SEKI-PROJEKT

SEKI
MEMO

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

Ein interaktives und syntaxorientiertes Eingabesystem für algebraische und algorithmische Spezifikationen

Horst Lichter

MEMO SEKI-85-01

1. Einleitung ..... Seite 1

1.1. Das interaktive Programmentwicklungs- und Verifikationssystem..... Seite 1
1.2. Die in der Arbeit [Li 85] implementierte Komponente ..... Seite 6

2. Normalform und Interface eines Specterms ..... Seite 8

2.1. Die Normalform eines Specterms ..... Seite 8
2.2. Das Exported Interface eines Specterms ..... Seite 20

3. Benutzeranleitung ..... Seite 32

3.1. Allgemeine Beschreibung des Eingabedialogs .. Seite 32
3.2. Der Eingabedialog am Bildschirm ..... Seite 39
3.3. Vordefinierte Teile einer Spezifikation ..... Seite 44
3.3.1. Die Spezifikation BOOL ..... Seite 45
3.3.2. Die EQ-Operation ..... Seite 46
3.3.3. Die ERROR-Konstante ..... Seite 48
3.3.4. Das charakteristische Prädikat ..... Seite 48
3.3.5. Die Konstruktordefinition ..... Seite 50
3.4. OK-Bedingungen ..... Seite 52
3.4.1. Begriffliche Grundlagen ..... Seite 53
3.4.2. Die OK-Bedingungen ..... Seite 56
3.4.2.1. Specterme und Mapterme ..... Seite 56
3.4.2.2. Opheader und Operationsdefinitionen . Seite 58
3.4.2.3. Die Klauseln des spec-headers ..... Seite 62
3.4.2.4. Die Klauseln des spec-bodies ..... Seite 64
3.5. ASPIK-Syntax ..... Seite 67
3.5.1. Notation ..... Seite 67
3.5.2. Die ASPIK-Syntax ..... Seite 68
3.5.2.1. Die Klauseln des spec-headers ..... Seite 68
3.5.2.2. Die Klauseln des spec-bodies ..... Seite 70
3.5.2.3. Specterme und Mapterme ..... Seite 73
3.5.2.4. Opheader und Operationsdefinitionen . Seite 74
3.5.2.5. Terme ..... Seite 76
3.5.2.6. Properties ..... Seite 77
3.5.2.7. Terminal-Regeln ..... Seite 79
3.5.3. Lexikalische Konventionen ..... Seite 82

4. Literaturverzeichnis ..... Seite 84

5. SPESY-Sitzungsprotokolle ..... Seite 86

5.1. Sitzungsprotokolle des Teil-Eingabesystems .. von Hans Matheis ..... Seite 87
5.2. Sitzungsprotokolle des Gesamt-Eingabesystems Seite 105

1. Einleitung

1.1. Das interaktive Programmentwicklungs- und Verifikationssystem

SPESY ist ein Entwicklungssystem für algebraische Spezifikationen [BV 81/84]. Es wurde aus einer Vorgängerversion ([KRST 83], [Som 84]) entwickelt, ist in INTERLISP implementiert und ist auf einer Siemens-Rechenanlage installiert [Sch 85]. SPESY ist die zentrale Komponente eines integrierten Programmentwicklungs- und Programmverifikationssystems [BGGORV 83]. 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], [BV 85]) ausdrücken. 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 nicht nur die Definition von Spezifikationen sondern auch von soge-

nannten Maps zur Beschreibung von Verfeinerungs- und Implementierungsbeziehungen. Aus diesem Grund ist ASPIK sogar als Spezifikationsentwicklungssprache anzusehen.

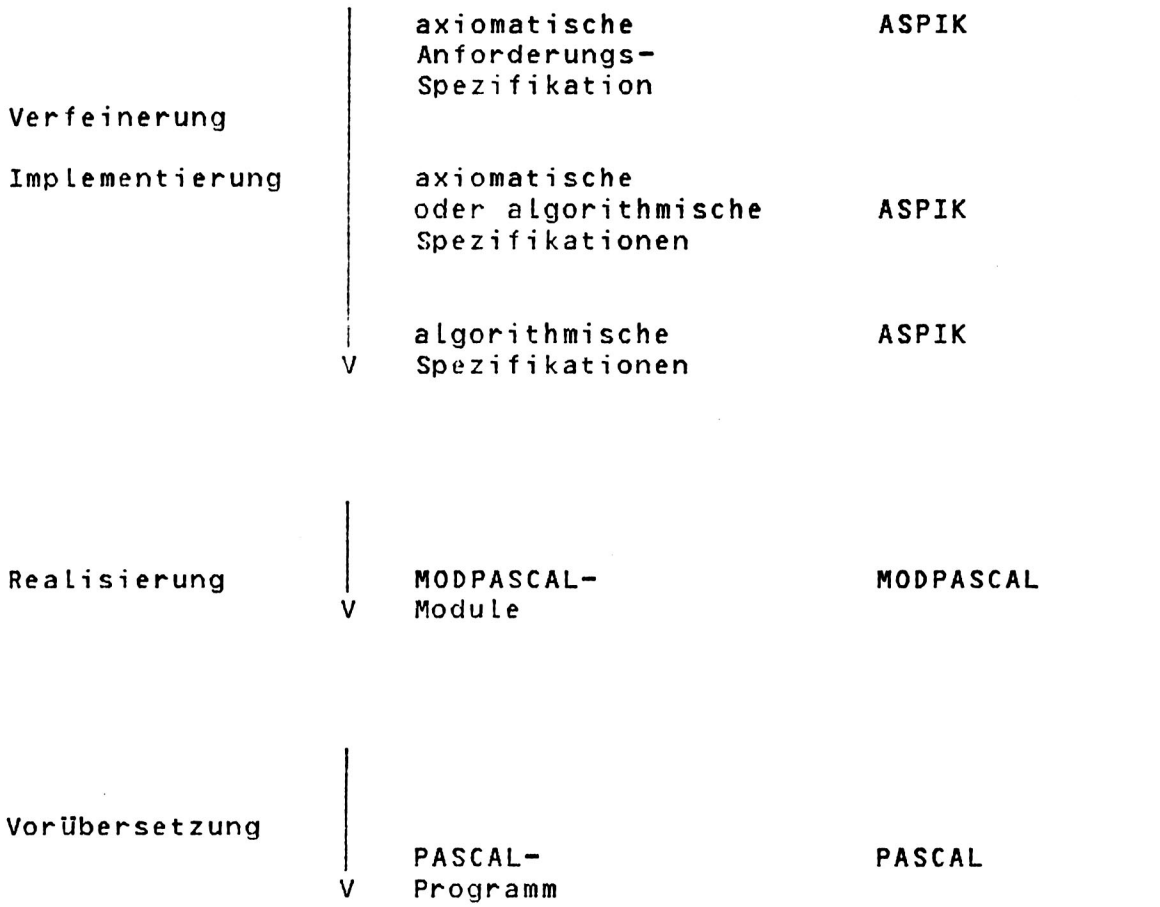


Abb.1 : Schrittweise Programmentwicklung

Die beiden Strukturierungsmechanismen von ASPIK wurden bereits angedeutet:
Die Use-Beziehung für Spezifikationen und Maps 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 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, Maps, 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.

Neben interaktiven Systemen zum Eingeben und Editieren von ASPIK-Objekten stellt SPESY folgende Werkzeuge zur Verfügung:

- einen symbolischen Interpretierer zum Testen von konstruktiven Spezifikationen (zur Zeit noch nicht auf die neue Sprachversion umgestellt) [KRST 83].

- eine allgemeine Beweiserschnittstelle, über die Beweisaufgaben gestellt, ihre Lösung entgegengenommen und in SPESY verarbeitet werden können. Zur Zeit sind zwei Beweissysteme über diese Schnittstelle ansprechbar :
  Der Karlsruher Resolutionsbeweiser "Markgraf Karl Refutation Procedure" [BES 81] und ein in Bonn entwickeltes Termersetzungssystem [Tho 84], um Konsistenzüberprüfungen durchzuführen.

- ein Subsystem zur Realisierung von konstruktiven ASPIK-Spezifikationen durch MODPASCAL bzw. PASCAL-Programme [OLT 85].

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

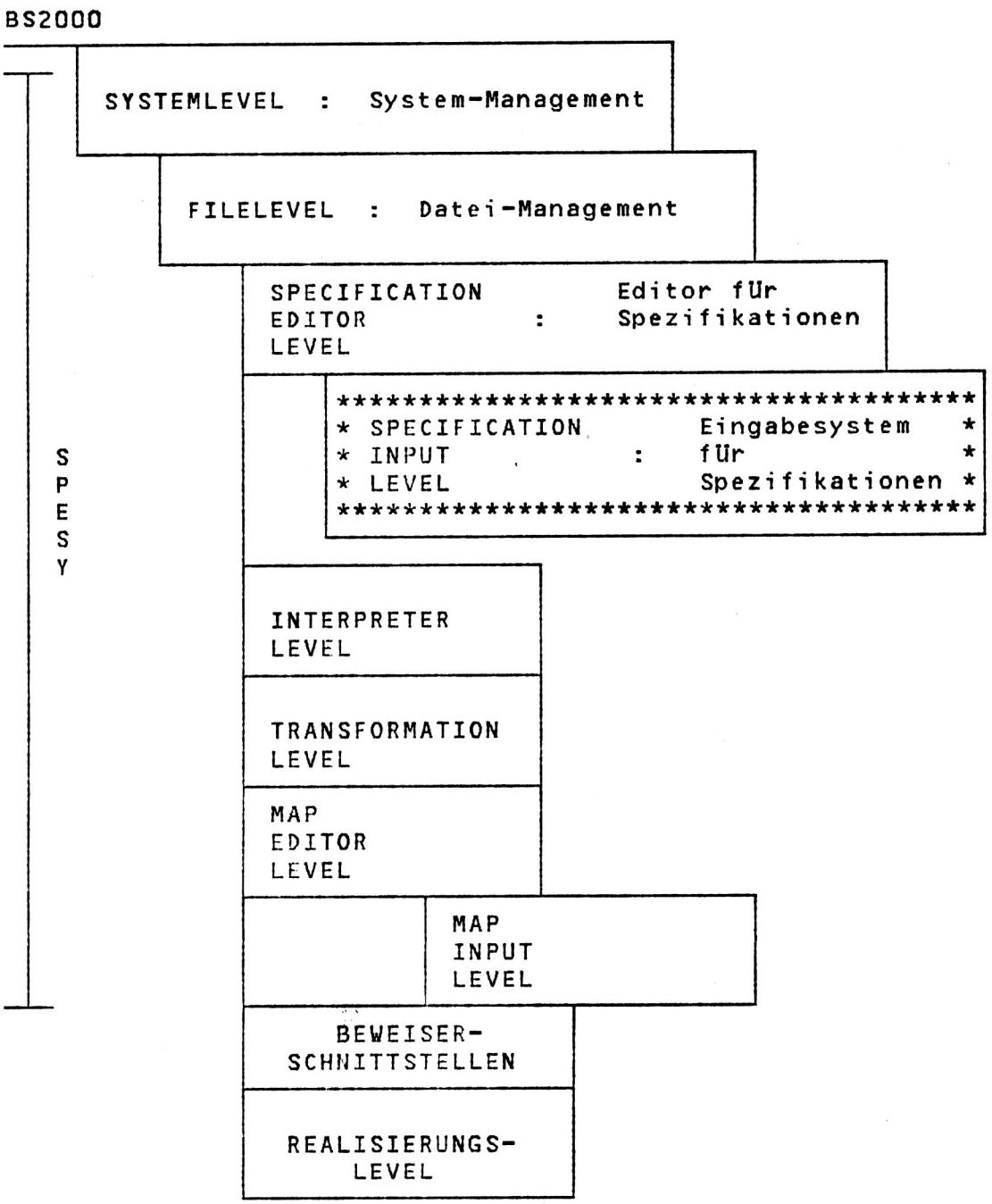


Abb.2 Grobstruktur von SPESY und
Einbettung des Eingabesystems für Spezifikationen

1.2. Die in der Arbeit [Li 85] implementierte Komponente

Das Eingabesystem für Spezifikationen - das Thema der Arbeit [Li 85] - 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 (siehe Abb.3 S.34) 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 linken Teil sowie in einen leeren rechten 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) beinhaltet 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 [Li 85] führt die Diplomarbeit von Hans Matheis [Mat 85a] fort. Sie gliedert sich in drei Teile. Der Band I ist als Benutzer-Einführung zu verstehen und umfaßt die Kapitel 1 bis 5. Band II besteht aus den Kapiteln 6 bis 8. In ihm sind das Modulkonzept, die Datenstrukturen, die Dokumentation aller implementierten LISP-Funktionen, sowie die Schnittstellen zu anderen Komponenten in SPESY zu finden. Band III beinhaltet die erstellten Programmlistings.

In Kapitel 2 sind Algorithmen für die Normalisierung und die interface-Bestimmung in abstrakter Form angegeben.

In Kapitel 3 befindet sich das Benutzermanual mit der Dialogbeschreibung des Eingabesystems, der konkreten ASPIK-Syntax und den kontextsensitiven Tests.

Die benutzte Literatur ist in Kapitel 4 zusammengestellt.

Um einen Einblick in das Dialogverhalten zu vermitteln, sind in Kapitel 5 SPESY-Sitzungsprotokolle wiedergegeben.

In Kapitel 6 wird das Modulkonzept sowie die Modulstruktur von SPESY dargestellt. Weiterhin sind hier die benutzten Datenstrukturen und die Programmstruktur erklärt.

Die Dokumentation aller implementieren LISP-Funktionen ist in Kapitel 7 enthalten.

Kapitel 8 enthält die Schnittstellenbeschreibung zu anderen Komponenten in SPESY.

In Kapitel 9 sind die Programmlistings aufgeführt.

2. Normalform und Interface eines Specterms

2.1. Die Normalform eines Specterms

Das 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'

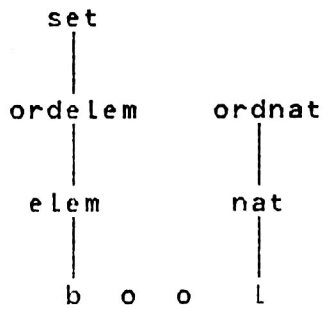
Der Normalisierungsalgorithmus NORM bestimmt die Normalform eines Specterms in vier Hauptschritten:

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 :



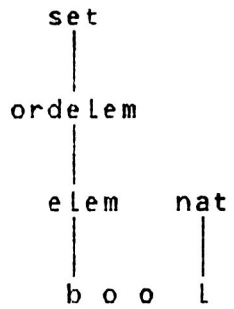
Erläuterung : die Use-Beziehung ist von oben nach unten zu lesen, d.h. set benutzt ordelem, ordelem benutzt elem, usw.

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 :

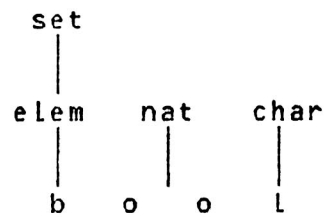


Zusätzlich sei die Abbildung elem --> nat definiert.
Dann ist die Instanz set{elem --> nat} zulässig, obwohl das sogenannte Between-Element ordelem nicht explizit aktualisiert worden ist. Aufgabe des Normalisierungsalgorithmus ist es, für solche Between-Elemente sogenannte Applikations-Abbildungen zu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



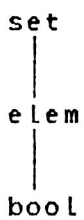
Zusätzlich seien die Abbildungen
elem --> nat und
nat --> char definiert.

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 map1 ... mapn)] =

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

        ptri = GET-MAP-PTR[NORM-MAP[mapi]] in

    REDUCE [COMPOSE [DIRECT [INSERT-USED-MAPS

                [(spect-nf ptr1 ... ptrn]]]]
```

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 ptr1 ... ptrn)] =

        let (ptr*1 ... ptr*K)

            = union [GET-USED-MAPS[ptr1], ...

                    ,GET-USED-MAPS[ptrn],

                    (ptr1 ... ptrn)]

        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_;-L,1 ... source_;-L,nL) in

let ptr-betw_L

  = GENERATE-MAP[betw_L, NORM1[(betw_L map_;-L,1 ... map_;-L,nL)],

                 (ptr_;-L,1, ... ,ptr_;-L,nL), 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 $param\text{-}set'_2$ =

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

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

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

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

$ptr_{1,i}$ if [$source_{1,i}$ = $source_K$] and notmember [$target_{1,i}$, param-set_2]

ptr'_K = $ptr_{2,j}$ if [$source_{2,j}$ = $source_K$] and notmember [$source_{2,j}$, param-set_1]

COMPOSE-MAP[$ptr_{1,i}$, $ptr_{2,j}$]
if [$source_K$ = $source_{1,i}$] and
[$target_{1,i}$ = $source_{2,j}$]

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]

= ($source_{i_1}$ ... $source_{i_j}$) in

REDUCE[(specid ptr_{i_1} ... ptr_{i_j})]

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 $spect_{i,K} \in$ 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$_{i,i1}$ überprüft, ob spect$_{i,i1}$ äquivalent ist zu einem Specterm spect aus der Ergebnisliste. Trifft dies nicht zu, dann wird spect$_{i,i1}$

	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$_i$, lst$_j$]	liefert TRUE, falls jedes Element der Liste lst$_i$ in der Liste lst$_j$ 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 SPECFROM 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 linken Teil sowie in einen leeren rechten 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. So kann eine ganze Klausel generiert werden (z.B. define-carriers-clause), ein Klauselelement oder auch nur ein Klauselschlüsselwort (siehe 5.2. S.109: ops-clause, S.115: define-ops-clause). 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 der Hauptknoten zurückgesetzt, sodaß der Benutzer den Fehler korrigieren kann. Diese Promptpositionen sind in Abbildung 4 durch PP dargestellt.

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 Fehler zuerst durch den ganzen Dialog bis zum Ende der Spezifikation gehen , um dann die Spezifikation mit dem Editor (allerdings ohne Unterstützung) korrigieren zu können.

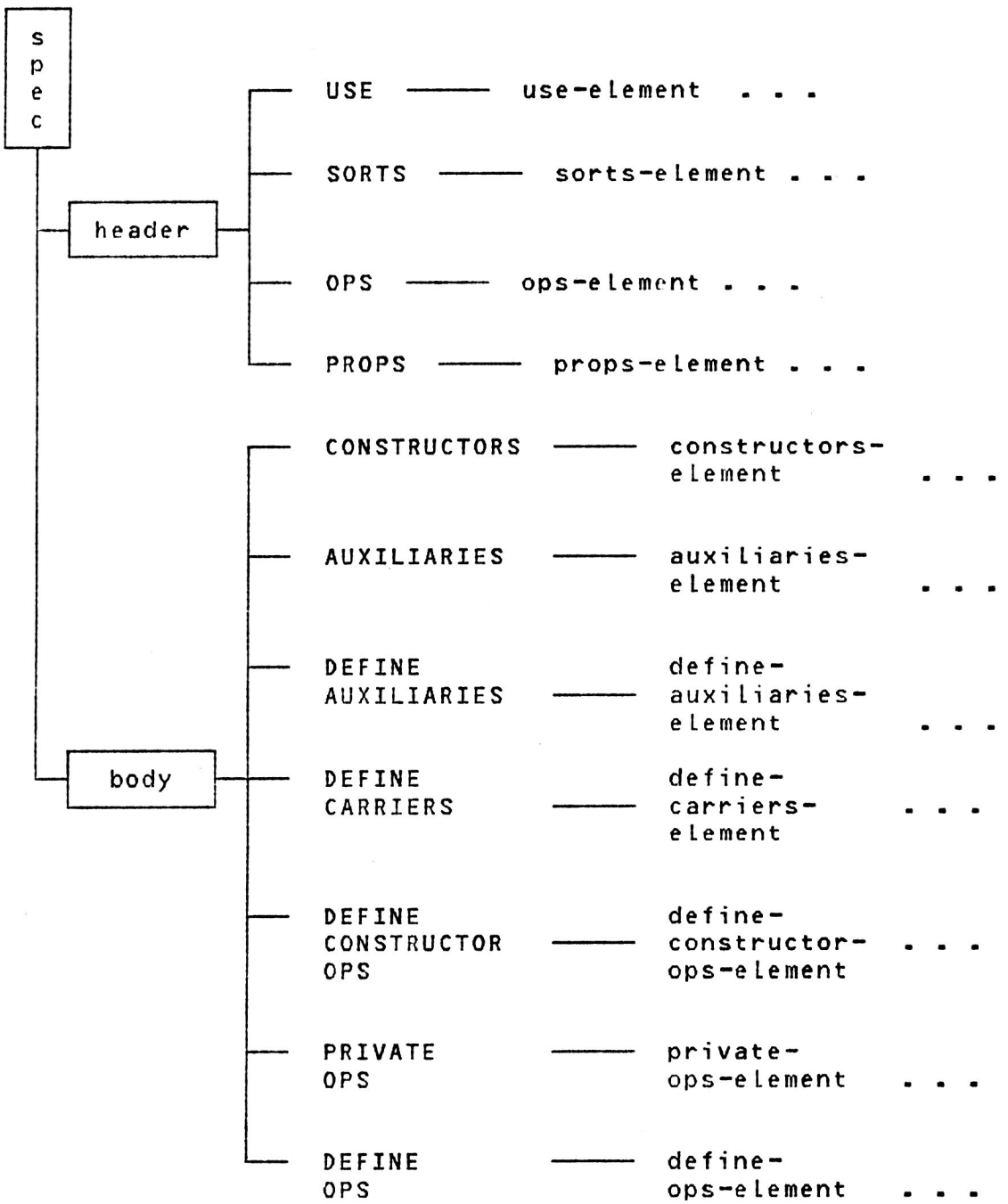


Abb.3 : Struktur des Syntaxbaumes einer Spezifikation. Hauptknoten (Klauseln und Klauselelemente) sind alle nicht umrahmten Knoten.

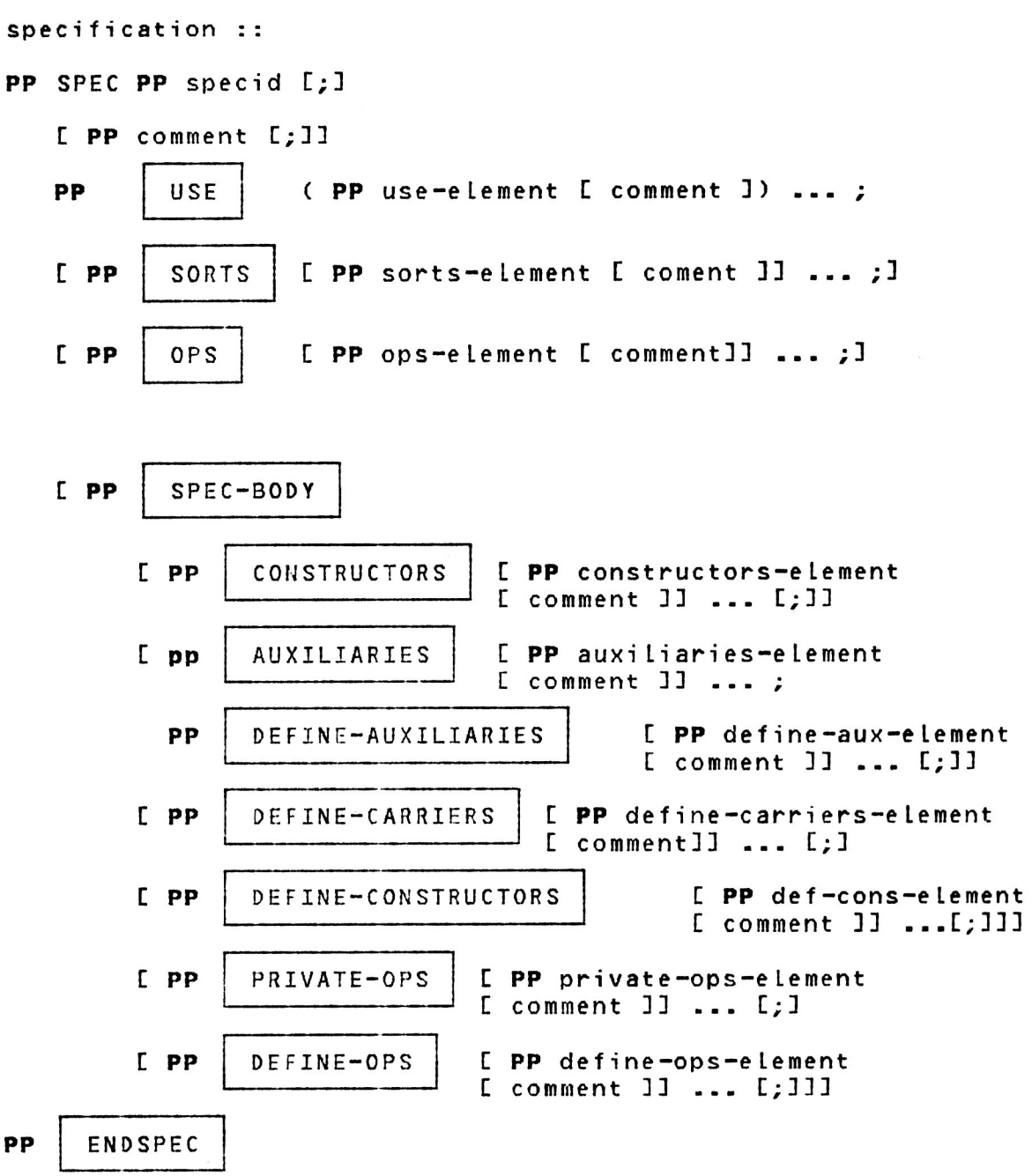
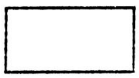


Abb.4 : Syntax einer Spezifikation bis zu den Klauselelementen ohne Berücksichtigung der Leerzeichen und Kommata.

Notation :

 kennzeichnet die automatisch generierbaren Teile einer Spezifikation.

GROSSCHREIBUNG kennzeichnet die Terminalsymbole.

:: trennt linke und rechte Seite einer Regel.

() sind Meta-Klammern.

[] unschließen optionele Teile.

PP kennzeichnet die Promptpositionen an den Hauptknoten.

In der neuen Version stehen dem Benutzer an jeder Promptposition außer an den Promptpositionen in den Operationsschemata der define-auxiliaries-clause und der define-ops-clause drei verschiedene Alternativen zur Verfügung (siehe 5.2. S.107,f) :

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)

Die zuletzt genannte Möglichkeit beendet den Eingabedialog, wonach sich der Benutzer im Editor befindet bzw. auf dem FILELEVEL, falls die Spezifikation vollständig ist.
An den Promptpositionen der define-carriers-clause und der define-constructor-ops-clause bestehen diese Möglichkeiten nicht, um den Programmieraufwand für diese Klauseln in einem vertretbaren Rahmen zu halten, da in der Regel größere Teile dieser Klauseln oder sogar die gesamten Klauseln automatisch generiert werden können.

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. Das Eingabesystem setzt dann bei der Promptposition auf, die auf das letzte korrekt eingegebene Klauselelement folgt, also wieder an einem Hauptknoten.

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 mit */ enden (siehe 5.2. S.111).
- 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 Pro-

mptposition (siehe 5.2. S.122,f).
- Die restriktiven Bezeichnerkonventionen der alten Version wurden weitgehend aufgehoben, sodaß der Benutzer möglichst frei seine Bezeichner wählen kann.

3.2. Der Eingabedialog am Bildschirm

Nachdem das Eingabesystem mit den dazugehörenden Moduln geladen ist, meldet es sich beim Benutzer mit:

******** I N P U T L E V E L F O R S P E C S ********

Hat der Benutzer beim INPUT-Kommando auf dem FILELEVEL einem neuen Spezifikationsnamen angegeben, so positioniert das System auf die comment-clause und promptet:

ENTER COMMENT OR ;

Soll dagegen eine bereits bestehende Spezifikation erweitert werden, so ist das System, falls alle notwendigen Vorbedingungen erfüllt sind, auf die letzte Klausel des gefüllten Teilbaumes der Spezifikation positioniert. Ist diese Klausel bereits vollständig gefüllt, so wird auf die nächstfolgende Klausel (siehe Abb.5) positioniert. Das System promptet anschließend (siehe 5.2. S.109: ops-clause ,S.115: define-ops-clause) :

ENTER <clause-element> COMMENT OR ;

Der Zusatz " OR ; " entfällt bei allen Klausel in der ersten Promptmeldung, die mindestens ein Klauselelement enthalten müssen, so z. B. bei der use-clause.

Ist eine Spezifikation bereits vollständig und korrekt, wird dies dem Benutzer mitgeteilt und zum FILELEVEL zurückgesprungen (siehe 5.2. S.119 unten). Kann das Eingabesystem keinen korrekten "Aufsetzpunkt" in einer bestehenden Spezifikation finden, weil diese z B. nicht korrekte Klauseln enthält, wird eine entsprechende Fehlermeldung ausgegeben und zum Editor für Spezifikatonen zurückgesprungen.

Wurde eine korrekte Startklausel vom Eingabesystem gefunden, so können solange Klauselelemente für die noch ausstehenden Klauseln eingegeben werden, bis entweder ein END-Kommando eingegeben wurde oder das Ende der Spezifikation erreicht wird. Eine Klausel kann explizit durch die Eingabe eines Semikolons, eines END-Kommandos oder durch ein Klauselschlüsselwort abgeschlossen werden. Die Klauseln, die eine feste Menge von Elementen enthalten müssen, werden nach Eingabe des letzten Klauselelementes vom System beendet (siehe 5.2. S.123: define-auxiliaries-clause). Die Klauselelemente sind bei der define-auxiliaries-clause durch die Menge der auxiliaries, bei der define-carriers-clause durch die Menge der neuen Sorten, bei der define-constructor-ops-clause durch die Menge der Konstruktoren und bei der define-ops-clause durch die Menge der noch nicht als auxiliary oder Konstruktor definierten Operationen vorgegeben. Die Reihenfolge, in der die Klauseln einer Spezifikation

durch das Eingabesystem gepromptet werden und welche Klauseln wann übersprungen werden, weil sie leer sein müssen, ist in Abbildung 5 dargestellt.

Bei der Eingabe einer Spezifikation muß darauf geachtet werden, daß die Zeilen mit Blanks aufgefüllt sind und kein end-of-line Zeichen bzw. kein end-of-text Zeichen darin enthalten ist, wenn die Länge eines Klauselelementes eine oder mehrere Zeilen überschreitet oder ein Klauselelement strukturiert eingegeben werden soll. Außerdem muß am Anfang jeder Folgezeile der Eingabe mindestens ein Leerzeichen stehen.

An allen Promptpositionen außer an der Promptpositionen in den Operationsschemata der define-carriers-clause und der define-constructor-ops-clause hat der Benutzer die Möglichkeit, den Dialog mit dem END-Kommando zu beenden oder Help-Information mit dem Help-Kommando abzufragen. Deshalb muß an diesen Positionen darauf geachtet werden, daß, falls keines dieser Kommandos eingegeben werden soll, das erste Teilstück der folgenden Eingabe nicht mit einem der Wörter "HELP", "H", "?" oder "END" übereinstimmt. Um diesen Konflikt zu vermeiden, emfielt es sich, keine Variablen oder Operationsnamen zu bilden, die als Ganzes oder als Teilstück eines dieser Wörter enthalten. Sollte es aber trotzdem erforderlich sein, daß nach einer Promptposition in einem Operationsschema eines dieser Wörter stehen muß, so kann man diesen Konflikt umgehen, indem man über diese Promptposition hinaus Eingabe zu Verfügung stellt. Die Möglichkeit, den Dialog zu beenden oder Help-Information anzufordern, besteht nicht bei den Promptpositionen der Operationsschemata in der define-carriers-clause und in der define-constructor-ops-clause. Diese Einschränkung war notwendig, um den Programmieraufwand und den damit verbundenen Zeitaufwand für diese Klauseln in einem vertretbaren Rahmen zu halten.

Die in der define-carriers-clause einzugebenden charakteristischen Prädikate können, wenn der Benutzer dies will, exlizit eingegeben werden. Sie müssen dann der in 3.3.4. beschriebenen Form entsprechen (siehe 5.2. S.124,f: is-stack). Schließt der Benutzer die define-carriers-clause mit einem Semikolon ab, generiert das System die noch ausstehenden Prädikate, wie in 3.3.4. beschrieben. Hat der Benutzer irgendein Prädikat selbst definiert und stimmt diese Definition nicht mit einer systemgenerierten Form überein, so gilt die define-carriers-clause als vom Benutzer eingeben. Dies hat zur Folge, daß er in der define-constructor-ops-clause auch diejenigen Konstruktoren selbst definieren muß, die zu den Sorten der von ihm eingegebenen Prädikate gehören (siehe 3.3.5. und 5.2. S.125: empty,push). Andernfalls werden, wenn alle Prädikate vom System generiert wurden oder ihre Definition mit einer systemgenerierten Form übereinstimmt, auch alle Konstruktoren automatisch definiert. Das Eingabesystem teilt dem Benutzer sowohl in der define-carriers-clause als auch in der define-constructor-ops-

clause mit, welche Prädikate bzw. Konstruktordefinitionen generiert wurden (siehe 5.2. S.113: nat,suc,zero).
Wird das Ende einer Spezifikation erreicht oder der Dialog mit dem END-Kommando abgebrochen, so verabschiedet sich das Eingabesystem mit:

******** E N D I N P U T L E V E L ********

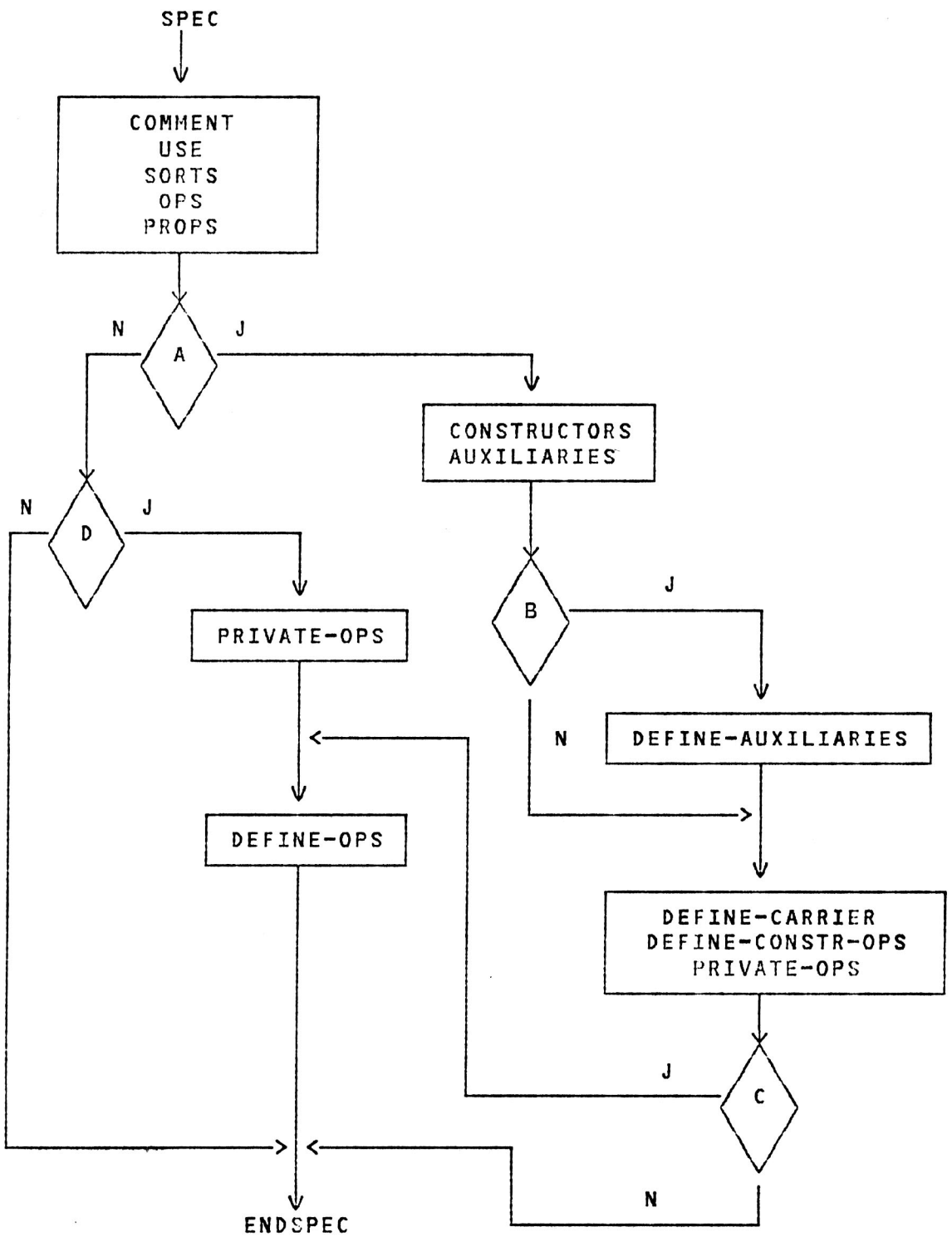


Abb5 : Reihenfolge, in der die Klauseln einer Spezifikation gepromptet werden.

Bedeutung der Abfragen :

A : Existieren Operationen in der ops-clause, die eine neue Sorte als Ergebnissorte haben?

B : Existieren auxiliaries?

C : Existieren Operationen aus der ops-clause und aus der private-ops-clause, die noch nicht als Konstruktor oder auxiliary definiert sind?

D : Existieren Operationen in der ops-clause?

3.3. Vordefinierte Teile einer Spezifikation

Einige der im folgenden beschriebenen vordefinierten Teile einer Spezifikation können vom Benutzer explizit eingegeben werden. Sie müssen dann der hier beschriebenen Form entsprechen. Werden sie nicht explizit eingegeben, so generiert das Eingabesystem diese Teile und meldet dies dem Benutzer. Zuerst folgt die Beschreibung der Spezifikation BOOL und anschließend die Beschreibung der einzelnen vordefinierten Teile einer Spezifikation ([BV 81/84], [BV 85]).

3.3.1. Die Spezifikation BOOL

Diese Spezifikation ist vordefiniert. Sie ist die Grundspezifikation jeder Hierachie von Spezifikationen und ist folgendermaßen definiert:

```
spec bool;

sorts bool;

ops true,false : --> bool,
    not : bool --> bool,
    _and_,_or_,eq-bool : bool bool --> bool;

spec-body

constructors true,false;

auxiliaries eq-bool : bool bool --> bool;

define-auxiliaries
    eq-bool(b1,b2) = case b1 is
                       * true : case b2 is
                                  * true : true
                                  otherwise : false
                                esac
                       * false : case b2 is
                                   * false : true
                                   otherwise : false
                                 esac
                     esac

define-carriers is-bool(b) = true;

define-constructor-ops true = * true
                       false = * false;

define-ops not(y) = case y is
                      * true : false
                      * false : true
                    esac
          a and b = case a is
                      * true : b
                      * false : false
                    esac
          a or b = case a is
                     * true : true
                     * false : b
                   esac

endspec
```

3.3.2. Die EQ-Operation

A) In einer losen Spezifikation, d.h. einer Spezifikation die keinen Spezifikationskörper hat, darf für jede neu eingeführte Sorte s der sorts-clause die Operation

```
eq-s : s s --> bool
```

in der ops-clause deklariert werden. Die Operation ist dann folgendermaßen vordefiniert:

```
all x,y : s eq-s(x,y) <==> x == y
```

B) In einer festen Spezifikation, d.h. einer Spezifikation mit einem Spezifikationskörper, ist die Operation eq-s für jede neue Sorte s erlaubt, die folgende Bedingung erfüllt:
- für alle Argumentsorten s_j der Konstruktoren von s ist ebenfalls eine Operation eq-s_j deklariert.

Erlaubte EQ-Operationen dürfen in der ops- und in der auxiliaries-clause explizit deklariert werden. Alle erlaubten, nicht explizit in der auxiliaries-clause deklarierten EQ-Operationen werden in der auxiliaries-clause implizit deklariert. Alle erlaubten EQ-Operationen werden in der define-auxiliaries-clause automatisch generiert (siehe 5.2. S.113,131: eq-nat).

Die Definition einer Operation eq-s ist nun konstruktiv wie folgt vorgegeben:

```
eq-s(x,y) = case x is
                * ctr_1 (x_1,1 ... x_1,n1) :
                  case y is
                       * ctr_1 (y_1,1 ... y_1,n1) :
                         true                                  if n_1=0
                         (...(eq-s_1,1(x_1,1,y_1,1) and ...
                          eq-s_1,n1(x_1,n1,y_1,n1) ...)  else
                       otherwise : false
                  esac
                .
                .
                * ctr_K (x_K,1 ... x_K,nK) :
                  case y is
                       * ctr_K (y_K,1 ... y_K,nK) :
                         true                                  if n_K=0
                         (...(eq-s_K,1(x_K,1,y_K,1) and ...
                          eq-s_K,nK(x_K,nK,y_K,nK) ...)  else
                       otherwise : false
                  esac
           esac
```

Dabei sind die $ctr_1 : s_{1,1} \ldots s_{1,n} \rightarrow s , \ldots , ctr_K : s_{K,1} \ldots s_{K,nK} \rightarrow s$ die Konstruktoren von s. Diese Definition wird in der define-auxilaries-clause für alle in der ops-clause und in der auxiliaries-clause implizit oder explizit enthaltenen EQ-Operationen erzeugt. Sie kann nicht vom Benutzer explixit eingegeben werden.

3.3.3. Die Error-Konstante

Für jede neu eingeführte Sorte s einer Spezifikation gibt es die Konstante

error-s

als exportierte Operation. Sie ist eine implizit deklarierte und definierte Operation und stellt das error-Element seiner Sorte dar. Sie kann nicht explizit vom Benutzer deklariert oder definiert werden.

3.3.4. Das charakteristische Prädikat

A) Das charakteristische Prädikat einer Sorte s, is-s(x), ist teilweise vordefiniert, wenn es vom Benutzer eingegeben wird (siehe 5.2. S.123,ff: is-stack) :

```
is-s(x) = PP case x is
                * ctr_1 (x_1,1 ... x_1,n1) :
                  [ if (...(not(is-s_1,L11(x_1,L11)) or ...
                       not(is-s_1,Lm1(x_1,Lm1)) ... )
                    then false
                    else ] OPSCHEME_1
                .
                .
                * ctr_K (x_K,1 ... x_K,nK) :
                  [ if (...(not(is-s_K,L11(x_K,L11)) or ...
                       not(is-s_K,LmK(x_K,LmK)) ... )
                    then false
                    else ] OPSCHEME_K
                [ otherwise : OPSCHEME_K + 1 ]
          esac
```

Dabei sind die $ctr_i : s_{i,1} \ldots s_{i,ni} \rightarrow s$,i=1...j, alle Konstruktoren von s und die Sorten $s_{i,L1} \ldots s_{i,Lmi}$, i=1...j, sind die neuen Argumentsorten dieser Konstruktoren. Der "if-then-else-Zweig" entfällt bei allen Konstruktoren, die keine neuen Argumentsorten besitzen. Der Benutzer darf den "otherwise-Zweig" dann eingeben, wenn j < k ist und alle fehlenden Konstruktoren $ctr_{j+1} \ldots ctr_K$ des case-schemes keine neuen Argumentsorten mehr besitzen.
Der Benutzer braucht nur die $OPSCHEME_i$,i=1...k, und gegebenenfalls das $OPSCHEME_{K+1}$ einzugeben. Diese Operationsschemata müssen wohldefiniert sein bzgl. der importierten Operationen und der auxiliaries (siehe 3.4.1.). Insbesondere dürfen diese keine charakteristischen Prädikate, Konstruktoren und *-Terme enthalten. Der Rest des Prädikates wird vom Eingabesystem generiert und gepromptet.

B) Wird das charakteristische Prädikat is-s(x) der Sorte s nicht explizit eingegeben, ist es wie folgt definiert:

1. Wenn es irgendeinen Konstruktor ctr : s_1 ... s_n --> s der Sorte s gibt, für dessen neue Argumentsorten ein charakteristisches Prädikat deklariert wurde, das explizit oder implizit definiert ist und nicht identisch 'true' ist (d.h. es gilt nicht is-s_i(x) = true), dann ist is-s(x) folgendermasen definiert:

```
is-s(x) = case x is
                * ctr_1 (x_1,1 ... x_1,n1) :
                  (...(is-s_1,L1(x_1,L1) and ...
                   is-s_1,Lm1(x_1,Lm1) ... )
                .
                .
                * ctr_K (x_K,1 ... x_K,nK) :
                  (...(is-s_K,L1(x_K,L1) and ...
                   is-s_K,LmK(x_K,LmK) ... )
                [ otherwise : true ]
          esac
```

Dabei sind die ctr_i : $s_{i,1}$ ... $s_{i,ni}$ --> s ,i=1...j, alle Konstruktoren von s und die Sorten $s_{i,Li}$ ... $s_{i,Lmi}$,i=1...j, sind die neuen Argumentsorten dieser Konstruktoren. Der "otherwise-Zweig" entfällt, wenn alle Konstruktoren von s neue Argumentsorten besitzen.

2. Wenn die Bedingungen für 1. nicht zutreffen, lautet die Definition des charakteristischen Prädikats:

```
is-s(x) = true
```

Diese Definitionen werden, falls der Benutzer die Prädikate nicht explizit eingibt, vom Eingabesystem in interner Darstellung in der define-carriers-clause erzeugt. Dem Benutzer wird mitgeteilt, welche Prädikate das System erzeugt hat (siehe 5.2. S.113: is-nat).

3.3.5. Die Konstruktordefinition

A) Falls die Definition des charakteristischen Prädikats der Sorte s einer systemgenerierten Form entspricht, sind deren Konstruktoren $ctr_i : s_1 \ldots s_n \dashrightarrow s$ wie folgt definiert:

$$ctr_i\ (x_1 \ldots x_n) = *\ ctr_i\ (x_1 \ldots x_n)$$

Wenn alle charakteristischen Prädikate einer Spezifikation vom Eingabesystem generiert wurden oder die vom Benutzer eingegeben Prädikate dieser Form entsprechen, werden alle Konstruktordefinitionen vom System erzeugt (siehe 5.2. S.113: nat,zero,suc). Andernfalls muß der Benutzer die Konstruktoren der Sorten, für die er das Prädikat selbst eingegeben hat, auch selbst definieren. Alle Konstruktordefinitionen, die das System selbst generieren kann, können nicht vom Benutzer explizit eingegeben werden.

B) Entspricht das charakteristische Prädikat einer Sorte s nicht einer systemgenerierten Form, so sind die Konstruktoren von s, $ctr_i : s_1 \ldots s_n \dashrightarrow s$, folgendermaßen vordefiniert (siehe 5.2. S.125,f: is-stack) :

$$ctr_i\ (x_1 \ldots x_n) = \text{TRANS}\ (\ \text{OPSCHEME}_i\ )$$

Dabei ist OPSCHEME_i, die Eingabe des Algorithmus TRANS, das Operationsschema, das der Benutzer bei der Definition des charakteristischen Prädikates is-s(x) im case-scheme für den Konstruktor ctr_i angegeben hat.

Der Algorithmus TRANS

Der Algorithmus besteht aus 6 Teilen. Jeder beschreibt die Behandlung einer der 6 Möglichkeiten, die OPSCHEME_i annehmen kann.

1. TRANS (true) = | $*\ ctr_i\ (x_1 \ldots x_n)$ |

2. TRANS (false) = op-scheme

3. TRANS (term) =

if term then $*\ ctr_i\ (x_1 \ldots x_n)$
else op-scheme

für alle Terme "term" außer 'true' und 'false'.

Die vom Benutzer einzugebenden Operationsschemata "op-

scheme" müssen wohldefiniert sein bzgl. der importierten Operationen, der auxiliaries und der Konstruktoren (siehe 3.3.1). Insbesondere dürfen sie keine *-Terme enthalten.

4. TRANS (if t then op1 else op2) =

```
┌─────────┐
│ if t    │
│ then    │ TRANS ( op1 )
└─────────┘
PP else TRANS ( op2 )
```

5. TRANS (let a_1=t_1 [,a_2=t_2] ... in op) =

```
┌──────────────────────────────┐
│ let a1=t1 [,a2=t1] ... in    │ TRANS (OP )
└──────────────────────────────┘
```

6. TRANS (case y is $leftpart_1$: op_1
.
.
$leftpart_m$: op_m
[otherwise : op_{m+1}]
esac) =

```
┌──────────────────────────┐
│ case y is leftpart1 :    │ TRANS ( op1 )
└──────────────────────────┘
        .
        .
   ┌──────────────┐
   │ leftpartm :  │ TRANS ( opm )
   └──────────────┘
[  ┌──────────────┐
   │ otherwise :  │ TRANS ( opm+1 ) ]
   └──────────────┘
┌──────┐
│ esac │
└──────┘
```

Bei der expliziten Eingabe des case-schemes ist darauf zu achten, daß die Reihenfolge der case-Zweige dieses Schemas der Reihenfolge im Argumentschema entspricht. Weiterhin müssen die in der linken Seite eines case-Zweiges benutzten Variablen mit den entsprechenden Variablen des Argumentschemas übereinstimmen. Diese Restriktionen sind notwendig, um den Programmieraufwand und den damit verbundenen Zeitbedarf für diese Klausel in einem vertretbaren Rahmen zu halten.

Die umrahmten Teile werden vom Eingabesystem beim Prompten generiert.

3.4. OK-Bedingungen

Schritthaltend mit der Eingabe werden lexikalische und kontextsensitive 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.

Dadurch wird erreicht, daß ein möglicherweise in der Eingabe enthaltener Fehler frühestmöglich erkannt wird. Die lexikalischen Konventionen und die konkrete ASPIK-Syntax sind im Abschnitt 3.5. aufgeführt. Die kontextsensitiven Bedingungen, kurz OK-Bedingungen, werden nachfolgend aufgeführt [BV 81/84].

3.4.1. Begriffliche Grundlagen

Umgebung einer Datei d :

alle Dateien, die beim Öffnen der Datei d geladen werden

Umgebung einer Spezifikation :

alle Objekte (Spezifikationen, Maps) der geöffneten Datei und alle Objekte in der Umgebung dieser Datei

Neue Sorten einer Spezifikation spec :

Menge der in der sorts-clause eingeführten Sorten.

Neue Operationen einer Spezifikation spec :

Menge der in der ops-clause deklarierten Operationen.

Signatur einer Spezifikation spec :

Menge der neuen Sorten und der neuen Operationen.

Explizite auxiliaries einer Spezifikation spec :

Alle vom Benutzer in der auxiliaries-clause explizit eingegebenen Operationen.

Implizite auxiliaries einer Spezifikation spec :

Vereinigung der Menge der in der ops-clause deklarierten EQ-Operationen mit der Menge der EQ-Operationen der auxiliaries-clause, die nicht explizit eingegeben wurden.

Sichtbare Opparts einer Spezifikation spec :

Vereinigung der Mengen von Opparts der Operationen des Imported Interface, der ops-clause, der auxiliaries-clause und der private-ops-clause, soweit vorhanden.

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 besteht opid aus einem einzigen oppart.
Ist opid eine Mixfix-Operation, so besteht opid aus einer Folge von Unterstrich "_" und oppart.

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$_m$.
Die Sorten s_1 ... s_{m-1} sind die Argumentsorten und sie Sorte s_m ist die Ergebnissorte von opid$_i$,i=1,...,n.

3.4.2. Die OK-Bedingungen

3.4.2.1. Specterme und Mapterme

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

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

mapid interface-ok ist und 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

3.4.2.2. Opheader und Operationsdefinitionen

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

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

a) die Sorten $sortid_k$ sind aus dem Exported Interface der aktuellen Spezifikation (siehe 5.2. S.106: into, S.122: empty?,push). 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 (siehe 5.2. S.111: eq-nat).
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) kein oppart von opid in folgender Menge enthalten ist:
{ IF, THEN, ELSE, LET, CASE, ESAC, IN, & , |, _, ==>, <==>, =|=, ==, ALL, EX, OTHERWISE, ELSIF }
Hinweis: der Unterstrich ist das Ersatzsymbol für NOT auf den verwendeten Terminals.

Eine **Operationsdefinition** op(v_1 ... v_n) = opscheme **ist wohldefiniert** bzgl. der Menge der Operationen $\mathcal{O}$, falls

a) die zu definierende Operation op n Argumentsorten hat

b) die Variablen v_1 ... v_n verschieden und nicht identisch mit sichtbaren Opparts oder reservierten Opids sind

c) das Operationschema opscheme wohldefiniert ist bzgl. $\mathcal{O}$, den Variablenmengen $\mathcal{U}$ und $\mathcal{U}_L$ und der Ergebnissorte S. $\mathcal{U}$ enthält die Variablen v_1 ... v_n, $\mathcal{U}_L$ ist leer und S ist die Ergebnissorte der Operation op. $\mathcal{U}_L$ ist eine Teilmenge von $\mathcal{U}$, in der alle let-Variablen einer Operationsdefinition abgelegt werden.

Ein **Operationsschema** ist entweder ein

Term oder
ein let-Schema oder
ein if-Schema oder
ein case-Schema

Ein **Term** t **ist wohldefiniert** bzgl. der Menge der Operationen $\mathcal{O}$, der Variablenmenge $\mathcal{U}$ und der Ergebnissorte S, falls

a) t nur aus Elementen von $\mathcal{O}$, $\mathcal{U}$ und aus ERROR-Konstanten aufgebaut ist

b) für jede Operation in t soviele Argumente existieren, wie durch seine Stelligkeit gefordert ist

c) für jede Operation in t die Sorten der Argumente mit den deklarierten Argumentsorten übereinstimmen

d) die Ergebnissorte der äußersten Operation mit S identisch ist

Ein **let-Schema** LET $v_1=t_1$ ... $v_n=t_n$ IN opscheme **ist wohldefiniert** bzgl. der Menge der Operationen $\mathcal{O}$, den Variablenmengen $\mathcal{U}$ und $\mathcal{U}_L$, und der Ergebnissorte S, falls

a) die Variablen v_1 ... v_n verschieden sind und nicht mit Variablen aus $\mathcal{U}$ sowie mit sichtbaren Opparts oder reservierten Opids übereinstimmen

b) die Terme t_1 ... t_n wohldefiniert bzgl. $\mathcal{O}$ und $\mathcal{U}$ sind

c) das Operationsschema opscheme wohldefiniert ist bzgl.

$\mathcal{O}$, der Menge $\mathcal{U}$ erweitert um die Variablen $v_1 \ldots v_n$, der Menge $\mathcal{U}_L$, ebenfalls erweitert um die Variablen $v_1 \ldots v_n$ und der Ergebnissorte S

Ein **if-Scheme** IF t THEN opscheme 𝕀 ELSEIF t THEN opscheme 𝕀 ... ELSE opscheme

ist wohldefiniert bzgl. der Menge der Operationen $\mathcal{O}$, den Variablenmengen $\mathcal{U}$ und $\mathcal{U}_L$ sowie der Ergebnissorte S, falls

a) die Terme t nach IF oder ELSEIF wohldefiniert sind bzgl. der Mengen $\mathcal{O}$ und $\mathcal{U}$ sowie der Ergebnissorte BOOL

b) die Operationsschemata opscheme wohldefiniert sind bzgl. der Mengen $\mathcal{O}$, $\mathcal{U}$ und $\mathcal{U}_L$ sowie der Ergebnissorte S

Ein **case-Schema** CASE v IS * $ctr_1(v_1 \ldots v_{n,1})$: $opscheme_1$
𝕀 * $ctr_K(v_1 \ldots v_{n,K})$: $opscheme_K$ 𝕀
𝕀 OTHERWISE : $opscheme_{K+1}$ 𝕀
ESAC

ist wohldefiniert , falls

a) die case-Variable v Element von $\mathcal{U}$ ist

b) die Sorte von v eine neue Sorte ist

c) die case-Variable nicht in $\mathcal{U}_L$ ist

d) die Operationen $ctr_1 \ldots ctr_K$ Konstruktoren der Sorte von v sind

e) die Variablen $v_1 \ldots v_{n,i}$ der Operation ctr_i, i=1,...,k, verschieden sind und nicht mit Variablen aus $\mathcal{U}$ sowie mit sichtbaren Opparts oder reservierten Opids übereinstimmen.

f) für alle Konstruktoren der Sorte von v ein case-Zweig existiert, oder der OTHERWISE-Zweig enthalten ist

g) für alle Operationsschemata $opscheme_i$, i=1,...,k, gilt:
$opscheme_i$ ist wohldefiniert bzgl. der Mengen $\mathcal{O}$, $\mathcal{U}'$ und $\mathcal{U}_L$, sowie der Ergebnissorte S . Dabei ist $\mathcal{U}'$ die um die Variablen $v_1 \ldots v_n$, die zur Operation ctr_i gehören, erweiterte Menge $\mathcal{U}$.

h) das Operationsschema $opscheme_{K+1}$, falls vorhanden, wohldefiniert ist bzgl. den Mengen $\mathcal{O}$, $\mathcal{U}$ und $\mathcal{U}_L$, so-

wie der Ergebnissorte S

3.4.2.3. Die Klauseln des spec-headers

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

1.Fall: specid ist in der Umgebung der geöffneten Datei schon definiert
- a) specid ist auf der geöffneten Datei 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'

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 $i=1,...,n$

Eine **Props-clause** PROPS $prop_1$ ... $prop_n$ **ist OK** , falls

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

Eine Property **prop** $_i$, die aus einer Formel der Prädikatenlogik 1. Stufe besteht **ist OK** , falls

a) alle quantifizierten Variablen eindeutig sind und nicht mit den sichtbaren Opparts oder den reservierten Opids übereinstimmen.

b) alle auftretenden Sorten aus dem exported interface sind (siehe 5.2. S.112: nut).

c) die enthaltenen Terme, die nicht auf der linken oder rechten Seite einer Gleichung bzw. Ungleichung stehen, wohldefiniert sind bzgl. der Operationenmenge $\mathcal{O}$, der Variablenmenge $\mathcal{U}$ und der Ergebnissorte BOOL (siehe 5.2. S.112 mitte). Die Menge $\mathcal{O}$ ist die Vereinigung der neuen Operationen mit den Operationen des imported interface. $\mathcal{U}$ enthält alle in der Umgebung des Terms durch Quantoren deklarierte Variablen.

d) die beiden Terme einer Gleichung bzw. Ungleichung wohldefiniert bzgl. $\mathcal{O}$ und $\mathcal{U}$ sind und die selbe Ergebnissorte haben (siehe 5.2. S.107 unten, S.108 oben).

e) das eingegebene label der Property nicht bereits als Propertymarke in der props-clause auftritt.

Eine Property **prop** $_i$ der Form attribute : opid **ist OK** , falls

a) das eingegebene label der Property nicht bereits als Propertymarke in der props-clause vorkommt und entweder

b1) die Attributsbezeichnung attribute ASSOCIATIVE oder COMMUTATIVE ist, die Operation opid zweistellig ist und die Ergebnissorte von opid mit den beiden Argumentsorten übereinstimmt,

oder

b2) die Attributsbezeichnung attribute REFLEXIVE, IRREFLEXIVE, SYMMETRIC, TRANSITIVE, ANTITRANSITVE, EQUIVALENCE oder LINEAR-ORDERING ist, die Operation opid zweistellig ist, die Ergebnissorte BOOL besitzt und ihre beiden Argumentsorten identisch sind.

3.4.2.4. Die Klauseln des spec-bodies

Eine **Cons-clause** CONSTRUCTORS ctr_1 ... ctr_n **ist OK** , falls

a) für jede neue Sorte mindestens ein Konstruktor enthalten ist

b) für alle ctr_i gilt: $i=1,...,n$
ctr_i ist ein Operator der ops-clause und die Ergebnissorte von ctr_i ist eine neue Sorte

c) für alle neuen Sorten s, für die eine Operation eq-s explizit deklariert wurde, nur Konstruktoren angegeben wurden, für deren sämtliche Argumentsorten s' die Operation eq-s' deklariert oder importiert wurden.

Daraus folgt, daß eine cons-clause **leer** und somit **OK** ist, wenn die sorts-clause ebenfalls leer ist.

Eine **Aux-clause** ist **leer** und **OK** , falls die cons-clause ebenfalls leer ist.

Eine **Aux-clause** AUXILIARIES aux_1 ... aux_n **ist OK** , falls

für alle aux_i gilt $i=1,...,n$

a) aux_i ist ein gültiger opheader, wobei aber für alle $opid_j$ von aux_i gilt:

a1) stimmt ein $oppart_i$ von $opid_j$ mit einem oppart aus der ops-clause überein, so sind es identische Operationen

a2) ist $opid_j$ die EQ-Operation eq-s, so muß für alle Argumentsorten s_K der Konstruktoren von s ebenfalls die Operation eq-s_K deklariert bzw. importiert sein.

Die EQ-Operationen der neuen Sorten, die nicht in der ops-clause oder in der auxiliaries-clause explizit deklariert sind und der obigen Bedingung genügen, werden vom Eingabesystem generiert (siehe 5.2. S.113: eq-nat, S.122: eq-stack).

Eine **Def-aux-clause** DEFINE_AUXILIARIES def-aux_1 .. def-aux_n **ist OK**, , falls

für jede in der aux-clause explizit eingeführte auxiliary, die keine EQ-Operation ist, eine entsprechende Operationsdefinition explizit eingegeben worden ist. Diese Operationsdefinitionen müssen wohldefiniert sein bzgl. der Operationen des imported interface und der ex-

pliziten und impliziten auxiliaries.

Die Definitionen der impliziten auxiliaries, nämlich der EQ-Operationen, werden vom Eingabesystem wie in 3.3.2. beschrieben erzeugt.
Daraus folgt, daß eine def-aux-clause **leer** und somit **OK** ist, wenn die auxiliaries-clause oder sogar die cons-clause leer ist.

Eine **Define-carriers-clause** DEFINE-CARRIER $def\text{-}car_1$ ... $def\text{-}car_n$ **ist OK**, falls

> für jede Sorte der sorts-clause die Definition des jeweiligen charakteristischen Prädikats enthalten ist. Diese müssen der in 3.3.4. beschriebenen Definition eines charakteristischen Prädikates entsprechen (siehe 5.2. S.123: is-stack). Die vom Benutzer einzugebenden Subschemata müssen wohldefiniert sein bzgl. der Operationen des imported interface und der auxiliaries.

Die Definitionen der charakteristischen Prädikate, die nicht vom Benutzer explizit eingegeben werden, werden vom Eingabesystem generiert (siehe 5.2. S.113: is-nat).
Daraus folgt, daß eine def-car-clause **leer** und somit **OK** ist, falls die sort-clause ebenfalls leer ist.

Eine **Def-cons-clause** DEFINE-CONSTRUCTOR-OPS $def\text{-}ctr_1$ ... $def\text{-}ctr_n$ **ist OK** , falls

> für jeden Konstruktor der cons-clause eine Operationsdefinition enthalten ist. Diese müssen der in 3.3.5. beschriebenen Konstruktor-Definition entsprechen (siehe 5.2. S.125: push, empty). Die darin auftretenden, vom Benutzer einzugebenden Subschemata müssen wohldefiniert sein bzgl. der Operationen des imported interface, der Konstruktoren und der auxiliaries.

Das Eingabesystem generiert die Konstruktordefinitionen wie in 3.3.5. beschrieben für alle Konstruktoren derjenigen Sorten, für die es auch das charakteristische Prädikat erzeugt hat (siehe 5.2. S.113: suc,zero).
Daraus folgt, daß eine def-cons-clause **leer** und somit **OK** ist, wenn die cons-clause ebenfalls leer ist.

Eine **Private-ops-clause** ist **leer** und **OK**, falls sowohl die sorts-clause als auch die ops-clause leer sind

Eine **Private-ops-clause** PRIVATE-OPS oph_1 ... oph_n **ist OK**, falls

für alle oph_i gilt: $i=1,...,n$

a) oph_i ist ein gültiger opheader.

b) alle opparts von oph_i sind nicht identisch mit opparts der ops- und der auxiliaries-clause.

c) alle opids von oph_i deklarieren keine EQ-Operationen.

Eine **Def-ops-clause** DEFINE-OPS $def\text{-}ops_1$ ... $def\text{-}ops_n$ **ist OK**, falls

für jede explizite neue Operation, die nicht bereits in der def-aux-clause oder in der def-cons-clause definiert wurde, eine Operationsdefinition vorhanden ist. Diese müssen wohldefiniert sein bzgl. der Operationen des imported interface, der ops-clause, der auxiliaries-clause und der private-ops-clause (siehe 5.2. S.114: one, S.115: pred,+, S.116: -,/, S.118: exp, S.126: full,empty?, S.127: pop).

Daraus folgt, daß eine def-ops-clause **leer** und somit **OK** ist, wenn die private-ops-clause leer ist und entweder alle Operationen der ops-clause bereits als Konstruktoren oder auxiliaries definiert sind, oder die ops-clause ebenfalls leer ist.

3.5. ASPIK-Syntax

3.5.1. Notation

_____	unterstrichene Teile können automatisch generiert werden
::	trennt linke und rechte Seite einer Regel
[ ]	umschließen optionale Teile
|	trennt zwei Alternativen fUr die rechte Seite einer Regel
[[ ]]	sind Metaklammern
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 % ( ) == =|= & | <==> ==> [ ] - -> *

3.5.2. Die ASPIK-Syntax

```
specification :: specid-clause
                 comment-clause
                 use-clause
                 [[ sorts-clause ]]
                 [[ ops-clause ]]
                 [[ props-clause ]]
                 [[ body ]]
                 PP endspec
```

3.5.2.1. Die Klauseln des spec-headers

```
specid-clause  :: PP spec PP specid [[ ; | b ]]
specid :: simple-sortid :: id (siehe 3.5.3.)

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 ]]

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 ]]

props-clause    :: PP props next-prop
next-prop       :: PP [[ ; ] [[ props-elem , next-prop ]] ]
props-elem      :: label property [[ b comment ]]
label           :: [ [[ letter ] digit ]] ... ] b
```

3.5.2.2. Die Klauseln des spec-bodies

```
body            :: PP spec-body
                   [ carrier-part ]
                   [ op-part ]

carrier-part    :: cons-clause
                   [ aux-clause
                       [ def-aux-clause ] ]
                   [ carrier-clause ]

cons-clause     :: PP constructors next-cons
next-cons       :: PP [ ; | b | [ cons-elem , next-cons ] ]
cons-elem       :: simple-opid [ b comment ]

aux-clause      :: PP auxiliaries next-aux
next-aux        :: PP [ ; | [ aux-elem , next-aux ] ]
aux-elem        :: opheader [ b comment ]

def-aux-clause  :: PP define-auxiliaries next-def-aux
next-def-aux    :: PP [ ; | b | [ def-aux-elem , next-def-aux
                   ] ]
```

```
def-aux-elem    :: op-body ⟦ b comment ⟧

carrier-clause  :: PP define-carriers next-carrier
next-carrier    :: PP [[ ; | b | [[ carrier-elem , next-carrier
                   ]] ]]
carrier-elem    :: char-predicate ⟦ b comment ⟧
char-predicate  :: IS = simple-sortid ( varid ) = char-case
char-case       :: PP case varid is
                         * left-part : op-scheme
                         ⟦ , PP * left-part : op-scheme ⟧ ...
                         ⟦ , otherwise : opscheme ⟧
                   PP esac
                   (siehe Abschnitt 3.3.4)
varid           :: id (siehe 3.5.3.)

op-part         :: ⟦ def-cons-clause ⟧
                   ⟦ priv-ops-clause ⟧
                   def-ops-clause

def-cons-clause :: PP define-constructor-ops next-def-cons
next-def-cons   :: PP [[ ; | b | [[ def-cons-elem , next-def-cons
                   ]] ]]
def-cons-elem   :: cons-body ⟦ b comment ⟧
```

```
cons-body       :: op-body
                   (siehe Abschnitt 3.3.5)


priv-ops-clause :: PP private-ops next-priv-op
next-priv-op    :: PP [[ ; | [[ priv-ops-elem , next-priv-op
                   ]] ]]
priv-ops-elem   :: opheader [ b comment ]


def-ops-clause  :: PP define-ops next-def-ops
next-def-ops    :: PP [[ ; | b | [[ def-ops-elem , next-def-ops
                   ]] ]]
def-ops-elem    :: op-body [ b comment ]
```

3.5.2.3. Specterme und Mapterme

```
specterm          :: specid ▯
                     specterm ▯ { simple-tripel ▯ , simple-triple
                     ▯ ... } ▯ ...
simple-tripel     :: PP specterm simple-arrowterm specterm
simple-arrowterm  :: arrow ▯ simple-arrow-composition
arrow             :: - ▯ arrowid ▯ ->
arrowid           :: id (siehe 3.5.3.)
simple-arrow-composition ::  simple-arrowterm
                             ▯ specterm simple-arrowterm ▯ ...

mapterm           :: simple-triple
```

3.5.2.4. Opheader und Operationsdefinitionen

```
opheader        :: simple-opid [ , simple-opid ] ... :
                   [ sortid [ , sortid ] ... ] --> sortid
simple-opid     :: [ _ ] ... simple-opid-part
                   [ _ ... simple-opid-part ] ... [ _ ] ...
simple-opid-part :: id (siehe 3.5.3.)
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 ]] ...
```

```
op-body         :: left-part = op-scheme
left-part       :: prefix-left-part | mixfix-left-part
prefix-left-part :: simple-opid [ (varid [ , varid ] ... ) ]
mixfix-left-part ::  [ [ varid  [ , varid ]  ... b ]
                     simple-opid-part b ] ...
                     [ varid [ ,  varid ] ...  b ] simple-opid-
                     part
                     [ b varid [ , varid ] ... ]
```

```
op-scheme      :: term ▯ let-scheme ▯ if-scheme ▯ case-scheme
let-scheme     :: let varid = term ▯ , varid = term ▯ ...
                  PP in op-scheme
if-scheme      :: if term
                  PP then op-scheme
                  ▯ elseif term
                  PP then op-scheme ▯ ...
                  PP else op-scheme
case-scheme    :: case varid is
                        PP * left-part : op-scheme
                        ▯ , PP * left-part : op-scheme ▯ ...
                        ▯ , otherwise : op-scheme ▯
                  PP esac
```

3.5.2.5. Terme

```
term            :: ( term ) ‖ varid ‖ prefix-term ‖ mixfix-term
prefix-term     :: opid-part ⟦ ( term ⟦ , term ⟧ ... ) ⟧
mixfix-term     :: ⟦ ⟦ term ⟦ , term ⟧ ... b ⟧ opid-part b ⟧ ...
                   ⟦ term ⟦ , term ⟧ ... b ⟧ opid-part
                   ⟦ b term ⟦ , term ⟧ ... ⟧
opid-part       :: ⟦ prefix . ⟧ simple-opid-part
simple-opid-part :: id (siehe 3.5.3)
prefix             (siehe 3.5.2.4.)
```

3.5.2.6. Properties

```
property        :: attribute ▯ quantification
attribute       :: attribute-name : opid
attribute-name  :: associative  ▯  reflexive  ▯  irreflexive  ▯
                   transitive ▯ equivalence ▯ linear-ordering  ▯
                   commutative ▯ symmetric ▯ antisymmetric
opid            :: ⟦ _ ⟧ ... opid-part  ⟦ _ ... opid-part ⟧
                    ... ⟦ _ ⟧ ...
quantification  :: coimplication ▯
                   quantifier varid ⟦ ,varid ⟧ ... : sortid b
                   quantification
quantifier      :: all ▯ ex
coimplication   :: implication ▯
                   implication <==> implication
implication     :: disjunction ▯
                   disjunction ==> disjunction
disjunction     :: conjunction ▯
                   conjunction | conjunction
conjunction     :: negation ▯
                   negation & negation
negation        :: atomic-formula ▯ ¬ atomic-formula
atomic-formula  :: ( quantification ) ▯ atom
atom            :: term ▯ equation ▯ inequation
equation        :: term == term
```

inequation :: term =|= term

3.5.2.7. Terminal-Regeln

```
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
-->     :: [ b ] --> [ b ]
(       :: [ b ] ( [ b ]
)       :: [ b ] ) [ b ]
==      :: b == b
=|=     :: b =|= b
&       :: b & b
|       :: b | b
<==>    :: b <==> b
==>     :: b ==> b
¬       :: [ b ] _ [ b ]
```

```
[       :: [ b ] [ [ b ]
]       :: [ b ] ] [ b ]
spec    :: [ b ] SPEC b
use     :: [ b ] USE b
sorts   :: [ b ] SORTS b
ops     :: [ b ] OPS b
props   :: [ b ] PROPS b
constructors ::  [ b ] CONSTRUCTORS b
auxiliaries ::  [ b ] AUXILIARIES b
define-auxiliaries ::  [ b ] DEFINE-AUXILIARIES b
define-carriers ::  [ b ] DEFINE-CARRIERS b
define-constructor-ops ::  [ b ] DEFINE-CONSTRUCTOR-OPS b
private-ops ::  [ b ] PRIVATE-OPS b
define-ops ::  [ b ] DEFINE-OPS b
endspec ::  [ b ] ENDSPEC b
spec-body ::  [ b ] SPEC-BODY b
if      :: [ b ] IF b
then    :: b THEN b
elseif  :: b ELSEIF b
else    :: b ELSE b
case    :: [ b ] CASE b
is      :: b IS b
otherwise :: b OTHERWISE [ b ]
esac    :: b ESAC [ b ]
let     :: [ b ] LET b
in      :: b IN b
```

```
all         :: [ b ] ALL b
ex          :: [ b ] EX b
associative :: [ b ] ASSOCIATIVE b
reflexive   :: [ b ] REFLEXIVE b
irreflexive :: [ b ] IRREFLEXIVE b
symmetric   :: [ b ] SYMMETRIC b
antisymmetric :: [ b ] ANTISYMMETRIC b
transitive  :: [ b ] TRANSITIVE b
equivalence :: [ b ] EQUIVALENCE b
linerar-ordering :: [ b ] LINIEAR-ORDERING b
commutative :: [ b ] COMMUTATIVE b
```

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

3.5.3. 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
()	umschließen einen Term
*	kennzeichnet einen syntaktischen Konstruktor

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

4. Literaturverzeichnis

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

- [BGGORV 83] Beierle,C., Gerlach, M., Göbel, R., Olthoff, W., Raulefs, P., Voß, A. : Integreated Program Development and Verifikation: IN: H. L. Hausen (ed.): Symposium on Software Validation, North-Holland Publ. Co., Amsterdam 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

- [BV 81/84] Beierle, C., Voß, A. : Entwurfsbeschreibung und Arbeitsunterlagen zum Spezifikationsentwicklungssystem SPESY, SEKI-Projekt, Univeratiät Kaiserslautern, Fachbereich Informatik, 1981-1984

- [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

- [BV 85] Beierle, C., Voß, A. : Specification and implementation of abstract data types in a support environment for the development of verified software, SEKI-Projekt, Universität Kaiserslautern, Fachbereich Informatik, (erscheint 1985)

- [Eis 82] Eisinger, N. : A pracmatic module concept for INTERLISP, Interner Bericht Nr.23/82, Universität Karlsruhe, Institut für Informatik 1, 1982

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

- [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

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

August 1980

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

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

- [Li 85] Lichter,H. : Ein interaktives und syntaxorientiertes Eingabesystem für algebraische und algorithmische Spezifikationen in der Spezifikationsentwicklungsumgebung SPESY, Universität Kaiserslautern, Fachbereich Informatik, Januar 1985

- [Mat 84a] Matheis, H. : Ein interaktives und syntaxorientiertes Eingabesystem für algebraische Spezifikationen, Universität Kaiserslautern, Fachbereich Informatik, April 1984

- [Mat 84b] Matheis, H. : Ein interaktives und syntaxorientiertes Eingabesystem für algebraische Spezifikationen, Memo SEKI-84-03-I, Universität Kaiserslautern, Fachbereich Informatik, April 1984

- [Olt 85] Olthoff, W. : The Realization Level, Universität Kaiserslautern, Fachbereich Informatik (erscheint 1985)

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

- [Sch 85] Schölles, V. : Das Spezifikationssytem SPESY, Universität Kaiserslautern, Fachbereich Informatik (erscheint 1985)

- [Som 84] Sommer, W. : Komponenten des Interaktiven Systems SPESY zur Unterstützung Integrierter Programm-Spezifikation und -Verifikation, SEKI-Projekt, Memo SEKI-84-02, Universität Kaiserslautern, Fachbereich Informatik, 1984

- [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

5. SPESY-Sitzungsprotokolle

Der erste Teil der folgenden Sitzungsprotokolle wurde bereits von Hans Matheis mit einem Rumpfsystem von SPESY erstellt [Mat 84]. Sie dokumentieren besonders anschaulich das Dialogverhalten des Eingabesystems bzgl. der use-clause und der use-Beziehung von Spezifikationen. Die in diesen Protokollen auftretenden System- und Filelevelkommandos sind zum Teil nur noch verändert, oder überhaupt nicht mehr in der zur Zeit implementierten SPESY-Version verfügbar.

Der zweite Teil wurde mit einer vollständigen SPESY-Version erstellt. Bei diesen Protokollen wurde besonders auf das Dialogverhalten des Systems bei vordefinierten Teilen einer Spezifikation und auf das Zusammenspiel zwischen Spec-Editor- und Spec-Inputlevel geachtet.

5.1. Sitzungsprotokolle des Teil-Eingabesystems von Hans Matheis

```
(OUT)   PROTOCOL STARTED.
(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)    _%<=_
(QUT)   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 ;
(OUT)
(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 ALGORIHMICALLY */;
(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
(OUT)   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} -NATORDSLOT
(OUT)     S-> INSERTION ORDELEEM -ORDELEM-> NA
(OUT)     000044 RECORDS PRINTED. CONTINUE?
(IN)
(OUT)     TORD}}
(OUT)
(OUT)     operations from ALG{ORDELEM -ORDELM-> NATORD}{SLOTS{ORDELEM -ORDELM-> NATORD} -NATORDSL
(OUT)     OTS-> INSERTION ORDELEM -ORDELEM->
(OUT)     NATORD}} :
(OUT)       SORT: LIST --> LIST
(OUT)
(OUT)     no sorts from ALG(NATORD)(INSERTION(NATORD))
(OUT)     no operations from ALG(NATORD)(INSERTION(NATORD))
(OUT)     ENTER CMD:
(OUT)
(IN)      close
(OUT)     %  D800 ERASE FILE LISP.DATA.WS.MAPTABLE.00
(OUT)     READY:
(OUT)
(IN)      end
```

5.2. Sitzungsprotokolle des Gesamt-Eingabesystems

```
(OUT)   PROTOCOL STARTED.
(OUT)   SYSTEMLEVEL:
(OUT)
(IN)     open sbq-level1 sys.boolfile ;
(OUT)   FILE SBQ-LEVEL1 GENERATED.
(OUT)   FILELEVEL:
(OUT)
(IN)     input spec elem
(OUT)           ************************************************************
(OUT)           ***                                                      ***
(OUT)           ***              S P E C  - E D I T O R                  ***
(OUT)           ***             VERSION VOM:  10. 12. 1984               ***
(OUT)           ***                                                      ***
(OUT)           ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)           ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN      ***
(OUT)           ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)           ***                                                      ***
(OUT)           ************************************************************
(OUT)   ***************   I N P U T L E V E L     FOR      SPECS    ***************
(OUT)   ***************             VERSION :  02.01.1985             ***************
(OUT)   ENTER COMMENT OR ;
(OUT)
(IN)    /* loose specification of just a set */
(OUT)   use
(OUT)   ENTER SPECTERM COMMENT
(OUT)
(IN)    bool ;
(OUT)   sorts
(OUT)   ENTER SORTID COMMENT OR ;
(OUT)
(IN)    ?
(OUT)   SORTID::= SIMPLE_CHAR |
(OUT)   UNSPECIALIZED_CHAR  [ SIMPLE_CHAR | UNSPECIALIZED_CHAR ] ...
(OUT)
(IN)     ?
(OUT)   SIMPLE_CHAR MAY BE ALL CHARACTERS DIFFERENT FROM BLANK PROZENT AND SPECIAL_CHAR
(OUT)
(IN)    elem ;
(OUT)   ops
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    end
(OUT)   ***************           E N D          I N P U T        ***************
(OUT)   CPU TIME USED : 8520 ms.
(OUT)
(OUT)   POSITION: TOP OF SPEC ELEM    LIST
(OUT)   spec ELEM
(OUT)       /* LOOSE SPECIFICATION OF JUST A SET */
(OUT)      use   BOOL ;
```

```
(OUT)       sorts ELEM;
(OUT)     endspec
(OUT)
(OUT)     SPEC-EDIT-LEVEL:
(OUT)
(IN)       end
(OUT)     EDITOR-LEVEL ENDED.
(OUT)     FILELEVEL:
(OUT)
(IN)       input spec limited-lifo
(OUT)               ************************************************************
(OUT)               ***                                                      ***
(OUT)               ***                S P E C  - E D I T O R                ***
(OUT)               ***               VERSION VOM:  10. 12. 1984             ***
(OUT)               ***                                                      ***
(OUT)               ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)               ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN       ***
(OUT)               ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)               ***                                                      ***
(OUT)               ************************************************************
(OUT)     ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)     ***************                VERSION :  02.01.1985            ***************
(OUT)     ENTER COMMENT OR ;
(OUT)
(IN)      /* loose specification of a limited container behaving lifo-like as long as it is not full */
(OUT)     use
(OUT)     ENTER SPECTERM COMMENT
(OUT)
(IN)      elem ; sorts
(OUT)     ENTER SORTID COMMENT OR ;
(OUT)
(IN)      container ; ops
(OUT)     ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)      lt : container container --> bool
(OUT)     ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)      gt : container container --> bool /* lower , greater */
(IN)      ?
(OUT)     OP-HEADER::= OPIDS : [ DOMAINS ] --> CODOMAIN
(OUT)
(IN)       ?
(OUT)     OPIDS::= ID [ ,ID, ... ,ID ]
(OUT)
(IN)       into : container elem -> conteiner
(OUT)     *** THE OPHEADER, BEGINNING WITH THE FOLLOWING OPIDS, HAS NO COMPLETE DOMAIN-LIST OR CODOMAIN ***
(OUT)      INTO  :
(OUT)     ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)       into : container elem --> conteiner
(OUT)     *** THE FOLLOWING SORTIDS ARE NEITHER IN THE INTERFACE NOR IN SORTS-CLAUSE ***
(OUT)      CONTEINER
```

```
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     into : container elem --> container
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    last-in : container --> elem
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    first-out : container --> elem , filled? : container --> bool
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   props
(OUT)   ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)    ?
(OUT)   PROPERTY ::= [LABEL] ATTRIBUTE | QUANTIFICATION
(OUT)
(IN)     ?
(OUT)   ATTRIBUTE ::= ATRIBUTENAME : OPID
(OUT)
(IN)     ?
(OUT)   ATTRIBUTENAME ::= ASSOCIATIVE | REFLEXIVE | IRREFLEXIVE | SYMMETRIC |
(OUT)                     TRANSITIVE | LINEAR-ORDERING | COMMUTATIVE |
(OUT)                     EQUIVALENCE | ANTITRANSITIVE
(OUT)
(IN)     ?
(OUT)   QUANTIFICATION ::= COIMPLICATION | QUANTIFIER VARID[, VARID] : SORTID QUANTIFICATION
(OUT)
(IN)     ?
(OUT)   COIMPLICATION ::= IMPLICATION | IMPLICATION <==> IMPLICATION
(OUT)   IMLPICATION   ::= DISJUNCTION | DISJUNCTION ==> DISJUNCTION
(OUT)   DISJUNCTION   ::= CONJUNCTION | CONJUNCTION ' | ' CONJUNCTION
(OUT)   CONJUNCTION   ::= NEGATION | NEGATION & NEGATION
(OUT)   NEGATION      ::= ATOMIC-FORMULA | _ ATOMIC-FORMULA
(OUT)   ATOMIC-FORMULA ::= (QUANTIFICATION) | ATOM
(OUT)   ATOM          ::= TERM | EQUATION | INEQUATION
(OUT)   EQUATION      ::= TERM == TERM
(OUT)   INEQUATION    ::= TERM =|= TERM
(OUT)
(IN)     all c: container all e elem _filled(c) ==> (last-in(into(c,a)) == e & first-aut( into(c,e)  == c)
(OUT)
(OUT)   *** MISSING COLON AFTER VARID-LIST ***
(OUT)    E,ELEM,_FILLED(C),==>,(LAST-IN(INTO(C,A)),==,E,&,FIRST-AUT(,INTO(C,E),==,C)
(OUT)   ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     all c: container all e:elem _filled?(c) ==> (last-in(into(c,a)) == e & first-aut (into(c,e)  == c)
(OUT)
(OUT)   *** INPUT IS IGNORED AT :    ) ) == E & FIRST-AUT ( INTO ( C,E ) . . .
(OUT)   *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)   A
(OUT)   ENTER PROPERTY COMMENT OR ;
```

```
(OUT)
(IN)      all c: container all e:elem _filled?(c) ==> (last-in(into(c,e)) == e & first-aut (into(c,e)  == c)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    ( INTO ( C,E )   == C )
(OUT)    *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)    FIRST-AUT
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)      all c: container all e:elem _filled?(c) ==> (last-in(into(c,e)) == e & first-out (into(c,e)  == c)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    == C )
(OUT)    *** MISSSING CLOSING-PARENTHESIS BEHIND THE FOLLOWING PREFIX-OPID ***
(OUT)    FIRST-OUT
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)      all c: container all e:elem _filled?(c) ==> (last-in(into(c,e)) == e & first-out (into(c,e)) == c)
(OUT)
(OUT)    *** THE FOLLOWING TERMS ARE OF DIFFERENT DOMAIN ***
(OUT)      (FIRST-OUT (INTO (C)
(OUT)                       (E)))
(OUT)      (C)
(OUT)
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     end
(OUT)    ****************          E N D          I N P U T          ****************
(OUT)    CPU TIME USED : 47060 ms.
(OUT)
(OUT)    POSITION: TOP OF SPEC LIMITED-LIFO    LIST
(OUT)    spec LIMITED-LIFO
(OUT)        /* LOOSE SPECIFICATION OF A LIMITED CONTAINER BEHAVING LIFO-LIKE AS    */
(OUT)        /* LONG AS IT IS NOT FULL */
(OUT)       use   ELEM ;
(OUT)       sorts CONTAINER;
(OUT)       ops   LE: CONTAINER CONTAINER  --> BOOL
(OUT)             GE: CONTAINER CONTAINER  --> BOOL
(OUT)             INTO: CONTAINER ELEM  --> CONTAINER
(OUT)             LAST-IN: CONTAINER  --> ELEM
(OUT)             FIRST-OUT: CONTAINER  --> ELEM
(OUT)             FILLED?: CONTAINER  --> BOOL;
(OUT)    endspec
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      ops
(OUT)
(OUT)    POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-LIFO    LIST
(OUT)       ops   LE: CONTAINER CONTAINER  --> BOOL
(OUT)             GE: CONTAINER CONTAINER  --> BOOL
(OUT)             INTO: CONTAINER ELEM  --> CONTAINER
(OUT)             LAST-IN: CONTAINER  --> ELEM
(OUT)             FIRST-OUT: CONTAINER  --> ELEM
```

```
(OUT)          FILLED?: CONTAINER  --> BOOL;
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      ch 5 first-out : container --> container
(OUT)    OPS
(OUT)    CH
(OUT)
(OUT)
(OUT)    POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-LIFO    LIST
(OUT)       ops   LE: CONTAINER CONTAINER  --> BOOL
(OUT)             GE: CONTAINER CONTAINER  --> BOOL
(OUT)             INTO: CONTAINER ELEM  --> CONTAINER
(OUT)             LAST-IN: CONTAINER  --> ELEM
(OUT)             FIRST-OUT: CONTAINER  --> CONTAINER
(OUT)             FILLED?: CONTAINER  --> BOOL;
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      input
(OUT)    ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)    ***************                 VERSION :  02.01.1985          ***************
(OUT)    ***************     START   WITH   THE   FOLLOWING   CLAUSE     ***************
(OUT)    OPS-CLAUSE
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    props
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     all c:container all e:elem _filled?(c) ==> (last-in(into(c,e)) == e & first-out (into(c,e)) == c)
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     end
(OUT)    ***************            E N D           I N P U T         ***************
(OUT)    CPU TIME USED : 9659 ms.
(OUT)
(OUT)    POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-LIFO    LIST
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      props
(OUT)
(OUT)    POSITION: TOP OF PROPS-CLAUSE OF SPEC LIMITED-LIFO    LIST
(OUT)       props [PROP1] ALL C:CONTAINER   ALL E:ELEM
(OUT)                      _ FILLED?(C)
(OUT)                       ==> LAST-IN(INTO(C,E)) == E & FIRST-OUT(INTO(C,E)) == C;
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      end
```

```
(OUT)    EDITOR-LEVEL ENDED.
(OUT)    FILELEVEL:
(OUT)
(IN)      help
(OUT)    commands of filelevel:
(OUT)        ABORT
(OUT)        CONTROL    alias    C
(OUT)        CONTROL?    alias    C?
(OUT)        DELETE    alias    DEL
(OUT)        DELETEMESSAGE    alias    DELM
(OUT)        DIRECTORY      alias    DIR
(OUT)        EDIT
(OUT)        END
(OUT)        ENVIRONMENT    alias    ENV
(OUT)        ENVIRONMENT?    alias ENV?
(OUT)        FIND    alias    F
(OUT)        HELP    alias    H      alias   ?
(OUT)        LIST       alias   L
(OUT)        LISTFILES       alias     LISTF    alias   LF
(OUT)        MESSAGE     alias     M
(OUT)        MESSAGE?      alias    M?
(OUT)        PRINT      alias    P
(OUT)        PROVE
(OUT)        RL
(OUT)
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)    FILELEVEL:
(OUT)
(IN)      end
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open sbq-level2
(OUT)    ENTER FILES OF ENVIRONMENT:
(OUT)
(IN)      sbq-level1 ;
(OUT)    FILE SBQ-LEVEL2 GENERATED.
(OUT)    FILELEVEL:
(OUT)
(IN)      input spec nat
(OUT)             ************************************************************
(OUT)             ***                                                      ***
(OUT)             ***                S P E C - E D I T O R                 ***
(OUT)             ***               VERSION VOM:  10. 12. 1984             ***
(OUT)             ***                                                      ***
(OUT)             ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)             ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN       ***
(OUT)             ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)             ***                                                      ***
(OUT)             ************************************************************
(OUT)    ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)    ***************               VERSION :  02.01.1985             ***************
(OUT)    ENTER COMMENT OR ;
```

```
(OUT)
(IN)    /* standard algorithmic definition of the natural numbers */
(OUT)   use
(OUT)   ENTER SPECTERM COMMENT
(OUT)
(IN)    nat
(OUT)   *** THE FOLLOWING SPECTERM DOESN'T EXIST IN THE ENVIRONMENT ***
(OUT)   NAT
(OUT)   ENTER SPECTERM COMMENT
(OUT)
(IN)    bool ;
(OUT)   sorts
(OUT)   ENTER SORTID COMMENT OR ;
(OUT)
(IN)    nat ; ops zero : --> nat
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    suc,pred : nat --> nat
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    _+_,_-_,mult,_/_,exp,eq-nat : nat nat --> nat
(OUT)   *** THE FOLLOWING OPIDS DECLARE AN INVALID OPERATION  EQ ***
(OUT)    EQ-NAT
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    zero? : nat --> bool
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    _lt_,_le_,_gt_,_ge_,eq-not : nat nat : bool
(OUT)   *** THE OPHEADER, BEGINNING WITH THE FOLLOWING OPIDS, HAS NO COMPLETE DOMAIN-LIST OR CODOMAIN ***
(OUT)    _LT_  _LE_  _GT_  _GE_  EQ-NOT  :
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    _lt_,_le_,_gt_,_ge_,eq-not : nat nat --> bool
(OUT)   *** THE FOLLOWING OPIDS DECLARE AN INVALID OPERATION  EQ ***
(OUT)    EQ-NOT
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    eq-nat : nat nat --> bool
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    [_] : nat nat --> nat /* binary natural numbers */
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    end
(OUT)   ***************          E N D          I N P U T          ***************
(OUT)   CPU TIME USED : 7818 ms.
(OUT)
(OUT)   POSITION: TOP OF SPEC NAT    LIST
(OUT)   spec NAT
(OUT)       /* STANDARD ALGORITHMIC DEFINITION OF THE NATURAL NUMBERS */
(OUT)      use   BOOL ;
```

```
(OUT)       sorts NAT;
(OUT)       ops   ZERO:  --> NAT
(OUT)             SUC,PRED: NAT  --> NAT
(OUT)             _+_,_-_,MULT,_/_,EXP: NAT NAT  --> NAT
(OUT)             ZERO?: NAT  --> BOOL
(OUT)             _LT_,_LE_,_GT_,_GE_: NAT NAT  --> BOOL
(OUT)             EQ-NAT: NAT NAT  --> BOOL
(OUT)             [_]: NAT NAT  --> NAT;
(OUT)    endspec
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      input
(OUT)    ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)    ***************                VERSION :  02.01.1985            ***************
(OUT)    ***************     START   WITH   THE   FOLLOWING   CLAUSE     ***************
(OUT)    OPS-CLAUSE
(OUT)    ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     one,two,three,four,five,six,seven,eigth,nine,ten : --> nat ;
(OUT)    props
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [nat1] all x,y:nut [ x,y ] =  (mult x,ten) + y)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     [ X,Y ] = ( MULT X,TEN ) + Y )
(OUT)    *** THE FOLLOWING SORTID IS INVALID IN PROPS-CLAUSE ***
(OUT)     NUT
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [nat1] all x,y:nat [ x,y ] =  (mult x,ten) + y) ;
(OUT)
(OUT)    *** INPUT IS IGNORED AT :      = ( MULT X,TEN ) + Y ) ;
(OUT)    *** THE FOLLOWING TERM MUST BE BOOLEAN ***
(OUT)      ([_] (X)
(OUT)           (Y))
(OUT)
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [nat1] all x,y:nat [ x,y ] == (mult x,ten) + y) ;
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    X,TEN ) + Y ) ;
(OUT)    *** MISSING OPENING-PARENTHESIS BEHIND THE FOLLOWING PREFIX-OP ***
(OUT)    MULT
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [nat1] all x,y:nat [ x,y ] == (mult(x,ten) + y) ;
(OUT)    spec-body
(OUT)    constructors
(OUT)    ENTER CONSTRUCTOR COMMENT
(OUT)
```

```
(IN)    ;
(OUT)
(OUT)   *** FIRST YOU MUST ENTER CONSTRUCTORS FOR THE FOLLOWING NEW SORTS ***
(OUT)    NAT
(OUT)   ENTER CONSTRUCTOR COMMENT
(OUT)
(IN)    zero, suc
(OUT)   ENTER CONSTRUCTOR COMMENT OR ;
(OUT)
(IN)    zero
(OUT)
(OUT)   *** THE FOLLOWING CONSTRUCTOR OCCURS TWO OR MORE TIMES, ACCEPTED ONLY ONCE ***
(OUT)    ZERO
(OUT)   ENTER CONSTRUCTOR COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   auxiliaries
(OUT)   ENTER AUXILIARY COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   !!! SPESY DECLARES THE EQ-OPERATION FOR THE FOLLOWING SORTS !!!
(OUT)   NAT
(OUT)   !!! SPESY GENERATES THE FOLLOWING EQ-DEFINITIONS !!!
(OUT)    EQ-NAT
(OUT)   define-carriers
(OUT)   ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)    ?
(OUT)   CHARACTERISTIC-PREDICATE ::= IS- SIMPLE-SORTID (VARID) = CASESCHEME
(OUT)
(IN)     ?
(OUT)   CASESCHEME :: = CASE VARID IS
(OUT)                       PP * LEFTPART : OPSCHEME
(OUT)                       [, PP * LEFTPART : OPSCHEME] ...
(OUT)                       [, OTHERWISE : OPSCHEME]
(OUT)                   PP ESAC
(OUT)   THIS CASESCHEME IS PREDEFINED, SEE USERMANUAL
(OUT)
(IN)     ;
(OUT)   !!! SPESY GENERATES CHARACTERISTIC PREDICATES FOR THE FOLLOWING SORTS !!!
(OUT)    NAT
(OUT)   define-constructor-ops
(OUT)   !!! SPESY GENERATES THE DEFINITION FOR THE FOLLOWING CONSTRUCTORS !!!
(OUT)    ZERO  SUC
(OUT)
(OUT)   private-ops
(OUT)   ENTER PRIVATE-OPERATION COMMENT OR ;
(OUT)
(IN)    ; define-ops
(OUT)   ENTER OP-DEFINITION COMMENT
(OUT)
(IN)    help
```

```
(OUT)    OPBODY ::= LEFTPART = OPSCHEME
(OUT)
(IN)      h
(OUT)    LEFTPART ::= PREFIX-LEFTPART | MIXFIX-LEFTPART
(OUT)    OPSCHEME ::= LETSCHEME | CASESCHEME | IFSCHEME | TERM
(OUT)
(IN)      one= suc(0)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     )
(OUT)    *** THE FOLLOWING OPID-PARTS ARE INVALID IN TERM ***
(OUT)     0
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     one = suc(zero) , two = suc(one)
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     thre= suc(two) , four = suc(three)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     SUC ( TWO ) , FOUR = SUC ( THREE )
(OUT)    *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)    =
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     three= suc(two) , four = suc(three)
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     five = suc(four)
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     six= suc(suc(suc(suc(suc(suc(zero)))))
(OUT)
(OUT)    *** MISSSING CLOSING-PARENTHESIS BEHIND THE FOLLOWING PREFIX-OPID ***
(OUT)    SUC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     six= suc(suc(suc(suc(suc(suc(zero))))))
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     seven = suc(six) , eigth = suc(seven) , nine = suc(eigth)
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     ten = suc(nine)
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     end
(OUT)    ***************          E N D          I N P U T          ***************
(OUT)    CPU TIME USED : 32261 ms.
(OUT)
(OUT)    POSITION: TOP OF SPEC NAT    LIST
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
```

```
(IN)      input
(OUT)     ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)     ***************                VERSION :  02.01.1985           ***************
(OUT)     ***************     START   WITH   THE   FOLLOWING   CLAUSE     ***************
(OUT)     DEFINE-OPS-CLAUSE
(OUT)     define-ops
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      pred(n) = case n is
(OUT)     * ZERO :
(OUT)
(IN)      error-nst
(OUT)
(OUT)     *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)     ERROR-NST
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      pred(n) = case n is
(OUT)     * ZERO :
(OUT)
(IN)      error-nat
(OUT)     * SUC ( NA0 ) :
(OUT)
(IN)      na0
(OUT)     ESAC
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      n.1 + n.2 = case n3 is
(OUT)
(OUT)     *** INPUT IS IGNORED AT :      + N.2 = CASE N3 IS
(OUT)     *** THE FOLLOWING VARIDS ARE INVALID IN OPERATION DEFINITION ***
(OUT)      N.1
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      n1  + n2  = case n3 is
(OUT)
(OUT)     *** INPUT IS IGNORED AT :      IS
(OUT)     *** THE FOLLOWING CASE-VARIABLE IS NOT DECLARED ***
(OUT)      N3
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      n1  + n2  = case n1 is
(OUT)     * ZERO :
(OUT)
(IN)      n2
(OUT)     * SUC ( NA0 ) :
(OUT)
(IN)      (na0 + suc(n2))
(OUT)     ESAC
(OUT)     ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)      n1 - n2 = case n2 is zero : zero
```

```
(OUT)
(OUT)    *** INPUT IS IGNORED AT :      : ZERO
(OUT)    *** MISSING ' * ' IN LEFTSIDE OF CASE-SCHEME ***
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     n1 - n2 = case n2 is * zero : zero
(OUT)    * SUC ( NA0 ) :
(OUT)
(IN)     (pred(n1) * na0)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :      NA0 )
(OUT)    *** THE FOLLOWING OPID-PARTS ARE INVALID IN TERM ***
(OUT)     *
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     n1  + n2  = case n1 is
(OUT)
(OUT)    *** INPUT IS IGNORED AT :      N2 = CASE N1 IS
(OUT)    *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)    +
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     n1 - n2 = case n2 is * zero : zero
(OUT)    * SUC ( NA0 ) :
(OUT)
(IN)     (pred(n1) - na0)
(OUT)    ESAC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     mult(n1,n2) =case n1 is
(OUT)    * ZERO :
(OUT)
(IN)     ?
(OUT)    OPSCHEME ::= LETSCHEME | CASESCHEME | IFSCHEME | TERM
(OUT)    * ZERO :
(OUT)
(IN)     zero
(OUT)    * SUC ( NA0 ) :
(OUT)
(IN)     case n2 is * zero : zero
(OUT)    * SUC ( NA1 ) :
(OUT)
(IN)     (n1 + mult(n1,na1))
(OUT)    ESAC
(OUT)    ESAC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     n1 / n2 = case n1 is
(OUT)    * ZERO :
(OUT)
(IN)     error-nat
(OUT)    * SUC ( NA0 ) :
```

```
(OUT)
(IN)    case n2 is * zero : zero
(OUT)   * SUC ( NA1 ) :
(OUT)
(IN)    if ((n1 -n2) bool.lt zero)
(OUT)
(OUT)   *** INPUT IS IGNORED AT :    ) BOOL.LT ZERO )
(OUT)   *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)   -N2
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 / n2 = case n1 is
(OUT)   * ZERO :
(OUT)
(IN)    error-nat
(OUT)   * SUC ( NA0 ) :
(OUT)
(IN)    case n2 is * zero : zero
(OUT)   * SUC ( NA1 ) :
(OUT)
(IN)    if ((n1 - n2) bool.lt zero)
(OUT)
(OUT)   *** INPUT IS IGNORED AT :    ZERO )
(OUT)   *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)   BOOL.LT
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 / n2 = case n1 is
(OUT)   * ZERO :
(OUT)
(IN)    zero
(OUT)   * SUC ( NA0 ) :
(OUT)
(IN)    case n3 is
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     IS
(OUT)   *** THE FOLLOWING CASE-VARIABLE IS NOT DECLARED ***
(OUT)    N3
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 / n2 = case n1 is
(OUT)   * ZERO :
(OUT)
(IN)     zero
(OUT)   * SUC ( NA0 ) :
(OUT)
(IN)    case n2 is * zero : zero
(OUT)   * SUC ( NA1 ) :
(OUT)
(IN)    if ((n1 - n2) lt zero)
(OUT)   THEN
(OUT)
```

```
(IN)    ?
(OUT)   OPSCHEME ::= LETSCHEME | CASESCHEME | IFSCHEME | TERM
(OUT)   THEN
(OUT)
(IN)    ?
(OUT)   LETSCHEME ::= LET VARID = TERM [, VARID = TERM] ... IN PP OPSCHEME
(OUT)   THEN
(OUT)
(IN)    zero
(OUT)   ELSE
(OUT)
(IN)    (one + ((n1 - n2) / n2))
(OUT)   ESAC
(OUT)   ESAC
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    exp(n1,n2) = case n2 as *zero true, * suc(n1) : false
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     *ZERO TRUE, * SUC ( N1 ) : FALSE
(OUT)   *** MISSING ' IS ' IN CASE-SCHEME ***
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    exp(n1,n2) = case n2 is *zero : true , * suc(n1) : false
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     : TRUE , * SUC ( N1 ) : FALSE
(OUT)   *** MISSING ' * ' IN LEFTSIDE OF CASE-SCHEME ***
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    exp(n1,n2) = case n2 is * zero : true , * suc(n1) : false
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     , * SUC ( N1 ) : FALSE
(OUT)   *** WRONG TARGETSORT OF TERM IN OPERATION-SCHEME ***
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    exp(n1,n2) = case n2 is * zero : one
(OUT)   * SUC ( NA0 ) :
(OUT)
(IN)    mult(n1,exp(n1,n2))
(OUT)   ESAC
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    zer0?(n) = case n is * zero : true , * suc(n1) : false
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     N ) = CASE N IS * ZERO : TRUE , * S . . .
(OUT)   *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)   %(
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    zero?(n) = case n is * zero : true , * suc(n1) : false
(OUT)   ESAC
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
```

```
(IN)    n1 nat.lt n2 = case n2 is
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     N2 = CASE N2 IS
(OUT)   *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)   NAT.LT
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 lt n2 = case n2 is * suc(n21) : case n1 is
(OUT)   * ZERO :
(OUT)
(IN)    bool.true
(OUT)   * SUC ( NA0 ) :
(OUT)
(IN)    (na0 lt n21)
(OUT)   ESAC
(OUT)   * ZERO :
(OUT)
(IN)    false
(OUT)   ESAC
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 le n2 = bool.not((n2 lt n1))
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 gt n2 = (n22 lt n1)
(OUT)
(OUT)   *** INPUT IS IGNORED AT :    LT N1 )
(OUT)   *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)   N22
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 gt n2 = (n2  lt n1)
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    n1 ge n2 = (n2 le n1) /* greater-equal */
(OUT)   ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)    [ n1,n2 ] = (mult(n1,n2) + n2) /* binary numbers */
(OUT)   endspec
(OUT)   ***************          E N D          I N P U T          ***************
(OUT)   CPU TIME USED : 103468 ms.
(OUT)
(OUT)   POSITION: TOP OF SPEC NAT    LIST
(OUT)
(OUT)   SPEC-EDIT-LEVEL:
(OUT)
(IN)     input
(OUT)   ***************   I N P U T L E V E L     FOR      SPECS     ***************
(OUT)   ***************              VERSION :  02.01.1985           ***************
(OUT)   *** THE CURRENT SPECIFICATION IS YET COMPLETE ***
(OUT)
(OUT)   ***************          E N D          I N P U T          ***************
```

```
(OUT)    CPU TIME USED : 719 ms.
(OUT)    EDITOR-LEVEL ENDED.
(OUT)    FILELEVEL:
(OUT)
(IN)      end
(OUT)    %  D800 ERASE FILE LISP.DATA.HORST.SBQ-LEVEL2.00
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open sbq-level2
(OUT)    FILE OPENED !
(OUT)    MESSAGEFILE EMPTY
(IN)     FILELEVEL:
(OUT)
(IN)      input spec limit
(OUT)             ************************************************************
(OUT)             ***                                                      ***
(OUT)             ***                S P E C  - E D I T O R                ***
(OUT)             ***               VERSION VOM:  10. 12. 1984             ***
(OUT)             ***                                                      ***
(OUT)             ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)             ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN       ***
(OUT)             ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)             ***                                                      ***
(OUT)             ************************************************************
(OUT)    ***************    I N P U T L E V E L      FOR       SPECS     ***************
(OUT)    ***************               VERSION :  02.01.1985             ***************
(OUT)    ENTER COMMENT OR ;
(OUT)
(IN)     /* loose specification of a non-error-natural number as limit */
(OUT)    use
(OUT)    ENTER SPECTERM COMMENT
(OUT)
(IN)     nat ;
(OUT)    sorts
(OUT)    ENTER SORTID COMMENT OR ;
(OUT)
(IN)     ; ops
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     limit : --> nat ;
(OUT)    props
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [label-1 limit =|= error-nat
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     LIMIT =|= ERROR-NAT
(OUT)    *** MISSING ']' AFTER LABEL IN PROPS-CLAUSE ***
(OUT)
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     [label-1] limit =|= error-nat
(OUT)
```

```
(OUT)   *** INPUT IS IGNORED AT :     LIMIT =|= ERROR-NAT
(OUT)   *** THE FOLLOWING LABEL IS INVALID IN PROPS-CLAUSE ***
(OUT)    LABEL-1
(OUT)   ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)    [label1] limit =|= error-nat ; endspec
(OUT)   ***************           E N D           I N P U T         ***************
(OUT)   CPU TIME USED : 9868 ms.
(OUT)
(OUT)   POSITION: TOP OF SPEC LIMIT    LIST
(OUT)   spec LIMIT
(OUT)       /* LOOSE SPECIFICATION OF A NON-ERROR-NATURAL NUMBER AS LIMIT */
(OUT)      use   NAT ;
(OUT)      ops   LIMIT:  --> NAT;
(OUT)      props [LABEL1] LIMIT =|= ERROR-NAT;
(OUT)   endspec
(OUT)
(OUT)   SPEC-EDIT-LEVEL:
(OUT)
(IN)     end
(OUT)   EDITOR-LEVEL ENDED.
(OUT)   FILELEVEL:
(OUT)
(IN)    input spec limited-stack
(OUT)           ************************************************************
(OUT)           ***                                                      ***
(OUT)           ***              S P E C  - E D I T O R                  ***
(OUT)           ***             VERSION VOM:  10. 12. 1984               ***
(OUT)           ***                                                      ***
(OUT)           ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)           ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN      ***
(OUT)           ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)           ***                                                      ***
(OUT)           ************************************************************
(OUT)   ***************   I N P U T L E V E L     FOR      SPECS     ***************
(OUT)   ***************             VERSION :  02.01.1985             ***************
(OUT)   ENTER COMMENT OR ;
(OUT)
(IN)    /* Standard algorithmic definition of a limited-stack.Push on a full stack, pop ot top of an empty
( )     stack result in errors */
(OUT)   use
(OUT)   ENTER SPECTERM COMMENT
(OUT)
(IN)    elem, limit ;
(OUT)   sorts
(OUT)   ENTER SORTID COMMENT OR ;
(OUT)
(IN)    stack ;
(OUT)   ops
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    empty : --> stack
```

```
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     empty?,full : stack-- bool
(OUT)    *** THE OPHEADER, BEGINNING WITH THE FOLLOWING OPIDS, HAS NO COMPLETE DOMAIN-LIST OR CODOMAIN ***
(OUT)     EMPTY?  FULL  :
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     empty?,full : stack --> bool
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     push : stack elem --> stck, pop : stack --> stack
(OUT)    *** INPUT WAS IGNORED BEGINNING AT :  , POP : STACK -- > STACK
(OUT)    *** THE FOLLOWING SORTIDS ARE NEITHER IN THE INTERFACE NOR IN SORTS-CLAUSE ***
(OUT)     STCK
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     push : stack elem --> stack, pop : stack --> stack
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     top : stack --> elem
(OUT)    ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)     lt,gt : stack stack --> bool ;
(OUT)    props
(OUT)    ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    spec-body
(OUT)    constructors
(OUT)    ENTER CONSTRUCTOR COMMENT
(OUT)
(IN)     empty, push ;
(OUT)    auxiliaries
(OUT)    ENTER AUXILIARY COMMENT OR ;
(OUT)
(IN)     depth: stack --> nat ;
(OUT)    !!! SPESY CAN'T DECLARE THE EQ-OPERATION FOR THE FOLLOWING SORTS !!!
(OUT)    STACK
(OUT)    define-auxiliaries
(OUT)    ENTER AUXILIARY-DEFINITION COMMENT
(OUT)
(IN)     depth(st) = case st is
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    OPSCHEME ::= LETSCHEME | CASESCHEME | IFSCHEME | TERM
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    LETSCHEME ::= LET VARID = TERM [, VARID = TERM] ... IN PP OPSCHEME
(OUT)    * EMPTY :
(OUT)
```

```
(IN)    ?
(OUT)   CASESCHEME :: = CASE VARID IS
(OUT)                       PP * LEFTPART : OPSCHEME
(OUT)                       [, PP * LEFTPART : OPSCHEME] ...
(OUT)                       [, OTHERWISE : OPSCHEME]
(OUT)                   PP ESAC
(OUT)   * EMPTY :
(OUT)
(IN)    ?
(OUT)   IFSCHEME ::= IF TERM
(OUT)                PP THEN OPSCHEME
(OUT)                [ELSEIF TERM PP THEN OPSCHEME] ...
(OUT)                PP ELSE OPSCHEME
(OUT)   * EMPTY :
(OUT)
(IN)    ?
(OUT)   TERM        ::= (TERM) | VARID | PREFIX-TERM | MIXFIX-TERM
(OUT)   PREFIX-TERM ::= OPID-PART [ (TERM [, TERM] ... )]
(OUT)   MIXFIX-TERM ::= [[ TERM [, TERM ] ... B ] OPID-PART B ] ...
(OUT)                   [ TERM [, TERM ] ... B ] OPID-PART
(OUT)                   [ B TERM [, TERM ] ... ]
(OUT)   * EMPTY :
(OUT)
(IN)    ?
(OUT)   *** NO MORE DETAILED INFORMATION AVAILAIBLE ***
(OUT)   * EMPTY :
(OUT)
(IN)    nat.zero
(OUT)   * PUSH ( ST0 , EL0 ) :
(OUT)
(IN)    suc(depth(st0))
(OUT)   ESAC
(OUT)   define-carriers
(OUT)   ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)    is-stuck(st) =
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     ( ST ) =
(OUT)   *** INVALID CHARACTERISTIC PREDICATE IN DEFINE-CARRIERS-CLAUSE ***
(OUT)    IS-STUCK
(OUT)   ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)    is-stack(st) = if st then
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     ST THEN
(OUT)   *** MISSING CASE IN DEFINITION OF CHARACTERISTIC PREDICATE ***
(OUT)
(OUT)   ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)    is-stack(st) = case st is
(OUT)   * PUSH ( ST0 , EL0 ) :
(OUT)   IF ( NOT (IS-STACK (ST0)) )
```

```
(OUT)    THEN FALSE
(OUT)    ELSE
(OUT)
(IN)     (depht(st0) lt limit)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    ( ST0 ) LT LIMIT )
(OUT)    *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)    DEPHT
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : false
(OUT)
(OUT)    *** MISSING IF-SCHEME IN DEFINITION OF CHARACTERISTIC PREDICATE AFTER THE FOLLOWING CONSTRUCTOR ***
(OUT)     PUSH
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if ( st and el0 ) then
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     AND EL0 ) THEN
(OUT)    *** MISSING ' NOT ' IN IF-SCHEME IN THE DEFINITION OF CHARACTERISTIC PREDICATE ***
(OUT)
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if (not(is-nat(el0)) then
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     ( EL0 ) ) THEN
(OUT)    *** INVALID CHARACTERISTIC PREDICATE IN DEFINE-CARRIERS-CLAUSE ***
(OUT)     IS-NAT
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if (not(is-stack(el0)) then
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     ) ) THEN
(OUT)    *** UNKNOWN VARID AS ARGUMENT OF CHARACTERISTIC PREDICATE ***
(OUT)     EL0
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if (not(is-stack(st0)) then true
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     THEN TRUE
(OUT)    *** MISSING ' ) ' AFTER TERM IN IF-SCHEME ***
(OUT)
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if (not(is-stack(st0))) then true
(OUT)
(OUT)    *** MISSING ' FALSE ' IN IF-SCHEME IN DEFINITION OF CHARACTERISTIC PREDICATE ***
(OUT)
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) = case st is * push(st0,el0) : if (not(is-stack(st0))) then false
(OUT)
```

```
(OUT)   *** MISSING ' ELSE ' IN IF-SCHEME ***
(OUT)   ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)    is-stack(st) = case st is * push(st0,el0) : if (not(is-stack(st0))) then false
(IN)     else (depth(st0) nat.lt limit)
(OUT)   ENTER THE MISSING CASE-PARTS OR 'OTHERWISE'
(OUT)
(IN)    otherwise : true
(OUT)   ESAC
(OUT)   define-constructor-ops
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT
(OUT)
(IN)    empty = empty
(OUT)
(OUT)   *** MISSING ' * ' IN CONSTRUCTOR DEFINITION ***
(OUT)
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT
(OUT)
(IN)    empty = * empty
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT OR ;
(OUT)
(IN)    push(st,e) =
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     =
(OUT)   *** THE CONSTRUCTOR-VARIABLES MUST BE THE IDENTICAL TO THE VARIABLES ENTERED IN THE DEFINE-CARR
(OUT)   IERS-CLAUSE ***
(OUT)
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT OR ;
(OUT)
(IN)    push(st0,el0) = case st0 is
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     ST0 IS
(OUT)   *** MISSING ' IF ' IN CONSTRUCTOR DEFINITION ***
(OUT)
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT OR ;
(OUT)
(IN)    push(st0,el0) = if  st0 then
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     THEN
(OUT)   *** INVALID TERM IN CONSTRUCTOR DEFINITION ***
(OUT)
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT OR ;
(OUT)
(IN)    push(st0,el0) = if  (depth(st0) nat.lt limit) then st0 else
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     ELSE
(OUT)   *** MISSING ' * ' IN CONSTRUCTOR DEFINITION ***
(OUT)
(OUT)   ENTER CONSTRUCTOR-DEFINITION COMMENT OR ;
(OUT)
(IN)    push(st0,el0) = if  (depth(st0) nat.lt limit) then * push(st0,el0) else error-stack
(OUT)   private-ops
```

```
(OUT)    ENTER PRIVATE-OPERATION COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    define-ops
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     ;
(OUT)
(OUT)    *** FIRST YOU MUST DEFINE THE FOLLOWING OPERATIONS ***
(OUT)     EMPTY?  FULL  POP  TOP  LT  GT
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     full?(st) = nat((depth(st) lt limit)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     ST ) = NAT ( ( DEPTH ( ST ) LT LIMIT )
(OUT)    *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)    %(
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     full(st) = nat((depth(st) lt limit)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    ( ( DEPTH ( ST ) LT LIMIT )
(OUT)    *** THE FOLLOWING SYMBOL IS NEITHER A VALID VARIABLE OR CONSTANT , NOR OPID ***
(OUT)    NAT
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     full(st) = bool.not((depth(st) lt limit)
(OUT)
(OUT)    *** INPUT IS IGNORED AT :    LIMIT )
(OUT)    *** WRONG QUANTITY OF ARGUMENTS OR INVALID SORT(S) OF THE ARGUMENTS FOR THE FOLLOWING OPERATION ***
(OUT)    LT
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     full(st) = bool.not((depth(st) nat.lt limit)
(OUT)
(OUT)    *** MISSSING CLOSING-PARENTHESIS BEHIND THE FOLLOWING PREFIX-OPID ***
(OUT)    BOOL.NOT
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     full(st) = bool.not((depth(st) nat.lt limit))
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     empty?(st) = case st is
(OUT)    * EMPTY :
(OUT)
(IN)     true otherwise : false
(OUT)    ESAC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     pop(limit) = case limit is
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     ) = CASE LIMIT IS
```

```
(OUT)
(OUT)    *** THE FOLLOWING VARIABLE IS IN CONFLICT WITH OPID-PARTS OF THE EXPORTED INTERFACE , AUXILIARY-CLA
( )      USE OR PRIVATE-OPS-CLAUSE ***
(OUT)     LIMIT
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     pop(st) = case st is * push(st1,el1) : st1 otherwise : error-stack
(OUT)    ESAC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     top(st) = case st is
(OUT)    * EMPTY :
(OUT)
(IN)     error-elem
(OUT)    * PUSH ( ST0 , EL0 ) :
(OUT)
(IN)     el0
(OUT)    ESAC
(OUT)    ENTER OP-DEFINITION COMMENT OR ;
(OUT)
(IN)     end
(OUT)    **********              E N D            I N P U T          **********
(OUT)    CPU TIME USED : 124218 ms.
(OUT)
(OUT)    POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-STACK    LIST
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      + +
(OUT)       ops   EMPTY:  --> STACK
(OUT)             EMPTY?,FULL: STACK  --> BOOL
(OUT)             PUSH: STACK ELEM  --> STACK
(OUT)             POP: STACK  --> STACK
(OUT)             TOP: STACK  --> ELEM
(OUT)             LT,GT: STACK STACK  --> BOOL;
(OUT)
(OUT)    POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-STACK    LIST
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      spec
(OUT)
(OUT)    POSITION: TOP OF SPEC LIMITED-STACK    LIST
(OUT)    spec LIMITED-STACK
(OUT)        /* STANDARD ALGORITHMIC DEFINITION OF A LIMITED-STACK.PUSH ON A FULL   */
(OUT)        /* STACK, POP OR TOP OF AN EMPTY STACK RESULT IN ERRORS */
(OUT)       use   ELEM
(OUT)             LIMIT ;
(OUT)       sorts STACK;
(OUT)       ops   EMPTY:  --> STACK
(OUT)             EMPTY?,FULL: STACK  --> BOOL
(OUT)             PUSH: STACK ELEM  --> STACK
```

```
(OUT)            POP: STACK  --> STACK
(OUT)            TOP: STACK  --> ELEM
(OUT)            LT,GT: STACK STACK  --> BOOL;
(OUT)    spec-body
(OUT)       constructors EMPTY
(OUT)                    PUSH;
(OUT)       auxiliaries DEPTH: STACK  --> NAT;
(OUT)       define-auxiliaries
(OUT)           DEPTH(ST) = case ST is
(OUT)                         * EMPTY : NAT.ZERO
(OUT)                         * PUSH(ST0,EL0) : SUC(DEPTH(ST0))
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)                       esac;
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      + +
(OUT)       define-constructor-ops
(OUT)           EMPTY = * EMPTY
(OUT)           PUSH(ST0,EL0) = if (DEPTH(ST0) NAT.LT LIMIT)
(OUT)                           then * PUSH(ST0,EL0)
(OUT)                           else ERROR-STACK;
(OUT)       define-ops
(OUT)           FULL(ST) = BOOL.NOT((DEPTH(ST) NAT.LT LIMIT))
(OUT)           EMPTY?(ST) = case ST is
(OUT)                         * EMPTY : TRUE
(OUT)                         otherwise FALSE
(OUT)                        esac
(OUT)           POP(ST) = case ST is
(OUT)                       * PUSH(ST1,EL1) : ST1
(OUT)                       otherwise ERROR-STACK
(OUT)                     esac
(OUT)           TOP(ST) = case ST is
(OUT)                       * EMPTY : ERROR-ELEM
(OUT)                       * PUSH(ST0,EL0) : EL0
(OUT)                     esac;
(OUT)    endspec
(OUT)
(OUT)    POSITION: TOP OF SPEC LIMITED-STACK    LIST
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
(IN)      end
(OUT)    EDITOR-LEVEL ENDED.
(OUT)    FILELEVEL:
(OUT)
(IN)      end
(OUT)    % D800 ERASE FILE LISP.DATA.HORST.SBQ-LEVEL2.00
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open sbq-level2
```

```
(OUT)    FILE OPENED !
(OUT)    MESSAGEFILE EMPTY
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-stack
(OUT)    ENTER INFORMATIONPARAMETER :
(OUT)
(IN)      interface
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  LIMITED-STACK  **
(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 NAT :
(OUT)       NAT
(OUT)    operations from NAT :
(OUT)       ZERO:  --> NAT
(OUT)       SUC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       MULT: NAT NAT --> NAT
(OUT)       EXP: NAT NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)       _-_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)       _/_: NAT NAT --> NAT
(OUT)       ZERO?: NAT --> BOOL
(OUT)       _LT_: NAT NAT --> BOOL
(OUT)       _LE_: NAT NAT --> BOOL
(OUT)       _GT_: NAT NAT --> BOOL
(OUT)       _GE_: NAT NAT --> BOOL
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)       [__]: NAT NAT --> NAT
(OUT)       ONE:  --> NAT
(OUT)       TWO:  --> NAT
(OUT)       THREE:  --> NAT
(OUT)       FOUR:  --> NAT
(OUT)       FIVE:  --> NAT
(OUT)       SIX:  --> NAT
(OUT)       SEVEN:  --> NAT
(OUT)       EIGTH:  --> NAT
(OUT)       NINE:  --> NAT
(OUT)       TEN:  --> NAT
(OUT)
(OUT)    no sorts from LIMIT
(OUT)    operations from LIMIT :
```

```
(OUT)       LIMIT:  --> NAT
(OUT)    000045 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    sorts from ELEM :
(OUT)       ELEM
(OUT)    no operations from ELEM
(OUT)
(OUT)    sorts from LIMITED-STACK :
(OUT)       STACK
(OUT)    operations from LIMITED-STACK :
(OUT)       EMPTY:  --> STACK
(OUT)       EMPTY?: STACK --> BOOL
(OUT)       FULL?: STACK --> BOOL
(OUT)       PUSH: STACK ELEM --> STACK
(OUT)       POP: STACK --> STACK
(OUT)       TOP: STACK --> ELEM
(OUT)       LT,GT: STACK STACK --> BOOL
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-stack ok
(OUT)             status of specification:   LIMITED-STACK
(OUT)
(OUT)                                 syntax: +                    semantics
(OUT)                                                                  |
(OUT)                                                                  |— terminating:
(OUT)                                                                  |— consistent :
(OUT)                                                                  |— derivable  :
(OUT)             interface: +                          rest: +
(OUT)
(OUT)
(OUT)
(OUT)               use:   +          |                   |                   |
(OUT)               sorts: +     props: 0+   constructors: +   def.constructors: +
(OUT)               ops:   +                 auxiliaries : +   private ops     : 0+
(OUT)                                        def.-aux.   : +   define ops      : +
(OUT)                                        def.-carrier: +
(OUT)
(OUT)                   0 = empty
(OUT)                   + = ok
(OUT)                   - = not ok
(OUT)                   ? = unknown
(OUT)             FILELEVEL:
(OUT)
(IN)      end
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open sbq-level3
(OUT)    FILE OPENED !
(OUT)    MESSAGEFILE EMPTY
(OUT)    FILELEVEL:
(OUT)
```

```
(IN)     dir  sbq-level3 **
(OUT)    UNKNOWN OBJECTTYPE. COMMAND BROKEN.
(OUT)    FILELEVEL:
(OUT)
(IN)     dir * *
(OUT)    SPECS ON SBQ-LEVEL4       :   NAT101
(OUT)    MAPS ON SBQ-LEVEL4        :   ELEM.NAT_IS   POINTED-ELEM.NAT_POINT0   LIMIT.NAT101_LIMIT100
(OUT)       LIMIT+1.NAT101_LIMIT100
(OUT)
(OUT)    SPECS ON RLBASE           :   BOOLEAN   INTEGER   INT
(OUT)    MAPS ON RLBASE           :THERE ARE NO MAPS!
(CUT)
(OUT)    SPECS ON SBQ-LEVEL3       :   LIMITED-QUEUE   LIMIT+1   POINTED-ELEM   STACK-SIMULATION
(OUT)    MAPS ON SBQ-LEVEL3        :   LIMIT.LIMIT+1_INCREASE   ELEM.POINTED-ELEM_EXTEND
(OUT)       LIMITED-STACK.STACK-SIMULATION_SIMULATE
(OUT)
(OUT)    SPECS ON SBQ-LEVEL2       :   NAT   LIMIT   LIMITED-STACK
(OUT)    MAPS ON SBQ-LEVEL2        :   LIMITED-LIFO.LIMITED-STACK_FIX
(OUT)
(OUT)    SPECS ON SBQ-LEVEL1       :   ELEM   LIMITED-LIFO
(OUT)    MAPS ON SBQ-LEVEL1       :THERE ARE NO MAPS!
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)     list nat total
(OUT)    spec NAT
(OUT)        /* STANDARD ALGORITHMIC DEFINITION OF THE NATURAL NUMBERS */
(OUT)       use   BOOL ;
(OUT)       sorts NAT;
(OUT)       ops   ZERO:  --> NAT
(OUT)             SUC,PRED: NAT  --> NAT
(OUT)             _+_,_-_,MULT,_/_,EXP: NAT NAT  --> NAT
(OUT)             ZERO?: NAT  --> BOOL
(OUT)             _LT_,_LE_,_GT_,_GE_: NAT NAT  --> BOOL
(OUT)             EQ-NAT: NAT NAT  --> BOOL
(OUT)             [__]: NAT NAT  --> NAT
(OUT)             ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGTH,NINE,TEN:  --> NAT;
(OUT)       props [NAT1] ALL X,Y:NAT   ([ X,Y ]) == (MULT(X,TEN) + Y);
(OUT)    spec-body
(OUT)       constructors ZERO
(OUT)                    SUC;
(OUT)       auxiliaries EQ-NAT: NAT NAT  --> BOOL;
(OUT)       define-auxiliaries
(OUT)           EQ-NAT(N,0) = case N is
(OUT)                           * ZERO : case 0 is
(OUT)                                      * ZERO : TRUE
(OUT)                                      otherwise FALSE
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)                                   esac
(OUT)                           * SUC(N0) : case 0 is
(OUT)                                       * SUC(00) : EQ-NAT(N0,00)
(OUT)                                       otherwise FALSE
```

```
(OUT)                                        esac
(OUT)                          esac;
(OUT)       define-carriers
(OUT)           IS-NAT(N) = TRUE;
(OUT)       define-constructor-ops
(OUT)           ZERO = * ZERO
(OUT)           SUC(NA0) = * SUC(NA0);
(OUT)       define-ops
(OUT)           ONE = SUC(ZERO)
(OUT)           TWO = SUC(ONE)
(OUT)           THREE = SUC(TWO)
(OUT)           FOUR = SUC(THREE)
(OUT)           FIVE = SUC(FOUR)
(OUT)           SIX = SUC(SUC(SUC(SUC(SUC(SUC(ZERO))))))
(OUT)           SEVEN = SUC(SIX)
(OUT)           EIGTH = SUC(SEVEN)
(OUT)           NINE = SUC(EIGTH)
(OUT)           TEN = SUC(NINE)
(OUT)    000045 RECORDS PRINTED. CONTINUE?
(OUT)           PRED(N) = case N is
(OUT)                       * ZERO : ERROR-NAT
(OUT)                       * SUC(NA0) : NA0
(OUT)                     esac
(OUT)           N1 + N2 = case N1 is
(OUT)                       * ZERO : N2
(OUT)                       * SUC(NA0) : (NA0 + SUC(N2))
(OUT)                     esac
(OUT)           N1 - N2 = case N2 is
(OUT)                       * ZERO : ZERO
(OUT)                       * SUC(NA0) : (PRED(N1) - NA0)
(OUT)                     esac
(OUT)           MULT(N1,N2) = case N1 is
(OUT)                           * ZERO : ZERO
(OUT)                           * SUC(NA0) : case N2 is
(OUT)                                          * ZERO : ZERO
(OUT)                                          * SUC(NA1) : (N1 + MULT(N1,NA1))
(OUT)                                        esac
(OUT)                         esac
(OUT)           N1 / N2 = case N1 is
(OUT)                       * ZERO : ZERO
(OUT)                       * SUC(NA0) : case N2 is
(OUT)    000067 RECORDS PRINTED. CONTINUE?
(OUT)                                    * ZERO : ZERO
(OUT)                                    * SUC(NA1) : if (((N1 - N2)) LT ZERO)
(OUT)                                                 then ZERO
(OUT)                                                 else (ONE + ((((N1 - N2)) / N2)))
(OUT)                                  esac
(OUT)                     esac
(OUT)           EXP(N1,N2) = case N2 is
(OUT)                          * ZERO : ONE
(OUT)                          * SUC(NA0) : MULT(N1,EXP(N1,N2))
(OUT)                        esac
```

```
(OUT)          ZERO?(N) = case N is
(OUT)                       * ZERO : TRUE
(OUT)                       * SUC(N1) : FALSE
(OUT)                     esac
(OUT)          N1 LT N2 = case N2 is
(OUT)                       * SUC(N21) : case N1 is
(OUT)                                      * ZERO : BOOL.TRUE
(OUT)                                      * SUC(NA0) : (NA0 LT N21)
(OUT)                                    esac
(OUT)                       * ZERO : FALSE
(OUT)                     esac
(OUT)    000088 RECORDS PRINTED. CONTINUE?
(OUT)          N1 LE N2 = BOOL.NOT((N2 LT N1))
(OUT)          N1 GT N2 = (N2 LT N1)
(OUT)          [ N1,N2 ] = (MULT(N1,N2) + N2);
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-queue total
(OUT)    spec LIMITED-QUEUE
(OUT)        /* ALGORITHMIC SPECIFICATION OF A LIMITED QUEUE. ENQUEUE IN A FULL     */
(OUT)        /* QUEUE,DEQUEUE OR FRONT ON AN EMPTY QUEUE RESULT IN ERRORS */
(OUT)       use   ELEM
(OUT)             LIMIT ;
(OUT)       sorts QUEUE;
(OUT)       ops   NEWQ:  --> QUEUE
(OUT)             NEWQ?,FULLQ?: QUEUE  --> BOOL
(OUT)             ENQ: QUEUE ELEM  --> QUEUE
(OUT)             DEQ: QUEUE  --> QUEUE
(OUT)             FRONT: QUEUE  --> ELEM;
(OUT)    spec-body
(OUT)       constructors ENQ
(OUT)                    NEWQ;
(OUT)       auxiliaries LENGTH: QUEUE  --> NAT;
(OUT)       define-auxiliaries
(OUT)           LENGTH(Q) = case Q is
(OUT)                         * ENQ(QU0,EL0) : SUC(LENGTH(QU0))
(OUT)                         * NEWQ : ZERO
(OUT)                       esac;
(OUT)       define-carriers
(OUT)           IS-QUEUE(Q) = case Q is
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)                           * ENQ(QU0,EL0) : if NOT(IS-QUEUE(EL0))
(OUT)                                            then FALSE
(OUT)                                            else (LENGTH(QU0) LT LIMIT)
(OUT)                           otherwise TRUE
(OUT)                         esac;
(OUT)       define-constructor-ops
(OUT)           NEWQ = * NEWQ
(OUT)           ENQ(QU0,EL0) = if (LENGTH(QU0) LT LIMIT)
(OUT)                          then * ENQ(QU0,EL0)
(OUT)                          else ERROR-QUEUE;
```

```
(OUT)       define-ops
(OUT)           NEWQ?(Q) = case Q is
(OUT)                        * ENQ(QU0,EL0) : FALSE
(OUT)                        * NEWQ : TRUE
(OUT)                      esac
(OUT)           FULLQ?(Q) = NOT((LENGTH(Q) LT LIMIT))
(OUT)           DEQ(Q) = case Q is
(OUT)                      * ENQ(QU0,EL0) : case QU0 is
(OUT)                                         * ENQ(QU1,EL1) : ENQ(DEQ(QU0),EL0)
(OUT)                                         * NEWQ : NEWQ
(OUT)                                       esac
(OUT)                      * NEWQ : ERROR-QUEUE
(OUT)    000045 RECORDS PRINTED. CONTINUE?
(OUT)                    esac
(OUT)           FRONT(Q) = case Q is
(OUT)                        * NEWQ : ERROR-ELEM
(OUT)                        * ENQ(QU0,EL0) : case QU0 is
(OUT)                                           * NEWQ : EL0
(OUT)                                           * ENQ(QU1,EL1) : FRONT(QU0)
(OUT)                                         esac
(OUT)                      esac;
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      list limit+1 total
(OUT)    spec LIMIT+1
(OUT)        /* AXIOMATIC DEFINITION OF LIMIT + 1 */
(OUT)       use   LIMIT ;
(OUT)       ops   SUC-OF-LIMIT:  --> NAT;
(OUT)       props [PROP1] SUC-OF-LIMIT == SUC(LIMIT);
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      list pointed-elem total
(OUT)    spec POINTED-ELEM
(OUT)        /* LOOSE SPECIFICATION OF A NON-ERROR CONSTANT OF SORT ELEM TOGETHER   */
(OUT)        /* WITH AN IDENTIFICATION-TEST */
(OUT)       use   ELEM ;
(OUT)       ops   POINT:  --> ELEM
(OUT)             POINT?: ELEM  --> BOOL;
(OUT)       props [PROP1] ALL E:ELEM   POINT?(E) <==> E == POINT
(OUT)             [PROP2] POINT =|= ERROR-ELEM;
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      dir sbq-level3 map
(OUT)    MAPS ON SBQ-LEVEL3       :   LIMIT.LIMIT+1_INCREASE   ELEM.POINTED-ELEM_EXTEND
(OUT)       LIMITED-STACK.STACK-SIMULATION_SIMULATE
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list limit.limit+1_increase total
```

```
(OUT)    map (LIMIT -INCREASE-> LIMIT+1)
(OUT)        /* REPLACE LIMIT BY LIMIT+1 */
(OUT)       is REFINEMENT;
(OUT)       base
(OUT)           NAT ;
(OUT)       ops   LIMIT = SUC-OF-LIMIT;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      list elem.pointed-elem_extend total
(OUT)    map (ELEM -EXTEND-> POINTED-ELEM)
(OUT)        /* THE INCLUSION OF ELEM IN POINTED-ELEM */
(OUT)       is REFINEMENT;
(OUT)       base
(OUT)           BOOL ;
(OUT)       sorts ELEM = ELEM;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-stack.stack-simulation_simulate total
(OUT)    map (LIMITED-STACK -SIMULATE-> STACK-SIMULATION)
(OUT)        /* MAPS STACK TO QUEUE AND THE STACK-OPERATIONS TO THEIR SIMULATING    */
(OUT)        /* QUEUE OPERATIONS. THIS MAP INCLUDES THE EXTENSION OF ELEM TO       */
(OUT)        /* POINTED-ELEM; BUT IT MAPS LIMIT TO ITSELF - SINCE THE INCREMENT IS */
(OUT)        /* NOT USED ! */
(OUT)       is IMPLEMENTATION;
(OUT)       base
(OUT)           LIMIT ;
(OUT)       use   (ELEM -EXTEND-> POINTED-ELEM);
(OUT)       sorts STACK = QUEUE;
(OUT)       ops   EMPTY = S-EMPTY
(OUT)             PUSH = ENQ
(OUT)             EMPTY? = S-EMPTY?
(OUT)             FULL? = FULLQ?
(OUT)             POP = S-POP
(OUT)             TOP = S-TOP;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      dir sbq-level4 spec
(OUT)    SPECS ON SBQ-LEVEL4       :   NAT101
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list nat101 total
(OUT)    OBJECT NOT FOUND. COMMAND BROKEN.
(OUT)    FILELEVEL:
(OUT)
(IN)      end
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open sbq-level4
```

```
(OUT)    FILE OPENED !
(OUT)    MESSAGEFILE EMPTY
(OUT)    FILELEVEL:
(OUT)
(IN)      list nat101 total
(OUT)    spec NAT101
(OUT)        /* ENRICHES NAT BY DERIVED CONSTANTS */
(OUT)       use   NAT ;
(OUT)       ops   HUNDRED,HUNDRED-AND-ONE:  --> NAT;
(OUT)    spec-body
(OUT)       define-ops
(OUT)           HUNDRED = ([ TEN,ZERO ])
(OUT)           HUNDRED-AND-ONE = ([ TEN,ONE ]);
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      dir sbq-level4 map
(OUT)    MAPS ON SBQ-LEVEL4       :   ELEM.NAT_IS   POINTED-ELEM.NAT_POINT0   LIMIT.NAT101_LIMIT100
(OUT)       LIMIT+1.NAT101_LIMIT100
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list elem.nat_is total
(OUT)    map (ELEM -IS-> NAT)
(OUT)        /* CHOOSES NAT AS ELEM */
(OUT)       is REFINEMENT;
(OUT)       base
(OUT)           BOOL ;
(OUT)       sorts ELEM = NAT;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      list pointed-elem.nat_point0 total
(OUT)    map (POINTED-ELEM -POINT0-> NAT)
(OUT)        /* CHOOSES 0 AS POINT */
(OUT)       is REFINEMENT;
(OUT)       use   (ELEM -IS-> NAT);
(OUT)       ops   POINT = ZERO
(OUT)             POINT? = ZERO?;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      list limit.nat101_limit100   total
(OUT)    map (LIMIT -LIMIT100-> NAT101)
(OUT)        /* CHOOSES 100 AS LIMIT */
(OUT)       is REFINEMENT;
(OUT)       base
(OUT)           NAT ;
(OUT)       ops   LIMIT = HUNDRED;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
```

```
(IN)      list limit+1.nat101_limit100 total
(OUT)    map (LIMIT+1 -LIMIT100-> NAT101)
(OUT)        /* CONSEQUENTLY CHOOSES 101 FOR SUC-OF-LIMIT */
(OUT)       is REFINEMENT;
(OUT)       use   (LIMIT -LIMIT100-> NAT101);
(OUT)       ops   SUC-OF-LIMIT = HUNDRED-AND-ONE;
(OUT)    endmap
(OUT)    FILELEVEL:
(OUT)
(IN)      end
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      end
```