Liste mit mit den Namen und den Orten des Auftretens sämtlicher als external vereinbarter Daten
10. eine Liste mit den Namen und den Orten des Auftretens der in der Sprache SPL vordefinierten Pointer-Variablen GLOBAL 0, GLOBAL 1, GLOBAL 2 und GLOBAL 3
11. eine natürliche Zahl zur eindeutigen Identifizierung eines Betrachtungsobjekts in der Verweisstruktur

3.2.6.2. Spezifikation Verweisstruktur

```
INSTANTIATE Liste to L(Charstring)
ACTUALIZE : #x by Charstring
            SORTS : #x.elem by charstring
RENAME : SORTS : liste by l(charstring)
           OPS : nil by nil_lstr
END

INSTANTIATE Tupel to Name_L(Nat)
ACTUALIZE : #x1 by Name
            #x2 by L(Nat)
            SORTS : #x1.elem by name
                    #x2.elem by l(nat)
RENAME : SORTS : tupel by name_l(nat)
           OPS : erz_tup by erz_nl
END

INSTANTIATE Liste to L(Name_L(Nat))
ACTUALIZE : #x by Name_L(Nat)
            SORTS : #x.elem by name_l(nat)
RENAME : SORTS : liste by l(name_l(nat))
           OPS : nil by nil_lnl
END

INSTANTIATE 11_Tupel to Knoten_vs
ACTUALIZE : #x1 by Name
            #x2 by l(Name)
            #x3 by Charstring
            #x4 by Name
            #x5 by Nat
            #x6 by L(Charstring)
            #x7 by L(Name_L(Nat))
            #x8 by L(Charstring)
            #x9, #x10 by L(Name_L(Nat))
            #x11 by Nat
            SORTS : #x1.elem by name
                    #x2.elem by l(name)
                    #x3.elem by charstring
                    #x4.elem by name
                    #x5.elem by nat
                    #x6.elem by l(charstring)
                    #x7.elem by l(name_l(nat))
```



```
                    #x8.elem by l(charstring)
                    #x9.elem, #x10.elem by l(name_l(nat))
                    #x11.elem by nat
RENAME : SORTS : 11_tupel by knoten_vs
           OPS : erz_elf by erz_kvs
END

INSTANTIATE Baum to Verweisstruktur_1
ACTUALIZE : #x by knoten_vs
            SORTS : #x.elem by knoten_vs
RENAME : SORTS : baum by verweisstruktur
           OPS : erz_baum by erz_vs
END

INSTANTIATE 3_Tupel to Erweitertes_Callstmt
ACTUALIZE : #x1 by Stammsatz
            #x2 by L(Charstring)
            #x3 by Name
            SORTS : #x1.elem by stammsatz
                    #x2.elem by l(charstring)
                    #x3.elem by name
RENAME : SORTS : 3_tupel by erw_callstmt
           OPS : erz_drei by erz_ecs
END

INSTANTIATE Liste to L(Erweitertes_Callstmt)
ACTUALIZE : #x by Erweitertes_Callstmt
            SORTS : #x.elem by erw_callstmt
RENAME : SORTS : liste by l(erw_callstmt)
           OPS : nil by nil_lecs
END
```

```
SSPEC VERWEISSTRUKTUR

USE SSPECS : Bool, Schnittstelle, L(Erweitertes_Callstmt), Verweis-
             struktur_1, Benutzthierarchie, Fenster, Externe Prozedu-
             ren, Statischer Schnittstellentest

PUBLIC OPS :
op_verweisstruktur:name fenster l(dat)-->verweisstruktur
```

PROPERTIES :

```
/*
   Die nach außen angebotene Operation op_verweisstruktur (m,f,ld)
entspricht in ihrem Aufbau der Operation op_benutzthierarchie (m,f,ld)
der SSPEC Benutzthierarchie.
   Abhängig vom angegebenem Fenster f wird die Operation anf_vs_hier
mit unterschiedlichen aktuellen Parametern aufgerufen, so daß sicher-
gestellt ist, daß die Verweisstruktur ab dem angegebenem Betrach-
tungsobjekt erstellt wird.
*/

/*
lac_vs (sb,dn) liefert die Liste der erweiterten Call-Statements. Ein
solches erweitertes Call-Statement besteht aus dem Stammsatz, der den
Aufruf des Entry enthält, den aktuellen Parametern mit denen der Entry
aufgerufen wird und dem Dateinamen der Datei, die den durch sb gegebe-
nen Quellmodul enthält.
   Die Aufrufe von Entries sind durch Stammsätze mit Satzart "call",
"callfct", "callext" und "callextfct" im SB festgelegt. Es ist darauf
zu achten, daß nur Aufrufe der aktuellen Prozedur und nicht etwa auch
Aufrufe in enthaltenen Prozeduren erfaßt werden. Dies wird berücksich-
tigt, indem kontrolliert wird, ob der Aufruf in der jeweils äußersten
Prozedur enthalten ist.
*/
V sb sb: V name dn:
  ∃ l(erw_callstmt) l: l=lac_vs (sb,dn)
  ∧ V erw_callstmt e:ist_enth_lecs (e,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
         ∧ (ssl_6 (ssts_1 (s))=call
           v ssl_6 (ssts_1 (s))=callfct
           v ssl_6 (ssts_1 (s))=callext
           v ssl_6 (ssts_1 (s))=callextfct)
         ∧ enth_vater_proc (s,sb)=best_äußerste_proz (sb)
         ∧ e=erz_ecs (s,op_akt_parameter (
                gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb),sb),dn)

/*
Die Operation lap_vs (l,ld) liefert für jedes erweiterte Call-
Statement der Liste l das zugehörige Objekt in Form eines Knotens der
Verweisstruktur.
Auch diese Operation entspricht in analoger Weise der Operation lap
der SSPEC der Benutzthierarchie.
*/

/*
Die Operation op_akt_parameter (l,sb) liefert die aktuellen Parameter
des Aufrufs.
l ist die Liste der Stammsätze aus sb mit der Statementnummer des
```



```
Call-Statements.
Sofern die Prozedur Parameter hat, existiert ein Stammsatz mit Satzart
"argument" für jeden aktuellen Parameter. Der gesamte Expression, der
das Argument bildet, ist dann der aktuelle Parameter.
*/
V l(stammsatz) lsb: V sb sb:
  ∃ l(charstring) lc: lc=op_akt_parameter (lsb,sb)
  ∧ (ist_leer_lsb (lsb)=true
    v ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
       ∧ ssl_6 (ssts_1 (s))=argument
       ∧ ∃ charstring c: ist_enth_lstr (c,lc)=true
       ∧ c=op_concat_inhalt (blätter_sb (t_baum_sb (s,sb))))

/*
Die Operation op_concat_inhalt (lsb) konkateniert sämtliche Felder In-
halt der Stammsätze aus lsb, nachdem sie vom Typ Name in den Typ
Charstring geändert wurden. Dies ist Voraussetzung, damit ein Ex-
pression als Charstring dargestellt werden kann.
*/

/*
Die Operation op_form_parameter (s,sb) liefert zu jedem formalen Para-
meter des durch s gegebenen Procedure- bzw. Entry-Statements den Ort
des Auftretens in Form von Statementnummern.
*/
V stammsatz s: V sb sb:
  ∃ l(name_l(nat)) l: l=op_form_parameter (s,sb)
  ∧ V name_l(nat)n: ist_enth_lnl (n,l)=true
     ⇒ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
         ∧ ssl_7 (ssts_1 (s'))=ssl_7 (ssts_1 (s))
         ∧ ssl_6 (ssts_1 (s'))=variable identifier
         ∧ n=erz_nl (sdt_3 (ssts_2 (s')),
              ort_d_auftretens_vs (sdt_3 (ssts_2 (s')),sb,
                                   sdt_10 (ssts_2 (s'))

/*
Die Operation op_rückgabe_parameter (sts,sb) liefert sämtliche Return-
Ausdrücke der Prozedur, die durch das Entry- bzw. Procedure-Statement
(sts) gegeben ist. Jeder Return ist durch ein Return-Statement im SB
festgelegt. Der Return-Value des Return-Statements wird durch ein Ex-
pression definiert. Dieser gesamte Ausdruck stellt den Returnwert dar,
der auszugeben ist.
*/
V stammsatz sts: V sb sb:
  ∃ l(charstring) l: l=op_rückgabe_parameter (sts,sb)
  ∧ V charstring c: ist_enth_lstr (c,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
         ∧ ssl_6 (ssts_1 (s))=return stmt
```

```
        ∧ sdt_10 (ssts_2 (sts))=sdt_10 (ssts_2 (
                                  enth_vater_proc (s,sb)))
        ∧ c=op_rückgabe_parameter_2 (
              gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb),sb)

∀ l(stammsatz) lsb: ∀ sb sb:
  ∃ charstring c: c=op_rückgabe_parameter_2 (lsb,sb)
  ∧ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
     ∧ ssl_6 (ssts_1 (s))=return value
     ∧ c=op_concat_inhalt (blätter_sb (t_baum_sb (s,sb)))

/*
Die Operation op_globals (sb) liefert zu den durch die Sprache defi-
nierten Pointer-Variablen GLOBAL0, GLOBAL1, GLOBAL2 und GLOBAL3 den
Ort des Auftretens. Das Auftreten einer solchen Pointer Variablen ist
durch einen Stammsatz mit Satzart "GLOBAL0", "GLOBAL1", "GLOBAL2" oder
"GLOBAL3" im SB festgelegt. Die Hilfsoperation op_global ruft für jede
Pointer Variable die Hilfsfunktion op_globals_1 auf.
*/
∀ l(stammsatz) lsb: ∀ name n: ∃ l(nat) l:
  l=op_globals_1 (lsb,n)
  ∀ nat n1: ist_enth_lnat(n1,l)=true
  ⇒ ∃ stammsatz s: ist_enth_lsb (s,lsb)=true
      ∧ ssl_6 (ssts_1 (s))=n
      ∧ ssl_7 (ssts_1 (s))=n1
      ∧ ¬ ∃ stammsatz s': ist_enth_lsb (s',lsb)=true
            ∧ ssl_6 (ssts_1 (s'))=n
            ∧ ist_enth_lnat (ssl_7 (ssts_1 (s'))l)=false

/*
Die Operation ort_d_auftretens (n,sb,nat) liefert sämtliche Stmtnum-
mern der Statements, in denen der Name n vorkommt. nat gibt dabei die
Blocknummer des Blocks an, in dem der Name n deklariert ist. Zu
berücksichtigen ist wiederum die Umgebung in der ein Name bekannt ist.
   Durch die Operation neue_def werden zwei Fälle unterschieden: Ein
Name kann durch ein Declare-Statement definiert werden oder durch ein
Label eines Statements ist.
   Die Operation gen_l_blocknr_neue_def liefert die Blocknummern der
Blöcke, in denen der aktuelle Name durch ein Declare-Statement umdefi-
niert wurde.
   Die Operation best_ort_def_label_enty liefert die Blocknummern der
Blöcke, in denen der aktuelle Name als Label bzw. als Entry-Konstante
umdefiniert wurde.
   Die Operation neue_def liefert alle Blocknummern der Blöcke, in de-
nen nicht nach einem Auftreten des Namens n gesucht werden darf.
*/
∀ name n: ∀ sb sb: ∀ nat i:
  ∃ l(nat)l: l=ort_d_auftretens (n,sb,i)
```



```
  ∧ ∀ nat j: ist_enth_lnat (j,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
         ∧ ssl_7 (ssts_1 (s))=j
         ∧ sdt3_ (ssts_2 (s))=n
         ∧ ist_enth_lnat (j,neue_def (n,preorder_sb (sb),sb),i))=false

∀ name n: ∀ l(stammsatz) lsb:
  ∀ sb sb: ∀ nat i: ∃ l(nat) l:
  l=gen_l_blöcke_neue_def (n,lsb,sb,i)
  ∧ ∀ nat j: ist_enth_lnat (j,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
         ∧ ssl_6 (ssts_1 (s))=declare identifier
         ∧ sdt_3 (ssts_2 (s))=n
         ∧ ∃ stammsatz s': ist_in_baum_sb (s',sb)=true
            ∧ ssl_6 (ssts_1 (s'))=declaration part
            ∧ sdt_10 (ssts_2 (s'))≠i
            ∧ ist_in_baum_sb (s,t_baum_sb (s',sb))=true
            ∧ sdt_10 (ssts_2 (s))=j

∀ name n: ∀ l(stammsatz) lsb:
  ∀ sb sb: ∀ nat i: ∃ l(nat) l:
  l=best_ort_def_label_entry_vs (n,lsb,sb,i)
  ∧ ∀ nat j: ist_enth_lnat (j,l)=true
     ⇒ ∃ stammsatz s: ist_in_baum_sb (s,sb)=true
         ∧ ssl_6 (ssts_1 (s))=label identifier
         ∧ sdt_3 (ssts_2 (s))=n
         ∧ sdt_10 (ssts_2 (s))=j
         ∧ (ssl_6 (ssts_1 (vater_sb (s,sb)))=proc stmt
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=entry stmt
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=ending
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=if clause
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=beg stmt
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=do stmt
          ∨ ssl_6 (ssts_1 (vater_sb (s,sb)))=non group stmt)
         ∧ sdt_10 (ssts_2 (vater_sb (s,sb)))≠i

PRIVATE OPS :
anf_vs_hier:l(knoten_vs) verweisstruktur knoten_vs l(dat)
                                              -->verweisstruktur
anfügen_vs:l(knoten_vs) verweisstruktur knoten_vs l(dat)
                                              -->verweisstruktur
lap_vs:l(erw_callstmt) l(dat)-->l(knoten_vs)
lac_vs:sb name-->l(erw_callstmt)
ermittle_uo_vs:erw_callstmt l(dat)-->knoten_vs
suche_obj_zu_call_vs:stammsatz l(charstring) name l(dat)
                                                  -->knoten_vs
suche_ext_obj_zu_call_vs:erw_callstmt l(dat)-->knoten_vs
gen_knoten_vs:erw_callstmt name_dat-->knoten_vs
```

```
suche_obj_vs_1:stammsatz l(charstring) name l(dat)-->knoten_vs
suche_obj_vs_2:stammsatz stammsatz sb name
                                    l(charstring)-->knoten_vs
entry_zum_call_vs:stammsatz stammsatz sb name
                                    l(charstring)-->knoten_vs
entry_zum_call_vs_1:stammsatz l(stammsatz) sb name
                                   l(charstring)-->knoten_vs
ermittle_calls_vs:knoten_vs l(dat)-->l(erw_callstmt)
v_qual_id:knoten_vs sb-->stammsatz
v_qual_id_vs_1:knoten_vs l(stammsatz)-->stammsatz
ist_bearbeitet_vs:knoten_vs verweisstruktur-->bool
anzahl_knoten_vs:knoten_vs verweisstruktur-->nat
anzahl_knoten_vs_1:knoten_vs verweisstruktur nat-->nat
ist_eq_t_qual_vs:knoten_vs knoten_vs-->bool
index_vs:knoten_vs verweisstruktur-->knoten_vs
high_index_vs:verweisstruktur-->nat
op_akt_parameter:l(stammsatz) sb-->l(charstring)
op_concat_inhalt:l(stammsatz)-->l(charstring)
neue_def:name l(stammsatz) sb nat-->l(nat)
gen_l_blocknr_neue_def:name l(stammsatz) sb nat-->l(nat)
gen_l_neue_def_1:name l(stammsatz)-->l(nat)
best_ort_def_label_entry_vs:name l(stammsatz) sb nat-->l(nat)
ist_ort_def_label_entry:name l(stammsatz)-->bool
best_ort_def_label_entry_vs_1:name l(stammsatz)-->nat
op_rückgabe_parameter:stammsatz sb-->l(charstring)
op_rückg_parameter_1:stammsatz l(stammsatz) sb-->l(charstring)
op_rückg_parameter_2:l(stammsatz) sb-->l(charstring)
ort_d_auftretens_vs:name sb nat-->l(nat)
ort_d_auftr_vs_1:name l(stammsatz) l(nat)-->l(nat)
op_form_parameter:stammsatz sb-->l(name_l(nat))
op_form_parameter_1:l(stammsatz) sb-->l(name_l(nat))
op_externals:sb-->l(name_l(nat))
auftreten_externals:l(name_nat_nat)) sb-->l(name_l(nat))
auftreten_mehrf_dekl_externals:name_nat_nat
                                  l(name_nat_nat) sb-->l(nat)
op_globals:sb-->l(name_l(nat))
op_globals_1:l(stammsatz) name-->l(nat)

DEFINE OPS :
op_verweisstruktur (m,f,ld):=
  let sb=lies_sb (m,liefere_eind_modul (m,ld)) in
   case f is
   *alles-->
    let x=erz_kvs (m,
                l_aller_entrynamen (m,liefere_eind_modul (,ld)),
                Modul,
                name_quelldatei (liefere_eind_modul (m,ld)),
                nil_lstr,
```



```
                0,
                op_formale_parameter (best_äußerste_proz (sb),sb),
                op_rückgabe_parameter (best_äußerste_proz (sb),sb),
                op_externals (sb),
                op_globals (sb),
                0) in
    anf_vs_hier (lap_vs (lac_vs (sb, name_quelldatei (
          liefere_eind_modul (m,ld))),ld),erz_vs (x),x,ld)
   *name_int_proz (p)-->
    let y1=vq_sa (m,procedure,sb) in
    let y2=t_baum_sb (y1,sb) in
    let y3=erz_kvs (m,
                l_aller_entry_points(y1,sb),
                Int. Prozedur,
                name_quelldatei (liefere_eind_modul (m,ld)),
                nil_lstr,
                ssl_7 (ssts_1 (y1)),
                op_formale_parameter (y1,y2),
                op_rückgabe_parameter (y1,y2),
                op_externals (y2),
                op_globals (y2),
                0) in
    anf_vs_hier (lap_vs (lac_vs (y2,name_quelldatei (
        liefere_eind_modul (m,ld))),ld),erz_vs (y3),y3,ld)
   *name_sec_entry (e)-->
    let z1=vq_sa (e,secondary entry,sb) in
    let z2=t_baum_sb (enth_vater_proc (z1,sb)) in
    let z3=erz_kvs (m,
                l_aller_entry_points (z1,sb)),
                Secondary Entry,
                name_quelldatei (liefere_eind_modul (m,ld)),
                nil_lstr,
                ssl_7 (ssts_1 (z1)),
                op_formale_parameter (z1,z2),
                op_rückgabe_parameter (z1,,z2),
                op_externals (z2),
                op_globals (z2),
                0) in
    anf_vs_hier (lap_vs (lac_vs (z2,name_quelldatei (
      liefere_eind_modul (m,ld))),ld),erz_vs (z3),z3,ld)
    esac ;

lac_vs (sb,dn):=lac_vs_1 (preorder_sb (sb),dn);

lac_vs_1 (lsb,sb,dn):=
   case lsb is
   *nil_lsb-->nil_lecs
   *cons_lsb (s,rest)-->
```

```
      if (eq.charstring (ssl_6 (ssts_1(s)),call)
          or eq.charstring (ssl_6 (ssts_1(s)),callfct)
          or eq.charstring (ssl_6 (ssts_1(s)),callext)
          or eq.charstring (ssl_6 (ssts_1(s)),callextfct))
         and eq.stammsatz (enth_vater_proc (s,sb)),
                           best_äußerste_proz (sb))
        then cons_lecs (erz_ecs (s,op_akt_parameter (
                        gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb),dn),
                       lac_vs_1 (rest,sb,dn))
        else lac_vs_1 (rest,sb,dn)
      fi
   esac ;

lap_vs (l,ld):=
   case l is
   *nil_lecs-->nil_lkvs
   *cons_lecs (ecs,rest)-->
     cons_lkvs (ermittle_uo_vs (ecs,ld),lap_vs (rest,ld))
   esac ;

ermittle_uo_vs (ecs,ld):=
   if eq.charstring (ssl_6 (ssts_1(s)),call)
      or eq.charstring (ssl_6 (ssts_1(s)),callfct)
     then suche_obj_zu_call_vs (secs_1 (ecs),secs_2 (ecs),
                                secs_3 (ecs),ld)
     else if eq.charstring (ssl_6 (ssts_1(s)),callext)
             or eq.charstring (ssl_6 (ssts_1s)),callextfct)
            then suche_ext_obj_zu_call_vs (ecs,ld)
            else error.knoten_vs
          fi
   fi ;

suche_ext_obj_zu_call_vs (ecs,ld):=
  gen_knoten_vs (ecs,liefere_eind_entry (
                       ssl_1 (ssts_1 (secs_1 (ecs))),
                       sdt_3 (ssts_2 (secs_1 (ecs))),ld));

gen_knoten_vs (ecs,nd):=
   let sb=lies_sb (snd_1 (nd),snd_2 (nd)) in
   let x=l_aller_entrynamen (snd_1 (nd),snd_2 (nd)) in
   let y=l_aller_sec.entry_namen (snd_1 (nd),
         sdt_3 (ssts_2 (secs_1 (ecs))),snd_2 (nd)) in
     if ist_enth_ln (sdt_3 (ssts_2 (secs_1 (ecs))),x)
       then erz_kvs (snd_1 (nd),
                     x,
                     Ext. Prozedur,
                     name_quelldatei (snd_2 (nd)),
                     0,
```



```
                     op_formale_parameter (
                       best_äußerste_proz (sb),sb),
                     op_rückgabe_parameter (
                       best_äußerste_proz (sb),sb),
                     op_externals (sb),
                     op_globals (sb),
                     0)
     else if ist_enth_ln (sdt_3 (ssts_2 (ecs_1 (ecs))),y)
           then erz_kvs (snd_1 (nd),
                         y,
                         Ext. Secondary Entry,
                         name_quelldatei (snd_2 (nd)),
                         secs_2 (ecs ),
                         0,
                         op_formale_parameter (
                           liefere_sts_des_entry_stmts (
                             sdt_3 (ssts_2 (secs_1 (ecs))),sb))
                         op_rückgabe_parameter (
                           liefere_sts_des_entry_stmts (
                             sdt_3 (ssts_2 (secs_1 (ecs))),sb))
                         op_externals (sb),
                         op_globals (sb),
                         0)
           else error.knoten_vs
         fi
   fi ;

suche_obj_zu_call_vs (sts,lstr,dn,ld):=
   let sb=lies_sb (ssl_1 (ssts_1 (sts)),lies_dat (dn,ld)) in
   let x=enth_vater_proc_beg (sts,sb) in
        suche_obj_vs_1 (sts,x,sb,dn,lstr);

suche_obj_vs_1 (sc,sv,sb,dn,lstr):=
   if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
      and ist_enth_ln (sdt_3 (ssts_2 (sc)),
                       gen_l_entry_points (sv,sb))
     then erz_kvs (ssl_1 (ssts_1 (sc)),
                   gen_l_entry_points (sv,sb);
                   Int. Prozedur,
                   dn,
                   lstr,
                   ssl_7 (ssts_1 (s)),
                   op_formale_parameter (sv,sb),
                   op_rückgabe_parameter (sv,sb),
                   op_globals (t_baum_sb (sv,sb)),
                   0)
     else suche_obj_vs_2 (lds_sb (sv,sb),sc,sv,sb,dn,lstr)
   fi ;
```

```
suche_obj_vs_2 (lsb,sc,sv,sb,dn,lstr):=
   case lsb is
   *nil_lsb-->
      if not eq.stammsatz (sv,best_äußerste_proz (sb))
        then suche_obj_vs_1 (sc,
             enth_vater_proc_beg (sv,sb),sb,dn,lstr)
        else error.knoten_vs
      fi
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),begin block)
        then suche_obj_vs_2 (rest,sc,sv,sb,dn,lstr)
       else if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
              then if ist_enth_ln (sdt_3 (ssts_2 (sc)),
                        gen_l_entry_points(s,sb))
                   then
                    erz_kvs (ssl_1 (ssts_1 (sc),
                             gen_l_entry_points (s,sb),
                             Int. Prozedur,
                             dn,
                             lstr,
                             ssl_7 (ssts_1 (s)),
                             op_formale_parameter (s,sb),
                             op_rückgabe_parameter (s,sb),
                             op_externals (t_baum_sb (s,sb)),
                             op_globals (t_baum_sb (s,sb)),
                             0)
                   else if ist_call_eines_sec.entry (s,sc,sb)
                          then entry_zum_call_vs (s,sc,sb,dn,lstr)
                          else suche_obj_vs_2(rest,sc,sv,sb,dn,lstr)
                        fi
             else suche_obj_vs_2 (
                   append_lsb (rest,lds_sb (s,sb)),
                              sc,sv,sb,dn,lstr)
           fi
   esac ;

entry_zum_call_vs (s,sc,sb,dn,lstr):=
   entry_zum_call_vs_1 (sc,lds_sb (sb,dn,lstr);

entry_zum_call_vs_1 (sc,lsb,sb,dn,lstr):=
   case lsb is
   *nil_lsb-->error.knoten_vs
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),procedure)
        or eq.charstring (ssl_6 (ssts_1 (s)),executable stmt)
       then entry_zum_call_vs_1 (sc,rest,dn,lstr)
       else if eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
              then if ist_enth_ln (sdt_3 (ssts_2 (sc)),
```



```
                        gen_l_entry_points (s,sb))
                   then erz_kvs (ssl_1 (ssts_2 (sc)),
                                 gen_l_entry_points (s,sb),
                                 Secondary Entry,
                                 dn,
                                 lstr,
                                 ssl_7 (ssts_1 (s)),
                                 op_formale_parameter (s,sb),
                                 op_rükgabe_parameter (s,sb),
                                 op_externals (
                                   enth_vater_proz (s,sb)),
                                 op_globals (
                                   enth_vater_proz (s,sb)),
                                 0)
                   else entry_zum_call_vs_1 (sc,rest,dn,lstr)
                 fi
            else entry_zum_call_vs_1 (sc,
              append_lsb (rest,lds_sb (s,sb)),sb,dn,lstr)
          fi
     fi
   esac ;

anf_vs_hier (l,vs,kvs,ld):=
   if ist_bearbeitet_vs (kvs,,vs)
     then vs
     else anfügen_vs (l,vs,kvs,ld)
   fi ;

anfügen_vs (l,vs,kvs,ld):=
   case l is
   *nil_lkvs-->vs
   *cons_lkvs (k,rest)-->
     anfügen_vs (rest,
                 anf_vs_hier (lap_vs (ermittle_calls_vs (k,ld),ld),
                              anf_vs (index_vs (k,vs)kvs,vs),
                              index_vs (k,vs),
                              ld),
                 kvs,
                 ld)
   esac ;

ermittle_calls_vs (kvs,ld):=
   let sb=lies_sb (skvs_1 (kvs),lies_dat (skvs_4 (kvs),ld)) in
   let x=v_qual_id_vs (kvs,sb) in
    if eq.charstring (skvs_3 (kvs),Modul)
       or eq.charstring (skvs_3 (kvs),Int. Prozedur)
      then lac_vs (t_baum_sb (x,sb),skvs_4 (kvs))
      else if eq.charstring (skvs_3 (kvs),Secondary Entry)
```

```
            then lac_vs (t_baum_sb (enth_vater_proz (x,sb)),
                         skvs_4 (kvs)
            else if eq.charstring (skvs_3 (kvs),Ext. Prozedur),
                    or eq.charstring (skvs_3 (kvs),Ext. Sec. Entry)
                 then lac_vs (sb,skvs_4 (kvs))
                 else error.l(erw_callstmts)
                 fi
         fi
    fi ;

v_qual_id_vs (kvs,sb):=v_qual_id_vs_1 (kvs,preorder_sb (sb));

v_qual_id_vs_1 (kvs,lsb):=
   case lsb is
   *nil_lsb-->error.stammsatz
   *cons_lsb (s,rest)-->
      if eq.nat (ssl_7 (ssts_1 (s)),skvs_5 (kvs)
         and (eq.charstring (ssl_6 (ssts_1 (s)),procedure)
              or eq.charstring (ssl_6 (ssts_1 (s)), secondary entry))
        then s
        else v_qual_id_vs_1 (kvs,rest)
      fi
   esac ;

ist_bearbeitet_vs (kvs,vs):=
   if ist_gt (anzahl_knoten_vs (kvs,vs),succ(0))
     then true
     else false
   fi ;

anzahl_knoten_vs (kvs,vs):=
   anzahl_knoten_vs_1 (kvs,vs,0);

anzahl_knoten_vs_1 (kvs,vs,n):=
   case vs is
   *erz_vs (k)-->
      if ist_eq_t_qual_vs (k,kvs)
        then incr (n)
        else n
   *son_vs (t1,t2)-->
     add (anzahl_knoten_vs_1 (kvs,t1,n),
          anzahl_knoten_vs_1 (kvs,t2,n))
   esac ;

ist_eq_t_qual_vs (k1,k2):=
   if eq.name (skvs_1 (k1),skvs_1 (k2))
      and eq.l(name) (skvs_1 (k1),skvs_2 (k2))
      and eq.charstring (skvs_3 (k1),skvs_3 (k2))
```



```
      and eq.name (skvs_4 (k1),skvs_4 (k2))
      and skvs_5 (k1),skvs_5 (k2))
     then true
     else false
   fi ;

index_vs (k,vs):=
   pkvs_11 (incr (high_index_vs (vs)),k);

high_index_vs (vs):=
   case _vs is
   *erz_vs (k)-->skvs_11 (k)
   *son_vs (t1,t2)-->
      if ist_gt (high_index_vs (t1),high_index_vs (t2))
        then high_index_vs (t1)
        else high_index_vs (t2)
      fi
   esac ;

op_akt_parameter (l,sb):=
   case lsb is
   *nil_lsb-->nil_lstr
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),argument)
        then cons_lstr (op_concat_inhalt (blätter_sb (
                                    t_baum_sb (s,sb))),
                        op_akt_parameter (rest,sb))
        else op_akt_parameter (rest,sb)
      fi
   esac ;

op_concat_inhalt (lsb):=
   case lsb is
   *nil_lsb-->blank
   *cons_lsb (s,rest)-->
     create (change_name_to_charstring (sdt_3 (ssts_2 (s))),
             op_concat_inhalt (rest))
   esac ;

op_rückgabe_parameter (sts,sb):=
   op_rückgabe_parameter_1 (sts,preorder_sb (sb),sb);

op_rückgabe_parameter_1 (sts,lsb,sb):=
   case lsb is
   *nil_lsb-->nil_lstr
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),return stmt)
         and eq.nat (sdt_10 (ssts_2 (enth_vater_proz (s,sb))),
```

```
                    sdt_10 (ssts_2 (sts)))
      then cons_lstr (op_rückgabe_parameter_2 (
                       gen_l_stmtnr (ssl_7 (ssts_1 (s)),sb),sb),
                     op_rückgabe_parameter_1 (sts,rest,sb))
      else op_rückgabe_parameter_1 (sts,rest,sb)
    fi
  esac ;

op_rückgabe_parameter_2 (lsb,sb):=
  case lsb is
  *nil_lsb-->error.charstring
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),return value)
      then op_concat_inhalt (blätter_sb (t_baum_sb (s,sb)))
      else op_rückgabe_parameter_2 (rest,sb)
  esac ;

auftreten_mehrf_dekl_externals (t,lt,sb):=
  case lt is
  *nil_lt-->nil_lnat
  *cons_lt (t1,rest)-->
    if eq.name (st_1 (t),st_1 (t1))
      then
        append_lnat (ort_d_auftretens_vs (st_1 (t1),sb,st_3 (t1)),
                     auftreten_mehrf_dekl_externals (t,rest,sb))
      else auftr_mehrf_dekl_externals (t,rest,sb)
    fi
  esac ;

auftreten_externals (lt,sb):=
  case lt is
  *nil_lt-->nil_lnl
  *cons_lt (t,rest)-->
    cons_lnl (erz_nl (st_1 (t),
                auftreten_mehrf_dekl_externals (t,lt,sb)),
                auftreten_externals (lösche_externals (t,lt,sb))
  esac ;

op_externals (sb):=
  let x=gen_l_sts_t_baum_dcl_part (sb) in
   auftreten_externals (gen_l_externals_1 (
     gen_l_stmtnr_d_externals (x),x));

gen_l_neue_def_1 (name,lsb):=
  case lsb is
  *nil_lsb-->nil_lnat
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),declare identifier)
```



```
       and eq.name (name,sdt_3 (ssts_2 (s))
      then cons_lnat (sdt_10 (ssts_2 (s)),
                      gen_l_neue_def_1 (name,rest()
      else gen_l_neue_def_1 (name,rest)
    fi
  esac ;

ort_d_auftretens_vs_1 (name,lsb,lnat):=
  case lsb is
  *nil_lsb-->nil_lnat
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),identifier)
       and eq.name (sdt_2 (ssts_2 (s)),name)
       and not ist_enth_lnat (sdt_10 (ssts_2 (s)),lnat)
      then cons_lnat (ssl_7 (ssts_1 (s)),
                      ort_d_auftretens_vs_1 (name,rest,lnat))
      else ort_d_auftretens_vs_1 (name,rest,lnat)
    fi
  esac ;

neue_def (name,lsb,sb,nat):=
  append_lnat (gen_l_blocknr_neue_def (name,lsb,sb,nat),
               best_ort_def_label_entry_vs (name,lsb,sb,nat));

op_formale_parameter (sts,sb):=
  op_form_parameter_1 (gen_l_stmtnr (ssl_7 (ssts_1 (sts)),sb),sb);

op_form_parameter_1 (lsb,sb):=
  case lsb is
  *nil_lsb-->nil_lnl
  *cons_lsb (s,rest)-->
    if eq.charstring (ssl_6 (ssts_1 (s)),variable identifier)
      then cons_lnl (erz_nl (sdt_3 (ssts_2 (s)),
                     ort_d_auftretens_vs (sdt_3 (ssts_2 (s)),
                       sb,sdt_10 (ssts_2 (s))),
                     op_form_parameter (rest,sb))
      else op_form_parameter_1 (rest,sb)
    fi
  esac ;

ort_d_auftretens_vs (name,sb,nat):=
  ort_d_auftretens_vs_1 (name,preorder_sb (sb),
                         neue_def (name,preorder_sb (sb),sb,nat));

gen_l_blocknr_neue_def (name,lsb,sb,nat):=
  case lsb is
  *nil_lsb-->nil_lnat
  *cons_lsb (s,rest)-->
```

```
      if eq.charstring (ssl_6 (ssts_1 (s)),declaration part)
         and not eq.nat (sdt_10 (ssts_2 (s)),nat)
        then gen_l_neue_def_1 (name,append_lsb (
              preorder_sb (t_baum_sb (s,sb)),
              gen_l_blocknr_neue_def (name,rest,sb,nat))
        else gen_l_blocknr_neue_def (name,rest,sb,nat)
      fi
   esac ;

best_ort_def_label_entry_vs (name,lsb,sb,nat):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
     if not eq.nat (sdt_10 (ssts_2 (s)),nat)
        and ist_ort_def_label_entry_vs_1 (name,lds_sb (s,sb))
        and (eq.charstring (ssl_6 (ssts_1 (s)),proc stmt)
             or eq.charstring (ssl_6 (ssts_1 (s)),secondary entry)
             or eq.charstring (ssl_6 (ssts_1 (s)),ending)
             or eq.charstring (ssl_6 (ssts_1 (s)),if clause)
             or eq.charstring (ssl_6 (ssts_1 (s)),begin stmt)
             or eq.charstring (ssl_6 (ssts_1 (s)),
                               non group stmt)
             or eq.charstring (ssl_6 (ssts_1 (s)),do stmt))
      then cons_lnat (best_ort_def_label_entry_vs_1 (
                      name,lds_sb (s,sb)),
                     best_ort_def_label_entry_vs (name,rest,sb,nat)
      else best_ort_def_label_entry_vs (name,rest,sb,nat)
     fi
   esac ;

ist_ort_def_label_entry_vs (n,lsb):=
   case lsb is
   *nil_lsb-->false
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
         and eq.name (sdt_3 (ssts_2 (s)),n)
        then true
        else ist_ort_def_label_entry_vs (n,rest)
      fi
   esac ;

best_ort_def_label_entry_vs_1 (n,lsb):=
   case lsb is
   *nil_lsb-->error.nat
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),label identifier)
         and eq.name (sdt_10 (ssts_2 (s)),n)
        then sdt_10 (ssts_2 (s))
```



```
        else best_ort_def_label_entry_vs_1 (n,rest)
      fi
   esac ;

op_globals (sb):=op_global (preorder_sb (sb));

op_global (lsb):=
  cons_lnl (erz_nl (global0, op_global_1 (lsb,global0)),
   cons_lnl (erz_nl (global1, op_global_1 (lsb,global1)),
    cons_lnl (erz_nl (global2, op_global_1 (lsb,global2)),
     cons_lnl (erz_nl (global3, op_global_1 (lsb,global3)),
               nil_lnat))));

op_global_1 (lsb,g):=
   case lsb is
   *nil_lsb-->nil_lnat
   *cons_lsb (s,rest)-->
      if eq.charstring (ssl_6 (ssts_1 (s)),g)
        then cons_lnat (ssl_7 (ssts_1 (s)),
                        op_global_1 (rest,x))
        else op_global_1 (rest,g)
      fi
   esac ;

ENDSPEC
```

4. Abschließende Stellungnahme

Meine persönlichen Erfahrungen mit der algebraischen Spezifikationsmethode lassen sich wie folgt zusammenfassen:

1. Es ist durchaus möglich, große Softwaresysteme algebraisch zu spezifizieren. Natürlich nehmen solche Spezifikationen einen großen Umfang an. Der Umfang dieser Spezifikation darf aber nicht darüber hinwegtäuschen, daß im Prinzip nur einfache Datenstrukturen verwendet wurden. Tatsächlich genügte es, sich auf Listen, Bäume und n-Tupel zu beschränken.
2. Man kann nicht erwarten, eine nicht selbst geschriebene auf den ersten Blick zu durchschauen. Dies gilt aber genauso für andere formale Spezifikationsmethoden, z.B. HDM-Spezifikationen in SPECIAL [Silv79]. Aber nach einer gewissen Einarbeitungszeit in das zu spezifizierende Problem und nach dem Verständnis der zugrundeliegenden Datenstrukturen und dem oftmals rekursivem Operationsschema ist eine solche algebraische Spezifikation relativ leicht zu lesen.
3. Eine Implementierung dieser Spezifikation, d.h. eine schrittweise Verfeinerung der Spezifikation bis hin zur konkreten Implementierung in eine Programmiersprache scheint aufgrund der einfachen Datentypen ohne großen Aufwand möglich zu sein. Die notwendigen Konzepte hierzu werden jedoch gerade erst entwickelt und in einer Diplomarbeit [SchT82] praktisch angewendet.
4. Die anfallende Schreiarbeit wird erheblich reduziert, wenn man auf eine Bibliothek von Spezifikationen zugreifen kann, die die gebräuchlichsten Datentypen enthält.
5. Eine Rechnerunterstützung in Form eines Editors mit Syntaxüberprüfung ist unumgänglich, um Schreibarbeit zu sparen und um einfache Fehler z.B. bei der Klammersetzung oder bei der Schreibweise von Namen zu erkennen und zu beseitigen. Außerdem gewährleistet ein Editor durch geeignete Mechanismen, daß die Spezifikationen einer Spezifikationshierarchie, insbesondere die Operationen der benutzten Spezifikationen, sinnvoll genutzt und in die neue Spezifikation mit eingebaut werden können.
6. Die Spezifikationssprache TRIPLEX liefert einen passenden Rahmen zum Schreiben von algebraischen Spezifikationen. Sämtliche durch die Sprache gegebenen Möglichkeiten werden in dieser Spezifikation nicht benötigt und erscheinen deshalb vielleicht als nicht unbedingt notwendig.
   Beispielsweise brauchte in dieser Spezifikation von INTAKT nicht zwischen Trägermenge und Herbrand Universum unterschieden werden. Dieser Fall tritt aber schon dann auf, wenn nicht nur die natürlichen Zahlen sondern auch die ganzen Zahlen (positive und negative) zu spezifizieren sind.
   Die PARM Spezifikation Elem besteht auch nur aus einer einfachen Namensgebung eines Parameters und beinhaltet sonst keine weiteren Anforderungen. Möchte man allerdings ein Array spezifizieren, in

dem nicht nur über natürliche Zahlen indiziert wird, so müßte man an den Parameter Index zumindest die Anforderung stellen, daß zwei Indizes vergleichbar sind.
7. Der Abschnitt, der die Properties enthält, ist noch konkreter festzulegen. Dies bezieht sich nicht auf die prädikatenlogische Darstellung sondern die allgemeine Syntax und die Inhalte dieses Abschnitts.

5. Verzeichnis der Spezifikationen

6.Literaturverzeichnis

[ADJ 76] Goguen, J. A.; Thatcher, J. W.; Wagner, E.G.;
An Initial Approach to the Specification, Correctness, and Implementation of Abstract Data Types, Research Report, IBM Thomas J. Watson Research Center, Yorktown Heights, New York,1976.

[BG 82a] Beierle, C.; Guntram, U.; Oberd-rster, W.; Raulefs, P.; Voss, A.;
The CTA-Approach to the Specification of Abstract Data Types, SEKI Projekt, Universität Bonn, interner Bericht (vorläufige Version), März 82.

[BG 82b] Beierle, C.; Guntram, U.; Oberdörster, W.; Raulefs, P.; Voss, A.;
Syntax of TRIPLEX (Version 1) - A Language for Systems of Algebraic Specifications, SEKI Projekt, Universität Bonn, interner Bericht (vorläufige Version), Januar 82.

[Gutt75] Guttag, J. V.;
Specification and Application to Programming of Abstrac Data Types, University of Toronto, Computer Sytsems Research Group, Technical Report CSRG-59, 1975.

[Hess81] Hesse, W.;
Methoden und Werkzeuge zur Software-Entwicklung - Ein Marsch durch die Technologie-Landschaft, Informatik-Spektrum 4, S. 229-245, 1981.

[HoRa79] Hornung, G.; Raulefs, P.;
Terminal Algebra Semantics and Retractions for Abstract Data Types, Universität Bonn, MEMO-SEKI-BN-79-6, 1979.

[Kimm79] Kimm, R.; Koch, W.; Simonsmeier, W.; Tontsch, F.;
Einführung in Software Engineering, Walter de Gruyter, Berlin,New York, 1979.

[Kreo78] Kreowsky, H.;
Algebra für Informatiker, Schriftliches Material zur gleichnamigen Lehrveranstaltung, Technische Universität Berlin, WS 1978/79.

[LiZi74] Liskov, B.; Zilles, S.;
Programming with Abstract data Types, Proc. of ACM Symp. on very High Level Languages, SIGPLAN Notices 9, pp. 50-59, 1974.

[RaSi80] Raulefs, P.; Siekmann, J.;
Programmverifikation - Darstellung des Forschungvorhabens, Universität Bonn, Universität Karlsruhe, August 1980.

[Raul79> Raulefs, P.;
Einführung in die Theorie der Datenstrukturen, Vorlesungsnotizen, Universität Bonn, SS 1979.

[SchT82] Schrupp, W.; Tamme, J.;
Spezifikation und Implementierung des Aufbereitungsteils von INTAKT, Diplomarbeit, FB Informatik, Universität Bonn, 1982.

[Silv79] Silverberg, B.; Robinson, L.; Lewitt, K.;
The HDM Handbook, Volume I-III, Stanford Research Institut, June 1979.

[Wan78] Wand, M.;
Final Algebra Semantics and Data Type Extensions, Tech. Rep. No. 65, Indiana University, Comp. Sc. Dept., Bloomington, 1978.

[Zill74] Zilles, S;
Algebraic Specification of Data Types, Project MAC Progress Report 11, MIT, Cambridge, Mass., pp. 28-52, 1974.

An internen SIEMENS Unterlagen wurde verwendet:
- Besprechungsprotokolle des INTAKT Teams, Juli, August 81,
- INTAKT - Interaktiver Arbeitsplatz zur Qualitätskontrolle von Software, Funktionskatalog, Oktober 1980,
- SPL 3 Reference Manual, Herausgegeben von: Dv WS SP313 Mch H/TQ,