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

SEKI-PROJEKT

SEKI MEMO

Ein interaktives und syntaxorientiertes Eingabesystem für algebraische Spezifikationen Band I

Hans Matheis

108/84 Mai 1984

Interner Bericht

Ein interaktives und syntaxorientiertes
Eingabesystem
für algebraische Spezifikationen
Band I

Hans Matheis

108/84 Mai 1984

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

Ein interaktives und syntaxorientiertes Eingabesystem für algebraische Spezifikationen Band I

Hans Matheis

108/84 Mai 1984

von Hans Matheis

Ein interaktives und syntaxorientiertes Eingabesystem für algebraische Spezifikationen

BAND I

Der vorliegende Bericht umfaßt Band I und Kapitel 7.
Band II ist als Memo SEKI-84-03-II zum Selbstkostenpreis erhältlich.

Zusammenfassung:
Das vorgestellte System gestattet die Eingabe von Spezifikationen in ASPIK: Es fordert die Eingabe der nächsten syntaktischen Komponente an, anlysiert sie, führt kontextsensitive Tests durch und unterstützt unmittelbare Fehlerkorrekturen. Soweit korrekte, evtl. noch unvollständige Spezifikationen können zur späteren Weiterverarbeitung gespeichert werden. Teil des Systems ist ein Kommando, das die Signatur kompletter Spezifikationshierarchien mit Instanziierungen listet, sowie ein Normalisierungsverfahren für die Erkennung aquivalenter Instanziierungsfolgen.

Abstract:
The system presented manages the input of specifications written in ASPIK: the next syntactical component is prompted for, analysed and tested for context sensitive errors. Any errors may be corrected immediately. A possibly incomplete but correct specification may be stored for future manipulations. The system offers a command to list the signature of a complete hierarchy of specifications including instantiations and a normalizing algorithm used to recognize equivalent patterns of instantiations.

BAND I

BAND II

1. Einleitung

1.1. Das interaktive Programmentwicklungs- und Verifikationssystem

SPESY, ein Entwicklungssystem für Spezifikationen, ist die zentrale Komponente eines integrierten Programmentwicklungs- und Programmverifikationssystems (IPDVS). Es entsteht aus einer Vorgängerversion ([KRST 83], [Som 84]), die in INTERLISP implementiert und Mitte 1983 auf einer Siemens-Rechenanlage installiert wurde. Diese Programmentwicklungsumgebung unterstützt es, Programme hierarchisch zu strukturieren, in kleinen Schritten zu entwickeln und zu verifizieren (verify-while-develop) , und sie stellt in allen Phasen des Softwareentwicklungszyklus', von der formalen Anforderungsspezifikation bis zum Pascalprogramm, geeignete Werkzeuge bereit. Dieser Entwicklungsprozeß läßt sich in eine Folge abstrakter und konkreter Schritte zerlegen (Abb.1).

Die abstrakten Schritte gehen von einer formalen Anforderungsspezifikation (axiomatisch) aus und enden in konstruktiv definierten Modellen (algorithmisch) . Dabei kann man nochmals in Verfeinerungsschritte und Implementierungsschritte klassifizieren. Erstere ersetzen Teile von Spezifikationen durch präzisere Anforderungen und verkleinern somit die Menge der Modelle. So wäre die Ersetzung axiomatisch spezifizierter Operationen durch konstruktiv definierte Operationen ein Verfeinerungsschritt. Mit Implementierungsschritten werden abstrakte Spezifikationen durch konkretere ersetzt. Dabei wird die Menge der Modelle so gewechselt, daß alle Rechnungen in der alten Modellmenge durch Rechnungen in der neuen Modellmenge simuliert werden können.

Die konkreten Schritte setzen bei konstruktiv definierten Modellen an und reichen bis zum ausführbaren PASCAL-Programm. Auch hier kann man nochmals in Realisierungsschritte und Vorübersetzungsschritte aufteilen. Realisierungsschritte ersetzen konkrete Spezifikationen durch Module aus MODPASCAL, eine um das Modulkonzept erweiterte PASCAL-Version, sodaß sich die Spezifikationshierarchie in der Modulhierarchie widerspiegelt. Vorübersetzungsschritte erzeugen aus einem MODPASCAL-Modul ein PASCAL-Programm.

Alle abstrakten Schritte lassen sich in der Spezifikationssprache ASPIK [BV 83a] ausdrücken, die aus einer Vorgängerversion (TRIPLEX) entstanden ist. Eine ASPIK-Spezifikation definiert durch ihre Signatur (Sorten- und Operationssymbole), ihre Formeln (Prädikatenlogik 1.Stufe) und ihre algorithmischen Definitionen eine Klasse von Algebren. Die Spezifikation kann hierarchisch strukturiert sein und kann instanziiert werden durch Ersetzung beliebiger, benutzter Spezifikationen [BGV 83]. Darüber hinaus ermöglicht ASPIK in Erweiterung von TRIPLEX die

Definition sowohl von Spezifikationen als auch von Verfeinerungs- und Implementierungsbeziehungen zwischen Spezifikationen als eigenständige Objekte, und ist deshalb sogar als Spezifikationsentwicklungssprache anzusehen.

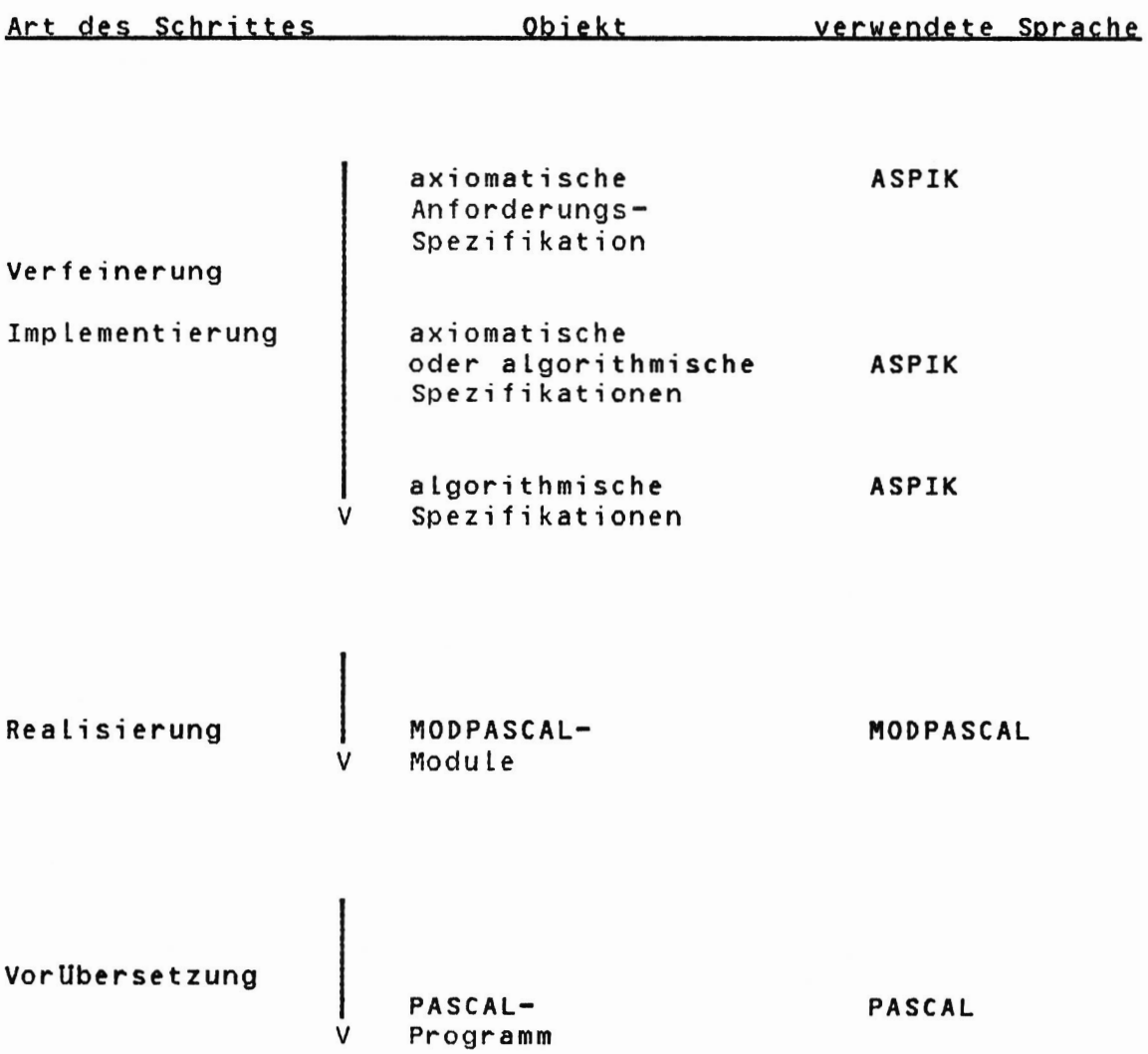


Abb.1 : Schrittweise Programmentwicklung

Die beiden Strukturierungsmechanismen von ASPIK wurden bereits angedeutet:
Die Use-Beziehung für Spezifikationen, Verfeinerungen und Implementierungen ermöglicht den hierarchischen Aufbau dieser Objekte. Das bedeutet insbesondere für Spezifikationen, daß sie andere Spezifikationen und somit deren Sorten- und Operationssymbole benutzen können. Das Parameterisierungskonzept "parameterization-by-use" ([BV 83a],[BV 83b]) kennt keine explizite Parameterdefinition , wie sie in der alten Spezifikationssprache TRIPLEX gefordert wurde, sondern läßt alle benutzten Spezifikationen als Parameter zu. Die Parameterspezifikation wird erst zum Zeitpunkt der Instanziierung festgelegt.

Zur Speicherung dieser ASPIK-Objekte bietet SPESY ein Dateikonzept an, das zwischen Privatdateien, die der Benutzer anlegt, und Systemdateien, die eine für jeden Benutzer zugängliche Bibliothek darstellen, unterscheidet. Für jede Datei wird eine Umgebung, bestehend aus Privat- und Systemdateien, definiert. Die Vorgängerversion erlaubte nur Systemdateien in der Umgebung einer Datei, um die Überprüfung auf Namenskonflikte zwischen Dateien zu vereinfachen. Innerhalb einer Datei sind alle darin definierten Objekte (Spezifikationen, Verfeinerungen, Implementierungen) sichtbar und zusätzlich alle sichtbaren Objekte aus der Umgebung.
SPESY ist ein Mehrbenutzersystem, wobei jedem Benutzer neben seinen Privatdateien auch die Systemdateien zur Verfügung stehen.

An Werkzeugen wurden in der Vorgängerversion interaktive Systeme zum Eingeben , Editieren und Instanziieren von TRIPLEX-Spezifikationen implementiert. Als Ergänzung zur Verifikation stand ein symbolischer Interpretierer als Werkzeug zur Verfügung, um konstruktive Spezifikationen zu testen. Diese Werkzeuge werden in die aktuelle Version übernommen, nachdem sie der neuen Sprache angepaßt worden sind.

Da es in ASPIK außer Spezifikationen auch noch Abbildungen (Verfeinerungen und Implementierungen) gibt, sind Editoren und Eingabesysteme für beide Objekte in Arbeit. Weiterhin ist geplant, folgende externe Komponenten an SPESY anzuschließen:

- ein Programmtransformationssystem, um konstruktiv definierte Operationen semantikerhaltend zu optimieren [Ge 83]

- der Karlsruher Resolutionsbeweiser "Markgraf Karl Refutation Procedure" [BES 81] und ein in Bonn entwickeltes Termersetzungssystem [Tho 84], um Konsistenzüberprüfungen durchzuführen

- Komponenten zur Unterstützung der konkreten Schritte [Olt 83]

All diese Werkzeuge werden in SPESY auf verschiedenen Kommandoebenen (levels) angeboten, in Abhängigkeit von den manipulierten Objekten.

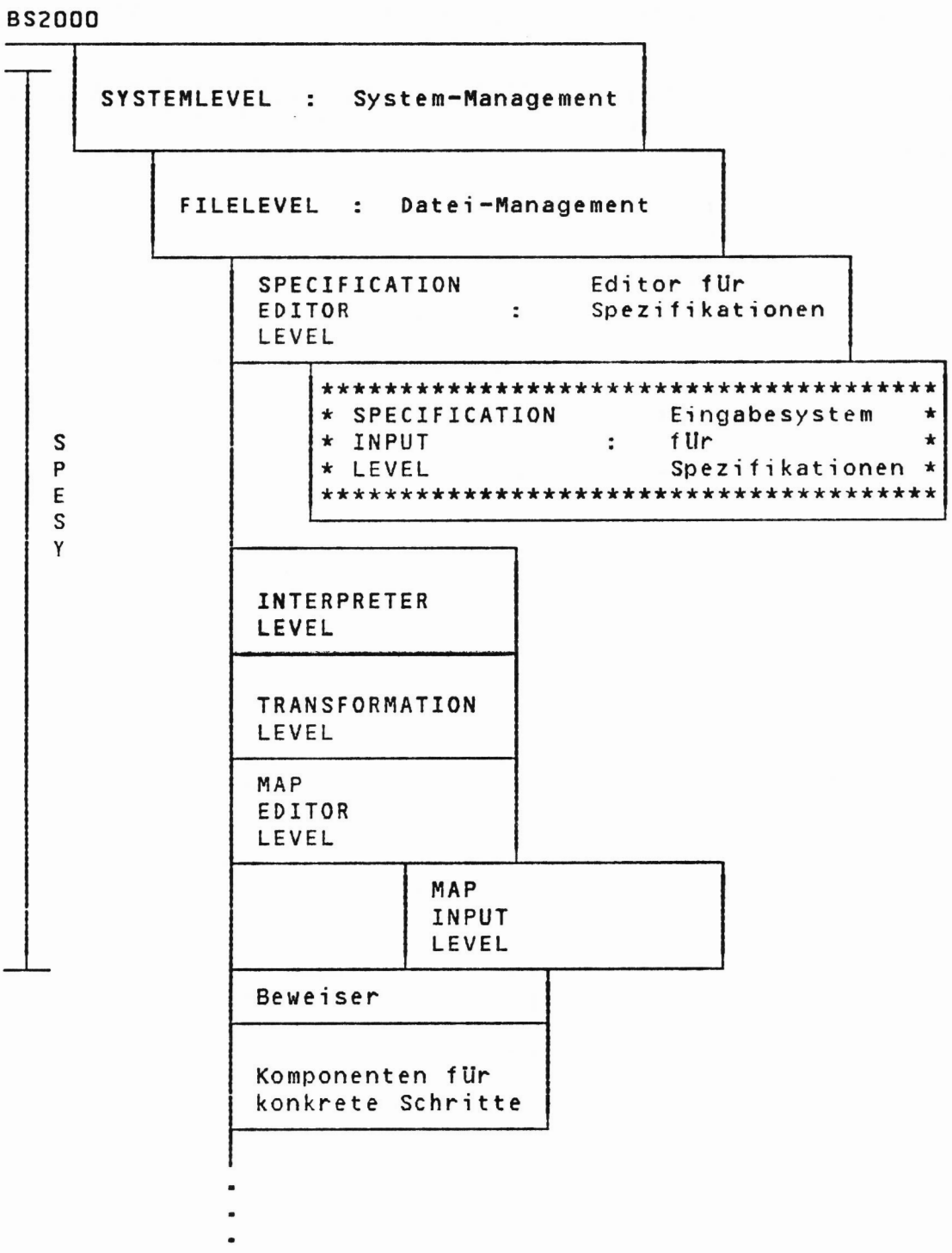


Abb.2 Grobstruktur von SPESY und Einbettung des Eingabesystems

1.2. Die in dieser Arbeit implementierte Komponente

Das Eingabesystem für Spezifikationen - das Thema dieser Arbeit - ermöglicht die interaktive und syntaxorientierte Definition einer neuen ASPIK-Spezifikation oder die Erweiterung einer schon bestehenden, solange sie sich auf einer Privatdatei befindet. Während der Eingabe wird die interne Darstellung der Spezifikation in Form eines Syntaxbaumes (Abb.3) aufgebaut. Der Anfangszustand bei der Eingabe einer neuen Spezifikation ist durch den leeren Syntaxbaum gekennzeichnet. Bei der Eingabe einer alten Spezifikation läßt sich der Syntaxbaum in zwei Teile zerlegen, in einen vollständig gefüllten, kontextfrei und kontextsensitiv korrekten Teil sowie in einen leeren Teil. Diese Vorbedingung muß am Anfang, während und am Ende der Eingabe erfüllt sein, um kontextsensitive Korrektheitseigenschaften garantieren zu können. In der Vorgängerversion von SPESY wurden solche Vorbedingungen nicht gefordert, um eine Top-Down-Entwicklung von Spezifikationen zu ermöglichen. Als Konsequenz dieser Entwurfsmethodik durfte man Knoten im Syntaxbaum leer lassen und undefinierte Sorten, Operationen und Spezifikationen referieren, wodurch kontextsensitive Tests unsinnig wurden. In der neuen Version wurde dieses Prinzip gegen folgendes ausgetauscht:

Falls man bei der Definition einer Spezifikation Systemunterstützung will, muß man die vom System erkannten Fehler korrigieren, bevor man die Eingabe fortsetzen kann.

Semantische Eigenschaften wie Konsistenz oder Terminierung von Algorithmen überprüft das Eingabesystem nicht. Das ist Aufgabe der Beweiser.

Zusätzlich zu den vorgenannten Anforderungen an das Eingabesystem (syntaxorientierter Dialog, Fehlertests) behandelt diese Arbeit noch zwei spezielle Problemstellungen, deren Bedeutung über das Eingabesystem hinausreicht :

1. die Normalisierung eines Specterms spect bestimmt den eindeutigen Repräsentanten aus der Menge äquivalenter Specterme

2. das Exported Interface eines Specterms spect legt die in spect verwendbaren Sorten- und Operationssymbole fest

Die Arbeit gliedert sich in zwei Teile. Der Band I ist als Benutzer-Manual zu verstehen und umfaßt die Kapitel 1 bis 4. Im Band II sind die Systemdokumentation und die Programmlistings zu finden.

In Kapitel 2 sind Algorithmen fUr beide Probleme in abstrakter Form angegeben.
Eine genaue Beschreibung des Eingabedialogs und der konkreten Syntax findet sich in Kapitel 3. Außerdem stellt dieses Kapitel alle getesteten Bedingungen zusammen und weist den durch diese Arbeit abgedeckten Teil des Eingabesystems aus.
Um einen Einblick in das Dialogverhalten zu vermitteln, sind in Kapitel 4 SPESY-Sitzungsprotokolle wiedergegeben.
In Kapitel 5 wird die Modulstruktur von SPESY dargestellt und die Programmstruktur, die Datenstrukturen und Funktionen des Eingabesystems erklärt.
In Kapitel 6 werden die Schnittstellen zu anderen Komponenten von SPESY dokumentiert.
Die benutzte Literatur ist in Kapitel 7 zusammengestellt. An einigen Stellen in dieser Arbeit werden Begriffe nur anschaulich erklärt, da sie Gegenstand nachfolgender Arbeiten sind.
In Kapitel 8 sind die Programmlistings aufgefUhrt.

2. Normalform und Interface eines Specterms

2.1. Die Normalform eines Specterms

Das neue Parameterisierungskonzept "parameterization-by-use" ([BV 83a],[BV 83b]) definiert Specterme (3.2.1.) zur Beschreibung von (evtl. zusammengesetzten) Instanziierungen. Verschiedene Specterme können die gleiche Semantik haben und werden dann als äquivalent bezeichnet. Ferner wird eine Normalform für Specterme definiert, sodaß jede Äquivalenzklasse genau einen Repräsentanten in Normalform besitzt. Ein Normalisierungsalgorithmus gestattet es, jeden Specterm in seine Normalform zu überführen. Durch Normalisieren läßt sich die Äquivalenz zweier Specterme entscheiden, was sowohl von der Syntaxanalyse (3.3.) als auch vom Algorithmus IFACE (2.2.) benötigt wird.

Ein Specterm spect ist in Normalform, wenn alle benutzten Specterme, die "über" einem formalen Parameter liegen, ebenfalls formale Parameter sind, wenn keine identischen Ersetzungen eines formalen Parameters durch sich selbst vorkommen und wenn der Specterm, der instanziiert werden soll, ein einfacher Spezifikationsname ist. Unter formalen Parametern eines Specterms spect versteht man die Quellen der Abbildungen in spect. Da der Algorithmus auf Listen statt auf Mengen arbeitet, müssen außerdem die Ersetzungen in einem Specterm in Normalform lexikographisch angeordnet sein.

Idee des Normalisierungsalgorithmus'

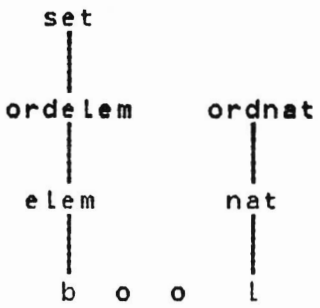
Der Normalisierungsalgorithmus NORM bestimmt die Normalform eines Specterms in vier Hauptschritten:

1. Einfügen von benutzten Abbildungen
   (Funktion INSERT-USED-MAPS)
2. Generieren von Applikationsabbildungen
   (Funktion DIRECT)
3. Komponieren von Abbildungen
   (Funktion COMPOSE)
4. Eliminieren von identischen Abbildungen
   (Funktion REDUCE)

In den nachfolgenden Beispielen werden diese Schritte weiter erläutert.

1. Beispiel :

Gegeben sei die folgende Hierarchie von Spezifikationen :



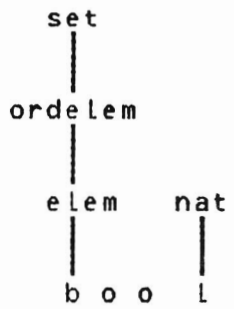
Erläuterung : die Use-Beziehung ist von oben nach unten zu lesen, d.h. set benutzt ordelem, ordelem benutzt elem, usw.

Zusätzlich sei die Abbildung ordelem --> ordnat definiert und benutze die Abbildung elem --> nat. Somit ist die Instanz set{ordelem --> ordnat} ein gültiger Specterm.
Eine erste Aufgabe des Algorithmus' besteht im Einfügen der benutzten Abbildungen.

Ergebnis von Schritt1 im Beispiel:
set{elem --> nat, ordelem --> ordnat}

2.Beispiel :

Gegeben sei die folgende Hierarchie :

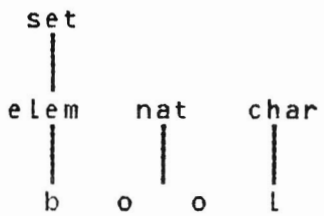


Zusätzlich sei die Abbildung elem --> nat definiert.
Dann ist die Instanz set{elem --> nat} zulässig, obwohl das sogenannte Between-Element ordelem nicht explizit aktualisiert worden ist. Aufgabe des Normalisierungsalgorithmus ist es, für solche Between-Elemente sogenannte Applikations-Abbildungen zu generieren und in die Instanz mit aufzunehmen.

Ergebnis von Schritt2 im Beispiel:
set{elem --> nat, ordelem --> ordelem{elem --> nat}}

3. Beispiel :

Gegeben sei die folgende Hierarchie:



Zusätzlich seien die Abbildungen
elem --> nat und
nat --> char definiert.

In den bisherigen Beispielen wurden nur einfache Specterme betrachtet, die dadurch gekennzeichnet sind, daß die Abbildungen parallel ausgeführt werden. Für einen zusammengesetzten Specterm, wie z.B. set{elem --> nat}{nat --> char}, sind die Abbildungen elem --> nat und nat --> elem sequentiell auszuführen. Der Algorithmus transformiert im dritten Schritt die zusammengesetzten Specterme durch Komposition von Abbildungen in einfache Specterme.

Ergebnis von Schritt3 im Beispiel:
set{elem --> nat --> char}

4. Beispiel :

Gegeben sei die folgende Hierarchie:

Zusätzlich sei die identische Abbildung elem --> elem definiert.
Dann ist der Specterm set{elem --> elem} zulässig. Im letzten Schritt eliminiert der Algorithmus die identischen Abbildungen.

Ergebnis von Schritt4 im Beispiel:
set

Bezeichner und deren Bedeutung im Algorithmus :

spect	Specterm (3.2.1.)
spect-nf	Specterm in Normalform
specid	Spezifikationsname
map	Mapterm (3.2.1.)
arrow	bezeichnet den Pfeil zwischen der Quelle und dem Ziel einer Map
ptr	jedem Mapterm ist eindeutig ein Pointer zugeordnet, da zwei syntaktisch verschiedene Mapterme semantisch äquivalent sein können
GET-MAP-PTR[map]	liefert den eindeutigen Pointer
GET-USED-MAPS[ptr]	liefert die Liste der benutzten Maps in Form ihrer Pointer
GET-MAP-SOURCE[ptr]	liefert die Quelle
GET-MAP-NAME[ptr]	liefert den Namen
GET-MAP-TARGET[ptr]	liefert das Ziel

GENERATE-MAP[source, target, used-maps, base-of-source]	generiert eine Applikations-Abbildung und liefert einen Pointer zurück
COMPOSE-MAP[ptr$_i$, ptr$_j$]	komponiert zwei Mapterme und liefert einen Pointer zurück
IS-IDENTITY-MAP[ptr]	prüft den mit ptr assoziierten Mapterm auf die identische Abbildung
map-lst	Liste von Maptermen
param-set	Liste der formalen Parameter eines Specterms
BASE[spect]	Liste der benutzten Specterme in Normalform (ohne spect selbst)
BETWEEN[spect, param-set]	Liste von Spectermen in Normalform, die von spect benutzt werden und ihrerseits einen einen Specterm aus param-set benutzen
union[lst$_1$, ... , lst$_n$]	Vereinigung der Listen lst$_1$ bis lst$_n$
intersection[lst$_i$, lst$_j$]	Durchschnitt der Listen lst$_i$ und lst$_j$
notmember[elem, lst]	liefert NIL, falls elem $\in$ lst TRUE sonst
diff[lst$_i$, lst$_j$]	bestimmt die Elemente der Liste lst$_i$, die nicht in lst$_j$ enthalten sind

Algorithmus NORM :

NORM überführt einen Specterm spect in seine interne Normalform spect-nf, indem er für alle in spect auftretenden Listen von Maptermen die Funktion NORM1 sukzessive mit folgenden Parametern aufruft :

1. normalisiertes Anfangsstück von spect
2. Liste von Abbildungen, die auf das normalisierte Anfangsstück von spect anzuwenden sind

Bemerkung: Die interne Darstellung der Normalform eines Specterm ist eine einfache Liste. Erstes Element der Liste ist der Spezifikationsname, alle weiteren Elemente stellen Pointer für Ersetzungen dar und sind lexikographisch geordnet.

NORM[specid] = specid

NORM[(specid map-lst$_1$ ... map-lst$_n$)] =

NORM[(NORM1[(specid map-lst$_1$)] map-lst$_2$ ... map-lst$_n$)]

NORM[spect-nf] = spect-nf

NORM[(spect-nf map-lst$_1$ ... map-lst$_n$)] =

NORM[(NORM1[(spect-nf map-lst$_1$)] map-lst$_2$ ... map-lst$_n$)]

NORM-MAP ersetzt alle Specterme in dem Mapterm (spect$_0$ arrow$_1$ spect$_1$ ... arrow$_n$ spect$_n$) durch ihre Normalform.

NORM-MAP[(spect$_0$ arrow$_1$ spect$_1$ ... arrow$_n$ spect$_n$)] =

let for i = 0, ... , n :

spect-nf$_i$ = NORM[spect$_i$] in

(spect-nf$_0$ arrow$_1$ spect-nf$_1$ ... arrow$_n$ spect-nf$_n$)

NORM1 erwartet einen Specterm (spect-nf map_1 ... map_n), wobei spect-nf ein Specterm in Normalform ist und map_1 ... map_n Abbildungen sind, die auf spect-nf angewendet werden sollen. NORM1 normalisiert mit den oben angegebenen Schritten 1 - 4.

```
NORM1[(spect-nf map_1 ... map_n)] =

    let for i = 1, ... , n :

        ptr_i = GET-MAP-PTR[NORM-MAP[map_i]] in

    REDUCE [COMPOSE [DIRECT [INSERT-USED-MAPS

                [(spect-nf ptr_1 ... ptr_n]]]]
```

INSERT-USED-MAPS erwartet einen Specterm (spect-nf ptr_1 ... ptr_n), wobei die Ersetzungen durch ihre eindeutigen Pointer dargestellt sind. INSERT-USED-MAPS fügt für alle Pointer die benutzten Abbildungen in Form ihrer Pointer hinzu.

```
INSERT-USED-MAPS[(spect-nf ptr_1 ... ptr_n)] =

        let (ptr*_1 ... ptr*_K)

              = union [GET-USED-MAPS[ptr_1], ...

                      ,GET-USED-MAPS[ptr_n],

                      (ptr_1 ... ptr_n)]

          in

        (spect-nf ptr*_1 ... ptr*_K )
```

DIRECT erwartet einen Specterm (spect-nf ptr_1 ... ptr_n). DIRECT generiert für jedes Between-Element eine sogenannte Applikations-Abbildung und fügt deren Pointer hinzu.
Hinweis: der Ausdruck $j_{L,m}$ wird als Index durch $;_{L,m}$ dargestellt.

```
DIRECT[(spect-nf ptr_1 ... ptr_n)] =
    let for i = 1, ... , n :
        source_i = GET-MAP-SOURCE[ptr_i]
        name_i   = GET-MAP-NAME[ptr_i]
        target_i = GET-MAP-TARGET[ptr_i] in
      let map_i = source_i name_i target_i in
        let param-set = (source_1 ... source_n) in
          let between-set = BETWEEN[spect-nf,param-set] in
            let between-set-min
                  = (bet-spect | bet-spect ist minimal in
                                 between-set)
                  = (betw_1 ... betw_k) in
              if between-set-min = ()
              then (spect-nf ptr_1 ... ptr_n)
              else let for l = 1, ... , k :
                     param-set_L
                           = intersection [param-set, BASE[betw_L]]
                           = (source_{i_L,1} ... source_{i_L,n_L}) in
let ptr-betw_L
  = GENERATE-MAP[betw_L, NORM1[(betw_L map_{i_L,1} ... map_{i_L,n_L})],
                 (ptr_{i_L,1}, ... ,ptr_{i_L,n_L}), BASE[betw_L]] in
    DIRECT[(spect-nf ptr_1 ... ptr_n ptr-betw_1 ... ptr-betw_k)]
```

COMPOSE erwartet als Argument einen Specterm (spect-nf $ptr_{2,1}$ ... $ptr_{2,m}$), wobei spect-nf ein Specterm in Normalform ist. COMPOSE komponiert die Ersetzungen in spect-nf mit den Ersetzungen $ptr_{2,1}$, ... , $ptr_{2,m}$ und liefert den Spezifikationsnamen aus spect-nf zusammen mit den komponierten Ersetzungen als Specterm zurück.

COMPOSE[(specid $ptr_{2,1}$ ... $ptr_{2,m}$)] =

(specid $ptr_{2,1}$ ... $ptr_{2,m}$)

COMPOSE[((specid $ptr_{1,1}$ ... $ptr_{1,n}$) $ptr_{2,1}$ ... $ptr_{2,m}$)] =

let for i = 1, ... , n :

for j = 1, ... , m :

$source_{1,i}$ = GET-MAP-SOURCE[$ptr_{1,i}$]

$name_{1,i}$ = GET-MAP-NAME[$ptr_{1,i}$]

$target_{1,i}$ = GET-MAP-NAME[$ptr_{1,i}$]

$source_{2,j}$ = GET-MAP-SOURCE[$ptr_{2,j}$]

$name_{2,j}$ = GET-MAP-NAME[$ptr_{2,j}$]

$target_{2,j}$ = GET-MAP-TARGET[$ptr_{2,j}$] in

let $param\text{-}set_1$ = ($source_{1,1}$ ... $source_{1,n}$)

$param\text{-}set_2$ = ($source_{2,1}$ ... $source_{2,m}$) in

let param-set'$_2$ =

intersection [$param\text{-}set_2$, BASE[specid]] in

let param-set = union [$param\text{-}set_1$, param-set'$_2$]

= ($source_1$ ... $source_L$) in

let for k = 1, ... , l :

$$ptr'_k = \begin{cases} ptr_{1,i} \text{ if } [source_{1,i} = source_k] \text{ and notmember } [target_{1,i}, param\text{-}set_2] \\ ptr_{2,j} \text{ if } [source_{2,j} = source_k] \text{ and notmember } [source_{2,j}, param\text{-}set_1] \\ COMPOSE\text{-}MAP[ptr_{1,i}, ptr_{2,j}] \text{ if } [source_k = source_{1,i}] \text{ and } [target_{1,i} = source_{2,j}] \end{cases}$$

in

(specid ptr'$_1$ ... ptr'$_l$)

REDUCE erwartet einen Spezifikationsnamen oder einen Specterm (specid ptr_1 ... ptr_n), wobei specid ein Spezifikationsname ist. REDUCE eliminiert die identischen Ersetzungen.

```
REDUCE[specid] = specid

REDUCE[(specid ptr_1 ... ptr_n)] =

      let for i = 1, ... , n :

          source_i  = GET-MAP-SOURCE[ptr_i]

          name_i    = GET-MAP-NAME[ptr_i]

          target_i  = GET-MAP-TARGET[ptr_i] in

        let param-set = (source_1 ... source_n) in

          let param-set-for-identity-maps =

              (source_k | source_k ist minimal in param-set  und

                          IS-IDENTITY-MAP[ptr_k]) in

            let sources-of-not-id-maps

                = diff [param-set,param-set-for-identity-maps]

                = (source_i1 ... source_ij) in

            REDUCE[(specid ptr_i1 ... ptr_ij)]
```

Mit Hilfe der Normalform läßt sich die folgende Relation definieren :

spect$_i$ ist semantisch äquivalent zu spect$_j$, falls
NORM[spect$_i$] = NORM[spect$_j$]

2.2. Das Exported Interface eines Specterms

Das Exported Interface oder einfach "Interface" eines Specterms spect legt die in spect verwendbaren Sorten- und Operationssymbole fest. Es bestimmt sich aus der Signatur und dem Imported Interface von spect.

Die Signatur eines Specterms spect ist bestimmt durch die von spect eingeführten Sorten- und Operationssymbole.

Das Imported Interface von spect ist die Vereinigung des Exported Interfaces aller von spect benutzten Specterme $spect_1$ ... $spect_n$ mit folgender Modifikation :

FÜhren zwei verschiedene Specterme dasselbe Sorten- oder Operationssymbol ein, so muß dieses Symbol mit dem entsprechenden Specterm gepräfixt werden.

Das Exported Interface von spect ergibt sich aus dem Imported Interface von spect durch Hinzufügen der Signatur von spect. Nun müssen aber auch alle die von einem benutzten Specterm $spect_i$ eingeführten Symbole gepräfixt werden, die mehrdeutig in Bezug auf die neu eingeführte Signatur sind.

Der Algorithmus IFACE bestimmt das Exported Interface eines Specterms spect durch Bestimmung von Repräsentanten $spect_1$, ... ,$spect_n$ für alle benutzten Specterme und listet deren zugehörigen Sorten und Operationen in der folgenden Form :

** THE EXPORTED INTERFACE OF spect : **

SORTS FROM $spect_1$
. . .

OPERATIONS FROM $spect_1$
. . .

SORTS FROM $spect_2$
. . .

OPERATIONS FROM $spect_2$
. . .

.
.
.

SORTS FROM $spect_n$
. . .

OPERATIONS FROM $spect_n$
. . .

Prinzipiell kann man das Exported Interface entweder aus einem globalen oder aus einem lokalen Blickwinkel betrachten. In beiden Fällen werden die Repräsentanten für die benutzten Specterme aus der Sicht von spect bestimmt. Aus der globalen Sicht bilden diese Repräsentanten auch die Präfixe für alle Sorten- und Operationsnamen. Aus der lokalen Sicht werden die Präfixe für die Signatur eines benutzten Specterms $spect_i$ durch die Repräsentanten aus der Sicht von $spect_i$ gebildet.
Die beiden Sichtweisen unterscheiden sich also lediglich durch die Wahl der Präfixe für die Sorten- und Operationsnamen, die jedoch äquivalent sind. Der Algorithmus IFACE arbeitet mit der globalen Sichtweise.

Ein erstes Problem resultiert aus der Tatsache, daß in der Hierarchie von spect semantisch äquivalente Specterme auftreten können. Im Interface soll jedoch nur ein Repräsentant einer Äquivalenzklasse aufgeführt sein. Der Algorithmus IFACE löst dieses Problem, indem er nach der "depth-first- Methode" die Hierarchie von spect durchläuft und den ersten, in dieser Reihenfolge auftretenden Repräsentanten einer Äquivalenzklasse in das Interface aufnimmt.

Ein weiteres Problem entsteht durch die globale Sichtweise, die die Liste der gültigen Präfixe für Sorten einer Operation global definiert. Das hat zur Folge, daß die von einem Specterm $spec_i$ lokal festgelegten Präfixe nicht mit den von spect global festgelegten Präfixe übereinstimmen müssen. Das nachfolgende Beispiel veranschaulicht diese Problematik :

Beispiel :

Es gelte :

* USED-SPECTS[spect] liefert alle Specterme im Interface von spect nach der Methode depth-first

* $spect_i, spect_j \in$ USED-SPECTS[spect]

* $spect_{i,k} \in$ USED-SPECTS[$spect_i$] und führt die Sorte s neu ein

* $spect_{j,k} \in$ USED-SPECTS[$spect_j$] und führt die Sorte s neu ein

* $spect_{i,k}$ ist semantisch äquivalent zu $spect_{j,k}$

* spect$_i$ definiert eine neue Operation op$_i$: --> spect$_{i,K}$.s

* spect$_j$ definiert eine neue Operation op$_j$: --> spect$_{j,K}$.s

Aus der Sicht von spect liegt entweder spect$_{i,K}$ oder spect$_{j,K}$ im Interface. Sei etwa spect$_{i,K}$ ∈ USED-SPECTS[spect]. Dann folgt, daß spect$_{j,K}$ kein gültiger Präfix ist im Interface von spect, sodaß die Operation op$_j$, die die Sorte spect$_{j,K}$.s referiert, erst ins Interface aufgenommen werden kann, nachdem der Präfix spect$_{j,K}$ durch spect$_{i,K}$ ersetzt worden ist. Deshalb muß der Algorithmus IFACE bei der Bestimmung der Operationen eine Umrechnung der lokal gültigen Präfixe in die gloabal gültigen Präfixe durchführen.

Eine weitere Anforderung besteht darin, daß nur "benutzerdefinierte Specterme" in das Interface aufgenommen werden sollen. Auf solche Specterme dürfen vom System keine Äquivalenzumformungen angewendet worden sein, wie etwa Komposition von Abbildungen oder Einfügen von Applikations-Abbildungen. Die Funktion BASE (2.1.) zur Bestimmung der durch spect benutzten Specterme ist in diesem Algorithmus aus zwei Gründen nicht geeignet und wird durch die Funktion USED-SPECTS ersetzt :

1. BASE[spect] liefert die von spect benutzten Specterme außer spect selbst, während USED-SPECTS[spect] alle benutzten Specterme liefert einschließlich spect.

2. BASE[spect] liefert Specterme in Normalform, während USED-SPECTS[spect] Repräsentanten aus der globalen Sicht von spect bestimmt.

Idee des Algorithmus IFACE

Der Algorithmus IFACE bestimmt das Exported Interface eines Specterms spect in vier Hauptschritten:

1. bestimme die Repräsentanten der von spect benutzten specterme
   (Funktion USED-SPECTS)

2. bestimme die Sorten und Operationen dieser Repräsentanten
   (Funktionen SORTS-OF und OPHS-OF)

3. bestimme die Sortenabbildung der Repräsentanten.
   Die Sortenabbildung wird benötigt, um die Operationen jedes Repräsentanten anpassen zu können. Aus der Sortenabbildung ergibt sich die Umrechnung der lokal gültigen Präfixe in die global gültigen Präfixe. Außerdem beinhaltet die Sortenabbildung die Abbildung der importierten Sorten gemäß den Ersetzungen in dem entsprechenden Repräsentanten und die identische Abbildung der in diesem Repräsentanten neu eingeführten Sorten. Abschließend werden die Präfixe vor eindeutigen Zielsorten in der Sortenabbildung entfernt.
   (Funktion GET-SORTMAP-OF)

4. passe die Sorten der Operationen gemäß dieser Sortenabbildung an
   (Funktion GET-UPDATE-OPH)

Die Schritte 3 und 4 sind notwendig, da in einer Operation eines benutzten Specterms spect; z.B. Sorten referiert werden können, die durch eine Ersetzung in spect; abgebildet werden. Dies wird in einem einfachen Beispiel verdeutlicht :

Beispiel:

Sei set{elem --> nat} ein Repräsentant aus dem zu bestimmenden Interface. Seien die Spezifikationen set, elem und nat definiert:

```
SPEC set
USE elem
SORTS set
OPS genset: elem --> set

SPEC elem
USE bool
SORTS elem

SPEC nat
USE bool
SORTS nat
```

Außerdem sei bool wie Üblich definiert.
Die Abbildung elem --> nat sei durch folgende Sortenabbildung definiert : elem -> nat

Schritt2 bestimmt folgende Sorten und Operationen für set{elem --> nat} :

```
    Sorten      : set
    Operationen : genset: elem --> set
```

Schritt3 bestimmt folgende Sortenabbildung für set{elem --> nat} :

```
    Sortenabbildung : elem -> nat
```

Schritt4 modifiziert die Stelligkeit der Operation genset wie folgt :

```
    genset: nat --> set
```

Bezeichner und deren Bedeutung im Algorithmus :

spect	Specterm (3.2.1.)
specid	Spezifikationsname
map	Mapterm (3.2.1.)
iface-spects$_i$	Liste der lokal zu spect$_i$ gültigen Präfixe
iface-spects	Liste der global gültigen Präfixe, also aus der Sicht von spect
USE-CLAUSE-OF[specid]	USE-Klausel von specid (Abb.4)
SORTS-CLAUSE-OF[specid]	SORTS-Klausel von specid (Abb.4)
OPS-CLAUSE-OF[specid]	OPS-Klausel von specid (Abb.4)
SINGLES-OF[sorts-list$_1$, ... , sorts-lst$_n$]	bestimmt die Liste der Sorten, die in genau einer der Listen sorts-lst$_1$ ... sorts-lst$_n$ auftreten
TARGET-OF[map]	bestimmt das Ziel von map
SOURCE-OF[map]	bestimmt die Quelle von map

GET-SORTMAP-OF[spect$_i$, iface-sorts$_i$, iface-spects$_i$, iface-spects, single-sorts]	bestimmt die vollständige Sortenabbildung für den Repräsentanten spect$_i$, um die Opheader OPHS-OF[spect$_i$] anpassen zu können. Die Sortenabbildung für den Repräsentanten spect$_i$ beinhaltet neben der Umrechnung der lokal gültigen Präfixe (iface-spects$_i$) in die global gültigen Präfixe (iface-spects) auch die Abbildung der importierten Sorten gemäß den Ersetzungen in spect$_i$ und die identische Abbildung der von spect$_i$ neu eingeführten Sorten iface-sorts$_i$. Anschließend wird für jedes Paar (old-sortid . new-sortid) der Sortenabbildung geprüft, ob new-sortid gepräfixt ist und eindeutig (single-sorts) ist im Interface von spect. Wenn das der Fall ist, wird dieses Paar durch (old-sortid . new-sortid-unprefixed) ersetzt, wobei new-sortid-unprefixed aus new-sortid durch Entfernen des Präfixes in new-sortid entsteht.
UPDATE-OPH[opheader,sortmap]	bestimmt aus der Operationsdeklaration opheader eine neue Operationsdeklaration durch Anwenden der Sortenabbildung auf alle Sorten von opheader
IFACE-UNION[spect-lst$_1$, ... spect-lst$_n$]	bestimmt die Vereinigungsliste der Specterm-Listen spect-lst$_1$ bis spect-lst$_n$ wie folgt: spect-lst$_1$ ist in der Ergebnisliste. Für jede Specterm-Liste spect-lst$_i$, i = 2, ... , n wird sukzessive für jeden Specterm spect$_{i,i_1}$ überprüft, ob spect$_{i,i_1}$ äquivalent ist zu einem Specterm spect aus der Ergebnisliste. Trifft dies nicht zu, dann wird spect$_{i,i_1}$

	in die Ergebnisliste aufgenommen, andernfalls nicht.
list[(spect$_1$ sorts$_1$ ophs$_1$), . . . (spect$_n$ sorts$_n$ ophs$_n$)]	listet die Specterme spect$_i$ und deren Sorten sorts$_i$ und Operationen ophs$_i$ gemäß dem oben angegebenen Schema.
is-inclusion[lst$_1$, lst$_2$]	liefert TRUE, falls jedes Element der Liste lst$_1$ in der Liste lst$_2$ enthalten ist; NIL sonst

Algorithmus IFACE :

IFACE erwartet einen Specterm spect und listet das Exported Interface von spect gemäß den Schritten 1 bis 4.

```
IFACE[spect] =
    let iface-spects  = USED-SPECTS[spect]
                      = (spect_1 ... spect_n) in
      let for i = 1, ... , n :
          iface-sorts_i  = SORTS-OF[spect_i]
          ophs_i         = OPHS-OF[spect_i]
          iface-spects_i = USED-SPECTS[spect_i] in
        let single-sorts = SINGLES-OF[iface-sorts_1, ...
                                      ,iface-sorts_n]
                         = (single_1 ... single_k) in
          let sortmap_i
              = GET-SORTMAP-OF[spect_i, iface-sorts_i,
                  iface-spects_i, iface-spects, single-sorts] in
            let iface-ophs_i = UPDATE-OPH[ophs_i, sortmap_i] in
            list [(spect_1 iface-sorts_1 iface-ophs_1)
                                  .
                                  .
                                  .
                  (spect iface-sorts_n iface-ophs_n)]
```

USED-SPECTS bestimmt für einen Specterm spect die Liste ($spect_1$ ... $spect_n$) der Repräsentanten der benutzten Specterme einschließlich spect selbst.

```
USED-SPECTS[specid] =

        let use-clause  = USE-CLAUSE-OF[specid]

                        = (spect_1 ... spect_L) in

    IFACE-UNION[(specid), USED-SPECTS[spect_1], ...

                USED-SPECTS[spect_L]]


USED-SPECTS[spect map_1 ... map_n] =

        let for i = 1, ... , n :

            target_i = TARGET-OF[map_i]

            source_i = SOURCE-OF[map_i]

          let between-set

              = diff [USED-SPECTS [spect],

                        union [USED-SPECTS[source_1], ...

                              , USED-SPECTS[source_n]]

              = (betw_1 ... betw_K) in

            let for j = 1, ... , k :

                used-maps-spect_j =

                   (map_m | USING[betw_j, source_m]) in

  IFACE-UNION [USED-SPECTS [target_1], ... ,

              USED-SPECTS[target_n],

              (( spect_1 used-maps-spect_1) ...

              (spect_K used-maps-spect_K ))
```

USING erwartet zwei Specterme $spect_i$ und $spect_j$ und überprüft, ob $spect_i$ $spect_j$ benutzt.

USING[$spect_i$, $spect_j$]

= is-inclusion [union [NORM[$spect_j$], BASE[$spect_j$]],
union [(NORM[$spect_i$]), BASE[$spect_i$]]]

SORTS-OF[specid] = SORTS-CLAUSE-OF[specid]

SORTS-OF[spect map_1 ... map_n] = SORTS-OF[spect]

OPHS-OF[specid] = OPS-CLAUSE-OF[specid]

OPHS-OF[spect map_1 ... map_n] = OPHS-OF[spect]

3. Benutzeranleitung

3.1. Allgemeine Beschreibung des Eingabedialogs

Der Aufruf des Eingabesystems erfolgt mit einem der folgenden Kommandos :

Auf dem Filelevel :

INPUT SPEC specid, falls die Spezifikation specid neu ist.
INPUT specid, falls die Spezifikation specid erweitert werden soll.
INPUT FROM filename, falls die Spezifikation auf der Datei filename steht.

Im Editor :

INPUT, falls die editierte Spezifikation erweitert werden soll.

Nach dem Start des Eingabesystems wird zuerst folgende Vorbedingung überprüft :
Der Syntaxbaum einer alten Spezifikation läßt sich in zwei Teile zerlegen, in einen vollständig gefüllten, kontextfrei und kontextsensitiv korrekten Teil sowie in einen leeren Teil. Der Syntaxbaum einer neuen Spezifikation ist leer. Außer dieser Vorbedingung gibt es noch eine mehr technische Bedingung. Der nächste leere Knoten, auf den bei der Eingabe einer alten Spezifikation positioniert werden soll, muß ein Hauptknoten sein (Abb.3). Damit definiert man eindeutige "Aufsetzpunkte" für eine alte Spezifikation. Das von diesen Aufsetzpunkten gebildete Raster ist als Kompromiß zwischen Benutzerfreundlichkeit und Programmieraufwand entstanden.

Wenn das Eingabesystem aktiviert wird und alle Vorbedingungen erfüllt sind, wird auf den nächsten Hauptknoten im Syntaxbaum positioniert. Falls die Eingabe automatisch generiert werden kann, erzeugt SPESY diese Eingabe und fordert weitere Eingabe an. Diese systemgenerierten Teile einer Spezifikation sind in Abb.4 speziell gekennzeichnet und können klein sein (Schlüsselwort am Anfang einer Klausel) oder einen vollständigen Hauptknoten umfassen (ein define-constructor-ops-element). Der Benutzer kann seine Eingabe bis zu einer beliebigen sogenannten Promptposition fortsetzen, d.h. bis zur nächsten, im Extremfall bis zur letzten Promptposition im Syntaxbaum (Abb.4). Dieses Konzept erlaubt es dem mit ASPIK vertrauten Benutzer, seine Spezifikation an einem Stück einzugeben, während der unerfahrene Benutzer durch die Syntax "gelotst" wird. Zusätzlich ist es möglich, eine Spezifikation von einer Datei zu lesen. Tritt hierbei ein Fehler auf oder wird das Ende der Spezifikation erkannt, so schaltet SPESY auf Eingabe vom Bildschirm um und beendet das Einlesen von der Datei.

Falls das System in der Benutzereingabe keinen Fehler entdeckt, kann der Dialog wie vorher beschrieben fortgesetzt werden. Ansonsten wird eine Fehlermeldung ausgegeben, aus der die Fehlerursache und die Position, ab der die Eingabe eventuell ignoriert wurde, hervorgeht. Gleichzeitig wird der Dialog an die letzte Promptposition zurückgesetzt, sodaß der Benutzer den Fehler korrigieren kann.

Die alte SPESY-Version listete ebenfalls die Fehlermeldung, aber es gab keine Möglichkeit, den Fehler noch während der Eingabe zu korrigieren. Dieses Verhalten resultierte ebenfalls aus der zugrunde gelegten Entwurfsphilosophie für Spezifikationen, die es gestattete, undefinierte Bezeichner zu verwenden, die erst später definiert werden brauchten. So mußte man bei einem einfachen Schreibfehler zuerst durch den ganzen Dialog gehen bis zum Ende der Spezifikation. Dann mußte der Benutzer die Spezifikation mit dem Editor (allerdings ohne Unterstützung) korrigieren.

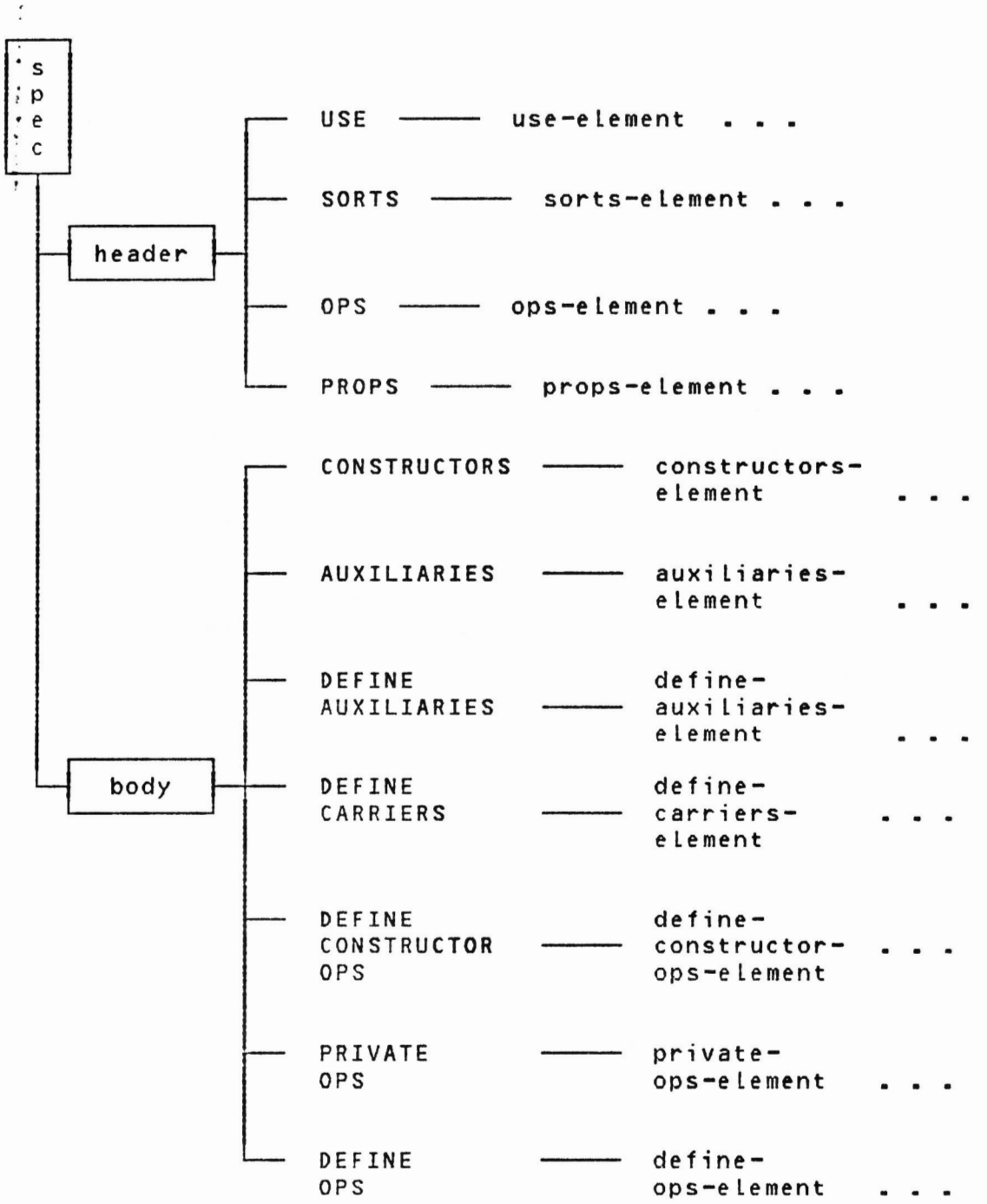


Abb.3 : Struktur des Syntaxbaumes einer Spezifikation. Hauptknoten (Klauseln und Klauselelemente) sind alle nicht umrahmten Knoten.

```
specification ::

PP SPEC PP specid [;]
[ PP comment [;]]
PP USE ( PP use-element [ comment ] ) . . . ;
[ PP SORTS [ PP sorts-element [ comment ] ] . . . ; ]
[ PP OPS [ PP ops-element [ comment ] ] . . . ; ]

**********************************************************

[ PP PROPS [ PP props-element [ comment ] ] . . . ; ]

[ PP SPECBODY [comment]
[  PP  CONSTRUCTORS  [  PP   constructors-element   [   comment
   ]]...[;]]
 [ PP AUXILIARIES [ PP auxiliaries-element [ comment ]] ... ;
   PP DEFINE AUXILIARIES     [ PP define-auxiliaries-element
                             [ comment ] ] . . . [;] ]
 [ PP DEFINE CARRIERS        [ define-carriers-element
                             [ comment ] ] . . . [;]
 [ PP DEFINE CONSTRUCTOR OPS [ define-constructor-ops-element
                             [ comment ] ] . . . [;] ] ]
 [ PP PRIVATE OPS [ PP private-ops-element [ comment ]] ... ;]
 [ PP DEFINE OPS [ PP define-ops-element [ comment ]] ... [;]]]
   PP ENDSPEC [;]
```

Notation :

Fettdruck kennzeichnet die automatisch generierbaren Teile einer Spezifikation

GROSSCHREIBUNG kennzeichnet die Terminalsymbole

:: trennt linke und rechte Seite einer Regel

() sind Meta-Klammern

[] umschließen optionale Teile

PP kennzeichnet die Promptpositionen

******** zeigt die Trennlinie an, bis zu der man momentan eine Spezifikation eingeben kann

Abb.4 : Syntax einer Spezifikation bis zu den Klauselelementen ohne Berücksichtigung der Leerzeichen und Kommata.

In der neuen Version stehen dem Benutzer an jeder Promptposition vier verschiedene Alternativen zur Verfügung :

1. den Dialog weiter fortsetzen

2. das System nach Help-Informationen fragen
   (Kommando HELP oder H oder ?)

3. den Dialog beenden mit Abspeichern der Spezifikation
   (Kommando END)

4. den Dialog beenden ohne Abspeichern der Spezifikation
   (Kommando ABORT)

Die beiden zuletzt genannten Möglichkeiten beenden den Eingabedialog, wonach sich der Benutzer im Editor befindet bzw. auf dem FILELEVEL, falls die Spezifikation vollständig ist.

Zusammenfassend gibt es in der neuen SPESY-Version drei wesentliche Änderungen :

1. Kontextsensitive Fehlerprüfungen finden schritthaltend mit der Eingabe statt und erlauben eine frühestmögliche Fehlererkennung. Der Dialog wird zurückgesetzt und eine Korrektur ermöglicht.

2. Die Eingabe einer Spezifikation kann an bestimmten, durch die ASPIK-Syntax vorgegebenen Promptpositionen beendet und zu einem späteren Zeitpunkt fortgesetzt werden.

3. Das Eingabesystem verarbeitet Spezifikationen, die aus einer Datei eingelesen werden.

Außer diesen konzeptionellen Änderungen ergaben sich noch Ergänzungen, die auf mehr Eingabekomfort abzielten:

- Nach jedem Klauselelement kann der Benutzer Kommentare eingeben, die mit /* beginnen und */ enden.
- Die Help-Informationen an jeder Promptposition sind mehrstufig und somit auf die verschiedenen Vorkenntnisse und Erfahrungen anpaßbar. Ist die angeforderte Help-Information zu allgemein gehalten, so liefert ein erneutes Help-Kommando detailliertere Auskünfte zur aktuellen Promptposition.
- Die restriktiven Bezeichnerkonventionen in der alten Version wurden weitgehend aufgehoben, sodaß der Benutzer möglichst frei seine Bezeichner wählen kann.

3.2. OK-Bedingungen

Schritthaltend mit der Eingabe werden lexikalische und syntaktische KorrektheitsÜberprÜfungen durchgeführt. Sobald ein Fehler erkannt ist, wird die noch nicht gelesene Eingabe ignoriert und eine Fehlermeldung zusammen mit einem Anfangsteil der ignorierten Eingabe auf dem Bildschirm angezeigt.

Die lexikalischen Konventionen und die konkrete ASPIK-Syntax, soweit für diese Arbeit relevant, sind im Abschnitt 3.3. aufgeführt. Die kontextsensitiven Bedingungen, kurz OK-Bedingungen, werden nachfolgend für all diejenigen Klauseln aufgeführt, die in dieser Arbeit implementiert worden sind (oberer Teil von Abb.4).

3.2.1. Begriffliche Grundlagen

Umgebung einer Datei d :

alle Dateien, die beim Öffnen der Datei d geladen werden

Umgebung einer Spezifikation :

alle sichtbaren Objekte (Spezifikationen, Verfeinerungen, Implementierungen) der geöffneten Datei und alle sichtbaren Objekte in der Umgebung dieser Datei

Signatur einer Spezifikation spec :

Menge der durch spec neu eingeführten Sorten- und Operationssymbole

Imported Interface einer Spezifikation :

Vereinigung der Signaturen aller benutzten Spezifikationen gemäß 2.2.

Exported Interface einer Spezifikation spec :

Vereinigung der Signatur von spec mit dem Imported Interface von spec gemäß 2.2.

Ein **Mapid** :

ist ein Tripel (source arrow target) und bedeutet einen Signaturmorphismus (Verfeinerung) zwischen den Spezifikationen source und target, der weitere Mapterme benutzen kann.

Eine **Mapsequenz** :

ist eine Folge (source $arrow_0$ $spect_1$ ... $arrow_{n-1}$ $spect_n$ $arrow_n$ target) und bezeichnet die Komposition folgender Mapids : (source $arrow_0$ $spect_1$) , ($spect_1$ $arrow_1$ $spect_2$) , ... ($spect_n$ $arrow_n$ target).

Ein **Mapterm** :

ist ein Mapid oder eine Mapsequenz.

Ein **Specid** :

ist ein Spezifikationsname specid.

Ein **Specterm** :

ist ein Specid oder
eine Instanziierung (specterm $mapterm_1$ ... $mapterm_n$) von specterm gemäß den Maptermen $mapterm_1$ bis $mapterm_n$. Insbesondere kann specterm ein Specid sein.

Basis eines Mapterms :

Menge aller Spezifikationen, die sowohl von source als auch von target benutzt und identisch aufeinander abgebildet werden

Basis eines Specterms spect :

Menge der benutzten Specterme in Normalform (2.1. BASE[spect])

Die durch spect $_1$ ... spect $_n$ festgelegten Repräsentanten :

Sei spec eine Spezifikation mit der folgenden USE-Klausel :
USE $spect_1$ ... $spect_n$;
Dann liefert die Funktion USED-SPECTS[spec] die durch $spect_1$ ... $spect_n$ festgelegten Repräsentanten (2.2.).

Ein **Opid** opid :

bezeichnet ein Operationssymbol in Präfix- oder Mixfix-Schreibweise (Verallgemeinerung der Infix-Schreibweise)

Die Opparts von einer Operation opid :

Ist opid eine Präfix-Operation, so ist opid einziger oppart.
Ist opid eine Mixfix-Operation, so trennt der Unterstrich "_" die einzelnen Opparts von opid.

Ein **Opheader** :

$opid_1$ ... $opid_n$: $sortid_1$ ... $sortid_{m-1}$ --> $sortid_m$

Deklaration der Opids $opid_1$ bis $opid_n$ mit Stelligkeit $sortid_1$ x ... x $sortid_{m-1}$ --> $sortid_m$

3.2.2. Die OK-Bedingungen

Eine **Specid-clause** SPEC specid **ist OK** , falls :

1.Fall: specid ist in der Umgebung der geöffneten Datei schon definiert

a) specid ist auf einer Privatdatei abgespeichert

b) der Syntaxbaum für specid läßt sich in zwei Teile aufspalten - einen vollständig gefüllten, kontextfrei und kontextsensitiv korrekten Teil und einen leeren Teil

c) der nächste leere Knoten, auf den positioniert werden soll, ist ein Hauptknoten des Syntaxbaumes

2.Fall: specid ist eine neue Spezifikation

keine OK-Bedingung

Eine **Comment-clause** /* Das ist ein Kommentar */ **ist** immer **OK**

Eine **Use-clause** USE $specterm_1$, ... , $specterm_n$ **ist OK** , falls

für alle j=1, ... , n gilt :

a) $specterm_j$ ist interface-ok

b) $specterm_j$ ist nicht semantisch äquivalent zu $specterm_i$, $i < j$

c) ist $specterm_j$ semantisch äquivalent zu einem durch $specterm_1$ ... $specterm_{j-1}$ festgelegten Repräsentanten spect', so ist $specterm_j$ gleich spect'

Ein **Specterm** (spect $mapt_1$ ... $mapt_n$) **ist interface-ok** , falls

a) der Specterm spect interface-ok ist

b) die Mapterme $mapt_j$ interface-ok sind $j=1,...,n$

c) die Mapterme $mapt_j$ die Hierarchieforderung erfüllen $j=1,...,n$

d) die Quelle von $mapt_j$ eine von spect benutzte Spezifikation ist $j=1,...,n$

e) die Quelle von $mapt_j$ nicht äquivalent ist zur Quelle von $mapt_i$ $j=1,...,n$ $i<j$

f) die Quelle von $mapt_j$ nicht äquivalent ist zu einem Specterm aus der Basis von $mapt_i$ $j=1,...,n$ $i<j$

g) die Quelle von $mapt_j$ nicht durch einen vorherigen Mapterm $mapt_i$ anders abgebildet wird $j=1,...,n$ $i<j$

Eine **Spezifikation** specid **ist interface-ok** , falls

a) specid eine neue Spezifikation ist oder in der Umgebung definiert ist

b) use-clause, sorts-clause und ops-clause von specid OK sind.

Eine **Spezifikation** specid **ist OK** , falls

a) specid interface-ok ist

b) alle referierten Specids und Mapids OK sind

c) der Rest der Spezifikation specid OK ist (unterer Teil in Abb.4)

Ein **Mapterm** mapt **ist interface-ok** , falls

die Komponenten von mapt interface-ok sind

Ein **Mapid** (source name target) **ist interface-ok** , falls

anschaulich :
source und target sind interface-ok, die Sorten- und Operationsabbildung ist ein Signaturmorphismus (Verfeinerung) und jede referierte Mapid und Specid ist interface-ok.

Ein **Mapid** mapid **ist OK** , falls

alle referierten Specids und Mapids OK sind

Mapterme $mapt_1$ ... $mapt_n$ **erfüllen die Hierarchieforderung** , falls

für je zwei verschiedene Mapterme $mapt_i$ und $mapt_j$ gilt:

falls die Quelle von $mapt_i$ in der **Basis** der Quelle von $mapt_j$ liegt, dann liegt das Ziel von $mapt_i$ in der Basis des Ziels von $mapt_j$ oder das Ziel von map_i ist gleich dem Ziel von map_j

Eine **Sorts-clause** SORTS $sortid_1$ ... $sortid_n$ **ist OK** , falls

$sortid_j$ genau einmal definiert ist $j=1,...,n$

Eine **Ops-clause** OPS $opheader_1$ ... $opheader_n$ **ist OK** , falls

$opheader_i$ OK ist $i=1,...,n$

Ein **Opheader** $_L$ $opid_1$ ... $opid_n$: $sortid_1$ ... $sortid_{m-1}$ --> $sortid_m$ **ist OK** , falls

für alle $i = 1, ..., n$ und
$k = 1, ..., m$ gilt :

a) die Sorten $sortid_K$ sind aus dem Exported Interface der aktuellen Spezifikation. D.h. die Sorten sind gepräfixt, falls sie mehrdeutig sind. Der Präfix ist aus der Menge der gültigen Repräsentanten, die durch die Specterme der Use-clause festgelegt werden.

b) die Opparts von $opid_i$ treten genau einmal auf und sind verschieden von den Opparts der bereits eingegebenen $opheader_j$, $j<l$

c) $opid_i$ ist nicht reserviert

d) ist $opid_i$ eine Mixfix-Operation, so enthält $opid_i$ genau (m-1) Unterstriche für (m-1) Argumentpositionen

e) beginnt ein Oppart $oppart_{i,j}$ von $opid_i$ mit "EQ-" , so ist $oppart_{i,j}$ von der folgenden Form :
EQ-sortid , wobei sortid eine Sorte aus der Sorts-clause der aktuellen Spezifikation ist.
Außerdem gilt:

$m=3$
$sortid_1$ = sortid
$sortid_2$ = sortid
$sortid_m$ = BOOL

Ein **Opid** opid **ist nicht reserviert** , falls

a) die opparts von opid nicht mit "ERROR-" beginnen

b) die opparts von opid ungleich IS-sortid sind, wobei sortid eine Sorte aus der Sorts-clause der aktuellen Spezifikation ist

c) die opparts von opid nicht in folgender Menge enthalten sind :
{IF, THEN, ELSE, LET, CASE, ESAC, IN, &, |, _, ==>, <==>, =|=, ==, ALL, EX, OTHERWISE, ELSIF}
Hinweis: der Unterstrich ist das Ersatzsymbol für NOT

3.3. ASPIK-Syntax

3.3.1. Notation

FETTDRUCK	kennzeichnet die automatisch generierbaren Teile einer Spezifikation
::	trennt linke und rechte Seite einer Regel
[]	umschließen optionale Teile
|	trennt zwei Alternativen für die rechte Seite einer Regel
PP	kennzeichnet die Promptpositionen
...	kennzeichnen die Wiederholung des vorangehenden Teils
b	kennzeichnet das Leerzeichen

Nichtterminale der Grammatik :

a) die kleingeschriebenen Worte
b) die Symbole { } ; , : = --> /* */ b

Terminale der Grammatik :

a) die großgeschriebenen Worte
b) die Symbole **{ } ; , _ : = --> /* */ . b %**

3.3.2. Die ASPIK-Syntax

```
specification :: specid-clause
                 comment-clause
                 use-clause
                 [sorts-clause]
                 [ops-clause]
                 [props-clause]
                 [body]
                 [spec-end-clause]

specid-clause  :: PP SPEC PP specid [; | b]
specid :: simple-sortid :: simple-opid-part :: arrowid :: id

comment-clause :: PP [comment] [; | b]
comment        :: comment-part [b comment-part] ...
comment-part   :: /* non-blank-ch ...
                  [b non-blank-ch ...] ... */

use-clause     :: PP USE use-elem , nextuse
nextuse        :: PP [; | [use-elem , nextuse]]
use-elem       :: specterm [b comment]
specterm       :: specid |
                  specterm  [{simple-tripel  [,  simple-tripel]
                  ... }]
simple-tripel  :: PP specterm simple-arrowterm specterm
```

```
simple-arrowterm :: arrow | simple-arrow-composition
arrow            :: -[arrowid]->
simple-arrow-composition ::  simple-arrowterm
                             [specterm simple-arrowterm] ...

sorts-clause     :: PP SORTS next-sort
next-sort        :: PP [; | [sorts-elem , next-sort]]
sorts-elem       :: simple-sortid [b comment]

ops-clause       :: PP OPS next-op
next-op          :: PP [; | [ops-elem , next-op]]
ops-elem         :: opheader [b comment]
opheader         :: simple-opid [, simple-opid] ... :
                    [sortid [, sortid] ... ] --> sortid
simple-opid      :: [_] ... simple-opid-part
                    [_ ... simple-opid-part] ... [_] ...
sortid           :: [prefix.] simple-sortid
prefix           :: specid |
                    prefix [{prefix-tripel [,  prefix-tripel] ...
                    }]
prefix-tripel    :: prefix prefix-arrowterm prefix
prefix-arrowterm :: arrow | prefix-arrow-composition
prefix-arrow-composition ::  prefix-arrowterm
                             [prefix prefix-arrowterm] ...
props-clause     :: nicht Gegenstand dieser Arbeit
body             :: nicht Gegenstand dieser Arbeit
```

spec-end-clause :: nicht Gegenstand dieser Arbeit

b :: **b** ...

(das Return-Zeichen wird als **b** interpretiert)

, :: [b] , [b] | b

; :: [b] ; [b]

: :: [b] : [b]

= :: [b] = [b]

{ :: [b] { [b]

} :: [b] } [b]

- :: b -

-> :: -> b

/* :: [b] /* b

*/ :: b */ [b]

--> :: [b] --> [b]

non-blank-ch bezeichnet jedes Zeichen, mit Ausnahme des Leerzeichens

3.3.3. Lexikalische Konventionen

Die folgenden Sonderzeichen besitzen eine besondere syntaktische Funktion :

Sonderzeichen	Beispiel für syntaktische Funktion des Sonderzeichens
>	Spitze des Pfeils zwischen den Quellsorten und der Zielsorte eines Opheaders
=	trennt rechte und linke Seite einer Gleichung
:	trennt Operationsbezeichner von ihrer Stelligkeitsdefinition im Opheader
,	trennt zwei Klauselelememte
.	trennt den Präfix vom Bezeichner
;	trennt zwei Klauseln
{}	umschließen einen Applikationsterm eines Specterms
_	kennzeichnet die Argumentpositionen in einer Mixfix-Operation

Will man diese Sonderzeichen innerhalb eines Bezeicners benutzen, so muß jedem dieser Sonderzeichen das %-Zeichen vorangestellt werden, um die syntaktische Bedeutung aufzuheben.

Folgende Regeln für das Nichterminal id kommen zur Grammatik hinzu :

```
id                    :: simple-char [id] |
                         unspecialized-char [id]

unspecialized-char    :: % special-char

special-char          :: > | = | , | . | ; | { | } | _

simple-char           bezeichnet alle Zeichen, mit Ausnahme von
                                  b (Leerzeichen)
                                  %
                                  special-char (Sonderzeichen)
```

Die folgenden Konventionen vervollständigen die Syntax für einen Bezeichner :

a) der Bezeichners ist nicht länger als 30 Zeichen
b) der Bezeichner ist ungleich **NIL**
c) der Bezeichner ist keine Zahl
d) ist der Bezeichner gepräfixt, so ist die Gesamtlänge kleiner als 254 Zeichen

Zusammenfassung:

An einen gültigen Bezeichner id werden folgende Anforderungen gestellt :

a) jedem Sonderzeichen (special-char) innerhalb von id ist das %-Zeichen vorangestellt
b) das %-Zeichen steht nur vor diesen Sonderzeichen
c) id bezeichnet keine Zahl im Sinne von INTERLISP
d) id bezeichnet nicht das INTERLISP-Atom **NIL**
e) id ist nicht länger als 30 Zeichen
f) ist id gepräfixt, so überschreitet die Gesamtlänge 254 Zeichen nicht

```
(OUT)    ENTER CMD:
(OUT)
(IN)     list bool
(OUT)    spec   BOOL;
(OUT)    use
(OUT)    sorts  BOOL;
(OUT)    ops    TRUE,FALSE:  --> BOOL
(OUT)           NOT: BOOL  --> BOOL
(OUT)           _AND_,_OR_: BOOL BOOL  --> BOOL;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec ordelem
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* a reflexive, linear ordering on elem */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     bool;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     elem;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     _%<=_:elem elem-->bool
(OUT)    *** THE FOLLOWING MIXFIX-OPIDS ARE INVALID IN THE OPS-CLAUSE ***
(OUT)     _%<=_
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     ?
(OUT)    OP-HEADER::= OPIDS : [ DOMAINS ] --> CODOMAINS
(OUT)
(IN)     ?
(OUT)    OPIDS::= ID [ ,ID, ... ,ID ]
(OUT)
(IN)     ?
(OUT)    DOMAINS::= PREF_SORTID ... PREF_SORTID
(OUT)    CODOMAIN::= PREF_SORTID
(OUT)
(IN)     ?
(OUT)    PREF_SORTID::= [ SPECTERM .]SORTID
(OUT)
(IN)     ?
(OUT)    *** ENTER END OR ABORT ***
(OUT)
(IN)     end
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 3991 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     input
```

```
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ****** START WITH INPUT AT THE FOLLOWING CLAUSE ******
(OUT)    SORTS-CLAUSE
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     _<=_:elem elem-->bool
(OUT)    *** THE FOLLOWING MIXFIX-OPIDS ARE INVALID IN THE OPS-CLAUSE ***
(OUT)     _<=_
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     _<%=_:elem elem-->bool /* = is a special_char */
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     eq-elem :elem elem --> bool;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 2249 ms.
(OUT)    ****** E D I T O R L E V E L  FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list ordelem
(OUT)    spec   ORDELEM /* A SET WITH A REFLEXIVE, LINEAR ORDERING  */;
(OUT)    use    BOOL;
(OUT)    sorts  ELEM;
(OUT)    ops    _<%=_: ELEM ELEM  --> BOOL /* = IS A SPECIAL_CHAR */
(OUT)           EQ-ELEM: ELEM ELEM  --> BOOL;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec natord
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* the natural numbers with the standard ordering */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     bool;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     nat;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     _<%=_:nat nat-->bool
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     eq-nat:nat nat-->bool
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
```

```
(IN)     0:-->nat
(OUT)    *** THE FOLLOWING OPIDS ARE INVALID IN THE OPS-CLAUSE ***
(OUT)     0
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     null:-->nat
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     succ,pred:nat--> nat
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     _+_,_-_:nat nat--> nat
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 3632 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list ordnat
(OUT)    ILLEGAL SPECIFICATION NAME. COMMAND BROKEN.
(OUT)    ENTER CMD:
(OUT)
(IN)     list ornat
(OUT)    ILLEGAL SPECIFICATION NAME. COMMAND BROKEN.
(OUT)    ENTER CMD:
(OUT)
(IN)     list natord
(OUT)    spec   NATORD /* THE STANDARD ORDERING OF THE NATURAL NUMBERS */;
(OUT)    use    BOOL;
(OUT)    sorts  NAT;
(OUT)    ops    _<%=_: NAT NAT  --> BOOL
(OUT)           EQ-NAT: NAT NAT  --> BOOL
(OUT)           NULL:  --> NAT
(OUT)           SUCC,PRED: NAT  --> NAT
(OUT)           _+_,_-_: NAT NAT  --> NAT;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec list
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* list of anything */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     ordelem;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     list;
(OUT)    ops
```

```
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     empty:-->list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     put:elem list-->liist ,first:list-->elem
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  ,FIRST:LIST-- > ELEM
(OUT)    *** THE FOLLOWING SORTIDS ARE NEITHER IN THE INTERFACE NOR IN SORTS-CLAUSE ***
(OUT)     LIIST
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     put:elem list-->list ,first:list-->elem
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     rest:list-->list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     append:list list --> list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     empty?,simple?:list-->bool
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     in?:elem list-->bool;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 4890 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list list
(OUT)    spec   LIST /* LIST OF ANYTHING */;
(OUT)    use    ORDELEM;
(OUT)    sorts  LIST;
(OUT)    ops    EMPTY:  --> LIST
(OUT)           PUT: ELEM LIST  --> LIST
(OUT)           FIRST: LIST  --> ELEM
(OUT)           REST: LIST  --> LIST
(OUT)           APPEND: LIST LIST  --> LIST
(OUT)           EMPTY?,SIMPLE?: LIST  --> BOOL
(OUT)           IN?: ELEM LIST  --> BOOL;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec slots
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     list;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
```

```
(IN)    part1,part2:list-->list
(OUT)   BCST !!!!!!ACHTUNG LASERPRINTER DEFEKT**BITTE KEINE L P R ' S :180657 :84-03-26086
(OUT)   *** THE FOLLOWING SORTID IS INVALID IN THE SORTS-CLAUSE ***
(OUT)    :LIST-->LIST
(OUT)   ENTER SORTID COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   ops
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    part1,part2:list-->list
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    combine:list list-->list
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    simple_sort: list-->list
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    part1:list-->list
(OUT)   *** THE FOLLOWING OPIDS OCCUR TWO OR MORE TIMES IN THE OPS-CLAUSE, ACCEPTED ONLY ONCE ***
(OUT)    PART1
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   ******      E N D   I N P U T       ******
(OUT)   CPU TIME USED : 3174 ms.
(OUT)
(OUT)   ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)   ENTER EDITOR-CMD:
(OUT)
(IN)    end
(OUT)   ENTER CMD:
(OUT)
(IN)    list slots
(OUT)   spec    SLOTS;
(OUT)   use     LIST;
(OUT)   sorts   PART1,PART2;
(OUT)   ops     PART1,PART2: LIST  --> LIST
(OUT)           COMBINE: LIST LIST  --> LIST
(OUT)           SIMPLE_SORT: LIST  --> LIST;
(OUT)   ENTER CMD:
(OUT)
(IN)    edit slots
(OUT)
(OUT)   ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)   ENTER EDITOR-CMD:
(OUT)
(IN)    delp sopu
(OUT)
(OUT)   ENTER EDITOR-CMD:
(OUT)
(IN)    list slots
(OUT)   *** ILLEGAL COMMAND ***
(OUT)
(OUT)   ENTER EDITOR-CMD:
(OUT)
```

```
(IN)     end
(OUT)    EDITOR ENDED.
(OUT)    ENTER CMD:
(OUT)
(IN)     list slots
(OUT)    spec    SLOTS;
(OUT)    use     LIST;
(OUT)    sorts
(OUT)    ops     PART1,PART2: LIST  --> LIST
(OUT)            COMBINE: LIST LIST  --> LIST
(OUT)            SIMPLE_SORT: LIST  --> LIST;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec alg
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* sorts lists of anything algorihmically */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     use slots
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  SLOTS
(OUT)    *** THE FOLLOWING SPECTERM DOESN'T EXIST IN THE ENVIRONMENT ***
(OUT)    USE
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     slots
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     sort_list_:list-->list
(OUT)    *** THE FOLLOWING MIXFIX-OPIDS HAVE NO CORRECT ARITY ***
(OUT)     SORT_LIST_
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     sort:list-->list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 1592 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list alg
```

```
(OUT)    spec    ALG /* SORTS LISTS OF ANYTHING ALGORITHMICALLY */;
(OUT)    use     SLOTS;
(OUT)    sorts
(OUT)    ops     SORT: LIST  --> LIST;
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec insertion
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* primitive operations for the insertion-sort algorithm */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     list,ordelem;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     list-of-first:list-->list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     insert:list list-->list
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     simple-insert:list-->list;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 2425 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     list insertion
(OUT)    *** ILLEGAL COMMAND ***
(OUT)
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list insertion
(OUT)    spec    INSERTION /* PRIMITIVE OPERATIONS FOR THE INSERTION-SORT ALGORITHM */;
(OUT)    use     LIST,ORDELEM;
(OUT)    sorts
(OUT)    ops     LIST-OF-FIRST: LIST  --> LIST
(OUT)            INSERT: LIST LIST  --> LIST
(OUT)            SIMPLE-INSERT: LIST  --> LIST;
(OUT)    ENTER CMD:
(OUT)
(IN)     emap
(OUT)    ENTER SOURCE OF MAP :
(OUT)
(IN)     slots
(OUT)    ENTER NAME OF MAP :
```

```
(OUT)
(IN)     (-slots-%>)
(OUT)    ENTER TARGET OF MAP :
(OUT)
(IN)     insertion
(OUT)    ENTER USE OF MAP :
(OUT)
(IN)     nil
(OUT)    ENTER SORTMAP OF MAP :
(OUT)
(IN)     nil
(OUT)    ENTER OPMAP OF MAP :
(OUT)
(IN)     ((simple-sort . simple-insert)(combine . insert)(part1 . list-of-first)(part2 . list-of-first))
(OUT)    THE FOLLOWING POINTER WAS GENERATED : E1
(OUT)    DO YOU WANT TO ENTER MAPS AGAIN ?  REPLY (Y/N)
(OUT)
(IN)     y
(OUT)    ENTER SOURCE OF MAP :
(OUT)
(IN)     ordelem
(OUT)    ENTER NAME OF MAP :
(OUT)
(IN)     (-ordelm-%>)
(OUT)    ENTER TARGET OF MAP :
(OUT)
(IN)     natord
(OUT)    ENTER USE OF MAP :
(OUT)
(IN)     nil
(OUT)    ENTER SORTMAP OF MAP :
(OUT)
(IN)     ((elem . nat))
(OUT)    ENTER OPMAP OF MAP :
(OUT)
(IN)     ((eq-elem . eq-nat)(_%<%%=_ . _%<%%=_))
(OUT)    THE FOLLOWING POINTER WAS GENERATED : E2
(OUT)    DO YOU WANT TO ENTER MAPS AGAIN ?  REPLY (Y/N)
(OUT)
(IN)     n
(OUT)    ENTER CMD:
(OUT)
(IN)     lmaptab
(OUT)    THE MAPTABLE CONTAINS THE FOLLOWING MAPS
(OUT)    POINTER          MAP
(OUT)    E1         SLOTS -SLOTS-> INSERTION
(OUT)    E2         ORDELEM -ORDELM-> NATORD
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec insertion-sort
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* the insertion-sort-algorithm */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
```

```
(IN)     alg{slots -slots-> insertion}
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     alg{slots -slots-> insertion, slots -slots-> insertion}
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  }
(OUT)    *** IN THE SPECTERM BEGINNING WITH ALG THE SOURCE OF THE FOLLOWING MAP IS ACTUALIZED TWICE ***
(OUT)    SLOTS -SLOTS-> INSERTION
(OUT)    ENTER MAP OR }
(OUT)
(IN)     alg{slots -slots-> insertion}
(OUT)    *** THE FOLLOWING MAP IN THE USE-CLAUSE IS INCOMPLETE ***
(OUT)     ALG{SLOTS -SLOTS-> INSERTION}
(OUT)    ENTER MAP OR }
(OUT)
(IN)     }
(OUT)    *** THE FOLLOWING SPECTERM OCCURS ALREADY IN THE USE-CLAUSE ***
(OUT)    ALG{SLOTS -SLOTS-> INSERTION}
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 5674 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list insertion-sort
(OUT)    spec    INSERTION-SORT /* THE INSERTION-SORT-ALGORITHM */;
(OUT)    use     ALG{SLOTS -SLOTS-> INSERTION};
(OUT)    sorts
(OUT)    ops
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec insertion-sort-nat;
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     list{ordlem -ordelem-> natord}
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  }
(OUT)    *** IN THE SPECTERM BEGINNING WITH LIST THE SOURCE OF THE FOLLOWING MAP IS  NOT ALLOWED AS FORMAL PARAMETER ***
(OUT)    ORDLEM -ORDELEM-> NATORD
(OUT)    ENTER MAP OR }
```

```
(OUT)
(IN)     list{ordelem -ordelem-> natord}
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  }
(OUT)    *** THE FOLLOWING MAP IN THE SPECTERM BEGINNING WITH LIST ISN'T DEFINED ***
(OUT)    ORDELEM -ORDELEM-> NATORD
(OUT)    ENTER MAP OR }
(OUT)
(IN)     ordelem -ordelem-> natord}
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  }
(OUT)    *** THE FOLLOWING MAP IN THE SPECTERM BEGINNING WITH LIST ISN'T DEFINED ***
(OUT)    ORDELEM -ORDELEM-> NATORD
(OUT)    ENTER MAP OR }
(OUT)
(IN)     abort
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 2784 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     lmaptab
(OUT)    THE MAPTABLE CONTAINS THE FOLLOWING MAPS
(OUT)    POINTER          MAP
(OUT)    E1         SLOTS -SLOTS-> INSERTION
(OUT)    E2         ORDELEM -ORDELM-> NATORD
(OUT)    ENTER CMD:
(OUT)
(IN)     input spec insertion-sort-nat
(OUT)
(OUT)    ******   I N P U T L E V E L   FOR   SPECS  ******
(OUT)    ****** START WITH INPUT AT THE FOLLOWING CLAUSE ******
(OUT)    USE-CLAUSE
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     list{ordelem -ordelm-> natord}
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     insertion{ordelem -ordelm-> natord}
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     alg(slots -slots-> insertion}{ordelem -ordelm-> natord}
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  -SLOTS- > INSERTION}{ORDELEM -ORDEL . . .
(OUT)    *** THE FOLLOWING SPECTERM DOESN'T EXIST IN THE ENVIRONMENT ***
(OUT)    ALG(SLOTS
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     alg{slots -slots-> insertion}{ordelem -ordelm-> natord}
(OUT)    ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     end
```

```
(OUT)    ******      E N D   I N P U T       ******
(OUT)    CPU TIME USED : 13067 ms.
(OUT)
(OUT)    ******  E D I T O R L E V E L   FOR SPECS     ******
(OUT)    ENTER EDITOR-CMD:
(OUT)
(IN)     end
(OUT)    ENTER CMD:
(OUT)
(IN)     list insertion-sort-nat
(OUT)    spec   INSERTION-SORT-NAT;
(OUT)    use    LIST{ORDELEM -ORDELM-> NATORD},INSERTION{ORDELEM -ORDELM-> NATORD},
(OUT)           ALG{SLOTS -SLOTS-> INSERTION}{ORDELEM -ORDELM-> NATORD};
(OUT)    sorts
(OUT)    ops
(OUT)    ENTER CMD:
(OUT)
(IN)     li insertion-sort-nat
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  INSERTION-SORT-NAT  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from NATORD :
(OUT)       NAT
(OUT)    operations from NATORD :
(OUT)       _<%=_: NAT NAT --> BOOL
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)       NULL:  --> NAT
(OUT)       SUCC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)       _-_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    sorts from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST
(OUT)    operations from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       EMPTY:  --> LIST
(OUT)       PUT: NAT LIST --> LIST
(OUT)       FIRST: LIST --> NAT
(OUT)       REST: LIST --> LIST
(OUT)       APPEND: LIST LIST --> LIST
(OUT)       EMPTY?: LIST --> BOOL
(OUT)       SIMPLE?: LIST --> BOOL
(OUT)       IN?: NAT LIST --> BOOL
(OUT)
(OUT)    no sorts from INSERTION{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from INSERTION{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST-OF-FIRST: LIST --> LIST
(OUT)       INSERT: LIST LIST --> LIST
```

```
(OUT)       SIMPLE-INSERT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG{SLOTS -SLOTS-> INSERTION}{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from ALG{SLOTS -SLOTS-> INSERTION}{ORDELEM -ORDELM-> NATORD} :
(OUT)       SORT: LIST --> LIST
(OUT)    000045 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    no sorts from INSERTION-SORT-NAT
(OUT)    no operations from INSERTION-SORT-NAT
(OUT)
(OUT)    ENTER CMD:
(OUT)
(IN)     li insertion-sort
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  INSERTION-SORT  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from ORDELEM :
(OUT)       ELEM
(OUT)    operations from ORDELEM :
(OUT)       _<%=_: ELEM ELEM --> BOOL
(OUT)       EQ-ELEM: ELEM ELEM --> BOOL
(OUT)
(OUT)    sorts from LIST :
(OUT)       LIST
(OUT)    operations from LIST :
(OUT)       EMPTY:  --> LIST
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)       PUT: ELEM LIST --> LIST
(OUT)       FIRST: LIST --> ELEM
(OUT)       REST: LIST --> LIST
(OUT)       APPEND: LIST LIST --> LIST
(OUT)       EMPTY?: LIST --> BOOL
(OUT)       SIMPLE?: LIST --> BOOL
(OUT)       IN?: ELEM LIST --> BOOL
(OUT)
(OUT)    no sorts from INSERTION
(OUT)    operations from INSERTION :
(OUT)       LIST-OF-FIRST: LIST --> LIST
(OUT)       INSERT: LIST LIST --> LIST
(OUT)       SIMPLE-INSERT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG{SLOTS -SLOTS-> INSERTION}
(OUT)    operations from ALG{SLOTS -SLOTS-> INSERTION} :
(OUT)       SORT: LIST --> LIST
(OUT)
(OUT)    no sorts from INSERTION-SORT
(OUT)    no operations from INSERTION-SORT
(OUT)
(OUT)    ENTER CMD:
```

```
(OUT)
(IN)     list insertion-sort(natord)
(OUT)    COMMAND NIL NOT FOUND. TRY AGAIN OR ENTER HELP.
(OUT)    READY:
(OUT)
(IN)     open test
(OUT)    FILE OPENED !
(OUT)    MESSAGE(S) FOR WS.TEST :
(OUT)
(OUT)    ENTER CMD:
(OUT)
(IN)     list insertion-sort(natord)
(OUT)    spec    INSERTION-SORT(NATORD) /* APPLICATION OF INSERTION-SORT TO NATORD */;
(OUT)    use     INSERTION-SORT{ORDELEM -ORDELM-> NATORD};
(OUT)    sorts
(OUT)    ops
(OUT)    ENTER CMD:
(OUT)
(IN)     li insertion-sort(natord)
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  INSERTION-SORT(NATORD)  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from NATORD :
(OUT)       NAT
(OUT)    operations from NATORD :
(OUT)       _<%=_: NAT NAT --> BOOL
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)       NULL:  --> NAT
(OUT)       SUCC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)       _-_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)
(OUT)    sorts from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST
(OUT)    operations from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       EMPTY:  --> LIST
(OUT)       PUT: NAT LIST --> LIST
(OUT)       FIRST: LIST --> NAT
(OUT)       REST: LIST --> LIST
(OUT)       APPEND: LIST LIST --> LIST
(OUT)       EMPTY?: LIST --> BOOL
(OUT)       SIMPLE?: LIST --> BOOL
(OUT)       IN?: NAT LIST --> BOOL
(OUT)
(OUT)    no sorts from INSERTION{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from INSERTION{ORDELEM -ORDELM-> NATORD} :
```

```
(OUT)       LIST-OF-FIRST: LIST --> LIST
(OUT)       INSERT: LIST LIST --> LIST
(OUT)       SIMPLE-INSERT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG{SLOTS -SLOTS-> INSERTION}{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from ALG{SLOTS -SLOTS-> INSERTION}{ORDELEM -ORDELM-> NATORD} :
(OUT)       SORT: LIST --> LIST
(OUT)    000045 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)
(OUT)    no sorts from INSERTION-SORT{ORDELEM -ORDELM-> NATORD}
(OUT)    no operations from INSERTION-SORT{ORDELEM -ORDELM-> NATORD}
(OUT)
(OUT)    no sorts from INSERTION-SORT(NATORD)
(OUT)    no operations from INSERTION-SORT(NATORD)
(OUT)
(OUT)    ENTER CMD:
(OUT)
(IN)     list alg(insertion(natord))
(OUT)    spec    ALG(INSERTION(NATORD)) /* APPLICATION OF ALG TO INSERTION OF NATORD */;
(OUT)    use     ALG{SLOTS -SLOTS-> INSERTION --> INSERTION{ORDELEM -ORDELM-> NATORD}};
(OUT)    sorts
(OUT)    ops
(OUT)    ENTER CMD:
(OUT)
(IN)     li alg(insertion(natord))
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  ALG(INSERTION(NATORD))  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from NATORD :
(OUT)       NAT
(OUT)    operations from NATORD :
(OUT)       _<%=_: NAT NAT --> BOOL
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)       NULL:  --> NAT
(OUT)       SUCC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)       _-_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)
(OUT)    sorts from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST
(OUT)    operations from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       EMPTY:  --> LIST
(OUT)       PUT: NAT LIST --> LIST
(OUT)       FIRST: LIST --> NAT
(OUT)       REST: LIST --> LIST
```

```
(OUT)       APPEND: LIST LIST --> LIST
(OUT)       EMPTY?: LIST --> BOOL
(OUT)       SIMPLE?: LIST --> BOOL
(OUT)       IN?: NAT LIST --> BOOL
(OUT)
(OUT)    no sorts from INSERTION{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from INSERTION{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST-OF-FIRST: LIST --> LIST
(OUT)       INSERT: LIST LIST --> LIST
(OUT)       SIMPLE-INSERT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG{SLOTS -SLOTS-> INSERTION --> INSERTION{ORDELEM -ORDELM-> NATORD}}
(OUT)    000043 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)    operations from ALG{SLOTS -SLOTS-> INSERTION --> INSERTION{ORDELEM -ORDELM-> NATORD}} :
(OUT)       SORT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG(INSERTION(NATORD))
(OUT)    no operations from ALG(INSERTION(NATORD))
(OUT)
(OUT)    ENTER CMD:
(OUT)
(IN)     list alg(natord(insertion(natord)))
(OUT)    ILLEGAL SPECIFICATION NAME. COMMAND BROKEN.
(OUT)    ENTER CMD:
(OUT)
(IN)     list alg(natord)(insertion(natord)))
(OUT)    ILLEGAL SPECIFICATION NAME. COMMAND BROKEN.
(OUT)    ENTER CMD:
(OUT)
(IN)     list alg(natord)(insertion(natord))
(OUT)    spec   ALG(NATORD)(INSERTION(NATORD))
(OUT)            /* APPLICATION OF ALG OF NATORD TO INSERTION OF NATORD */;
(OUT)    use
(OUT)           ALG{ORDELEM -ORDELM-> NATORD}
(OUT)              {SLOTS{ORDELEM -ORDELM-> NATORD} -NATORDSLOTS-> INSERTION{ORDELEM
(OUT)                -ORDELM-> NATORD}};
(OUT)    sorts
(OUT)    ops
(OUT)    ENTER CMD:
(OUT)
(IN)     li alg(natord)(insertion(natord))
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  ALG(NATORD)(INSERTION(NATORD))  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from NATORD :
(OUT)       NAT
(OUT)    operations from NATORD :
(OUT)       _<%=_: NAT NAT --> BOOL
```

```
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)       NULL:  --> NAT
(OUT)       SUCC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)       _-_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)
(OUT)    sorts from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST
(OUT)    operations from LIST{ORDELEM -ORDELM-> NATORD} :
(OUT)       EMPTY:  --> LIST
(OUT)       PUT: NAT LIST --> LIST
(OUT)       FIRST: LIST --> NAT
(OUT)       REST: LIST --> LIST
(OUT)       APPEND: LIST LIST --> LIST
(OUT)       EMPTY?: LIST --> BOOL
(OUT)       SIMPLE?: LIST --> BOOL
(OUT)       IN?: NAT LIST --> BOOL
(OUT)
(OUT)    no sorts from INSERTION{ORDELEM -ORDELM-> NATORD}
(OUT)    operations from INSERTION{ORDELEM -ORDELM-> NATORD} :
(OUT)       LIST-OF-FIRST: LIST --> LIST
(OUT)       INSERT: LIST LIST --> LIST
(OUT)       SIMPLE-INSERT: LIST --> LIST
(OUT)
(OUT)
(OUT)    no sorts from ALG{ORDELEM -ORDELM-> NATORD}{SLOTS{ORDELEM -ORDELM-> NATORD} -NATORDSLOTS-> INSERTION{ORDELEM -ORDEL
(  )     M-> NA
(OUT)    000044 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)    TORD}}
(OUT)
(OUT)    operations from ALG{ORDELEM -ORDELM-> NATORD}{SLOTS{ORDELEM -ORDELM-> NATORD} -NATORDSLOTS-> INSERTION{ORDELEM -ORD
(  )     ELM->
(OUT)    NATORD}} :
(OUT)       SORT: LIST --> LIST
(OUT)
(OUT)    no sorts from ALG(NATORD)(INSERTION(NATORD))
(OUT)    no operations from ALG(NATORD)(INSERTION(NATORD))
(OUT)
(OUT)    ENTER CMD:
(OUT)
(IN)     close
(OUT)    %  D800 ERASE FILE LISP.DATA.WS.MAPTABLE.00
(OUT)    READY:
(OUT)
(IN)     end
```

7. Literaturverzeichnis

- [Kr 79] Kreowski, H.-J. : Algebra fUr Informatiker, Skript zur gleichnamigen Vorlesung im WS 78/79, TU Berlin,

- [Ra 79] Raulefs, P. : EinfUhrung in die Theorie der Datenstrukturen, Vorlesungsnotizen zur gleichnamigen Vorlesung im SS 1979, Universität Bonn, Fachbereich Informatik, 1979

- [BES 81] Bläsius, K., Eisinger, N., Siekmann, J., Smolka, G., Herold, A., Walther, C. : The Markgraf Cal Refutation Procedure, Proc., 7th IJCAI, 1981

- [BV 83a] Beierle, Ch., Voß, A. : Canonical Term Functors and Parameterization-by-use for the Specification of Abstract Data Types. SEKI-Projekt, Memo SEKI-83-07, Universität Kaiserslautern, Fachbereich Informatik, May 1983

- [BV 83b] Beierle,Ch., Voß, A., : Parameterization-by-use for hierarchically structured objects. SEKI-Projekt, Memo SEKI-83-08, Universität Kaiserslautern, Fachbereich Informatik, May 1983

- [BGV 83] Beierle, Ch., Gerlach, M., Voß, A. : Parameterization without Parameters-in : The History of a Hierarchy of Specifications. SEKI-Projekt, Memo SEKI-83-09, Universität Kaiserslautern, Fachbereich Informatik, September 1983

- [KRST 83] Kücke, R., Rome, E., Sommer, W., Thomas, Ch. : Das SPEC-System (SPESY) : Benutzerhandbuch, SEKI-Projekt, Universität Kaiserslautern, Fachbereich Informatik, 1983

- [Ge 83] Gerlach, M. : A Second-Order Matching Procedure for the Practical Use in a Program Transformation System. SEKI-Projekt, Memo SEKI-83-13, Universität Kaiserslautern, Fachbereich Informatik, September 1983

- [Tho 84] Thomas, Ch. : RRLab- Rewrite Rule Labor. Entwurf, Spezifikation und Implementierung eines Softwarewerkzeuges zur Erzeugung und Vervollständigung von Rewrite-Rule Systemen. SEKI-Projekt, Memo SEKI-84-01, Universität

Kaiserslautern, Fachbereich Informatik, 1984

- [Som 84] Sommer, W. : SPESY - ein interaktives System zur Unterstützung integrierter Programmspezifikation und Programmverifikation, SEKI-Projekt, Memo SEKI-84-02, Universität Kaiserslautern, Fachbereich Informatik, 1984

- [Epp] Epp, B. : Interlisp-Programmierhandbuch, Institut für deutsche Sprache

- [Int] SIEMENS-INTERLISP, Benutzerhandbuch Version 4.0, August 1980