

TUM

INSTITUT FÜR INFORMATIK

The Specification Language SPECTRUM Core Language Report V1.0

Radu Grosu, Dieter Nazareth



TUM-I9429
August 1994

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-8-1994-I9429-350/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©1994 MATHEMATISCHES INSTITUT UND
INSTITUT FÜR INFORMATIK
TECHNISCHE UNIVERSITÄT MÜNCHEN

Typescript: ---

Druck: Mathematisches Institut und
 Institut für Informatik der
 Technischen Universität München

The Specification Language SPECTRUM Core Language Report V1.0*

R. Grosu, D. Nazareth

Fakultät für Informatik, Technische Universität München
80290 München, Germany

E-Mail: {grosu,nazareth}@informatik.tu-muenchen.de

*This work is sponsored by the German Ministry of Research and Technology (BMFT) as part of the compound project “KORSO - Korrekte Software” and by the German Research Community (DFG) project SPECTRUM.

Contents

1	Introduction	2
1.1	The Concrete Syntax	2
1.2	The Abstract Syntax	3
1.3	The Translation Rules	4
2	Lexical Syntax	5
3	The Core Language	7
3.1	In the Small	7
3.1.1	Context Free Syntax	7
3.1.2	Semantic Objects	8
3.1.3	Translation Rules	11
3.1.4	The Sort Inference	15
3.1.5	Class Inference	16
3.1.6	Infix Symbols	17
3.2	In the Large	17
3.2.1	Context Free Syntax	17
3.2.2	Compound Semantic Objects	18
3.2.3	Translation Rules	19
A	Context Free Syntax	27
A.1	In the Small	27
A.2	In the Large	28

Chapter 1

Introduction

This report formally describes the core syntax of the specification language SPECTRUM. It does not explain the particular language constructs but only gives an exact definition of the syntax. An informal introduction to SPECTRUM can be found in [BFG⁺93a, BFG⁺93b]. The formal semantics is given in [GR94]. Thus, in the sequel we assume that the reader is familiar with the concepts used in SPECTRUM. As usual this description distinguishes between the *concrete syntax* and the *abstract syntax*.

1.1 The Concrete Syntax

The concrete syntax treats the language as a *set of strings* over an alphabet of symbols. Concrete syntax is usually specified by a *context free grammar* that gives *productions* for generating strings of symbols, using auxiliary *nonterminal* symbols.

It is common practice to distinguish a *lexical level* and a *phrase level* in concrete syntax. The terminal symbols in the grammar specifying the lexical level are single characters; those in the phrase-level grammar are the *nonterminal* symbols in the lexical grammar.

For example the following rules¹

$$\begin{aligned}\langle \text{sortexp} \rangle & ::= \langle \text{sortexp1} \rangle \rightarrow \langle \text{sortexp} \rangle \\ \langle \text{sortexp1} \rangle & ::= \langle \text{alphanumeric} \rangle | (\langle \text{sortexp} \rangle)\end{aligned}$$

with nonterminals $\langle \text{sortexp} \rangle$, $\langle \text{sortexp1} \rangle$ and $\langle \text{alphanumeric} \rangle$ define possibly parenthesized arrow sorts. The sort identifiers $\langle \text{alphanumeric} \rangle$ are defined by the lexical syntax:

$$\begin{aligned}\langle \text{alphanumeric} \rangle & ::= \langle \text{letter} \rangle | \langle \text{digit} \rangle | _ \\ \langle \text{letter} \rangle & ::= \mathbf{a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u} \\ & \quad | \mathbf{v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M} \\ & \quad | \mathbf{N|O|P|Q|R|S|T|U|V|W|X|Y|Z} \\ \langle \text{digit} \rangle & ::= \mathbf{0|1|2|3|4|5|6|7|8|9}\end{aligned}$$

The concrete syntax of SPECTRUM is presented as an EBNF-like grammar. The notations used are summed up below:

¹These rules are simplified versions of the Spectrum rules.

$[rhs]$	rhs is optional
$\{rhs\}^*$	zero or more repetitions of rhs
$\{rhs // sep\}^*$	zero or more repetitions of rhs separated by sep
$\{rhs\}^+$	one or more repetitions of rhs
$\{rhs // sep\}^+$	one or more repetitions of rhs separated by sep
$\{rhs\}$	grouping
$rhs_1 rhs_2$	choice
$rhs_{\overline{\{rhs\}}}$	difference: elements generated by rhs except those generated by \overline{rhs}
terminal	terminal syntax is given in boldface
$\langle nonterminal \rangle$	nonterminals are enclosed in angle brackets
$\langle nonterminal \rangle$	emphasized nonterminals are not defined in the grammar but represent a non-printable letter of the ASCII character set or are given informally

Remark: It is important to distinguish the meta-symbols $[,], \{, \}, (,)$ and $|$ introduced above from the corresponding terminal symbols $[,], \{, \}, (,)$ and $|$ printed in boldface font.

To each nonterminal we associate a *phrase sort* containing all expressions which can be inferred from that nonterminal.

Notation: For each nonterminal $\langle nonterm \rangle$ its phrase sort is written as $Nonterm$ and we use $nonterm$ (possibly indexed) to range over $Nonterm$.

1.2 The Abstract Syntax

The *abstract syntax* treats the language as a *set of trees* each representing the structure of a parsed specification text. The important thing about trees is that unlike strings, their compositional structure is *inherently* unambiguous: there is only one way of constructing a particular tree out of its (immediate) sub-trees.

Given a concrete sort expression *sortexp* or a concrete expression *exp* we use $abs(sortexp)$ and $abs(exp)$ respectively, to denote their corresponding abstract trees. For the other abstract syntax objects we use a mathematical notation based on set theory which is summarized below:

$A \times B$	The Cartesian product of A and B . $A^k = \underbrace{A \times \dots \times A}_k$
$A \xrightarrow{fin} B$	The set of finite mappings from A to B . A finite map will be written explicitly in the form $\{a_1 \rightarrow b_1, \dots, a_n \rightarrow b_n\}, n \geq 0$; in particular, the empty map is $\{\}$.
$Dom f, Ran f$	The domain and range of a finite map f .
$f + g$	For finite maps f and g , $f + g$ is called <i>f modified by g</i> . It is defined by: $Dom(f + g) = Dom(f) \cup Dom(g)$ $(f + g)(a) =$ if $a \in Dom(g)$ then $g(a)$ else $f(a)$ If $Dom f \cap Dom g = \emptyset$ then $f + g = f \cup g$.
$A \cup B$	The union of A and B .
$A B$	The disjoint union of A and B .
$Fin(A)$	The set of finite subsets of A .
$List(A)$	The set of lists with elements taken from A . We write $a:l$ for a list with first element a and rest l .

For example the following declaration:

$$ct \in ConType = \bigcup_{k \geq 0} (ClassId^k \times ClassId)$$

defines the set of sort constructor signatures as $ConType$ and designate the variable ct (possibly indexed) to range over this set. The sets of identifiers Id , $ClassId$, etc. are taken directly from the concrete syntax. The advantage of the mathematical notation is the automatic provision of very abstract object manipulation primitives.

1.3 The Translation Rules

In general, the meaning of an expression occurring in a specification text is context dependent (e.g. a variable declared with type integer can be used in subsequent text only on an integer position). As a consequence, expressions generated by the context free grammar of the *concrete syntax* are called *rough expressions* because among them there are also erroneous ones. The set of *well formed expressions* i.e. the set of expressions respecting the context dependencies or static semantics (e.g. the typing discipline) are filtered out by the *context sensitive syntax*. We give this syntax by using a formalism based on logic. Specifically, we define concrete expressions and their translation into abstract (or semantic) objects simultaneously using axioms and inference rules. An inference rule has the form:

$$\frac{B \vdash cexp_1 \Rightarrow aexp_1 \dots B \vdash cexp_n \Rightarrow aexp_n}{B \vdash cexp \Rightarrow aexp} \quad \{ \text{side conditions} \}$$

It allows to derive the conclusion $B \vdash cexp \Rightarrow aexp$ if all the assumptions $B \vdash cexp_i \Rightarrow aexp_i$ are true. The expressions $B \vdash cexp \Rightarrow aexp$ with $B = \{id_1 \rightarrow a_1, \dots, id_n \rightarrow a_n\}$ are called *translation assertions*. They intuitively say that if identifiers id_1, \dots, id_n have attributes a_1, \dots, a_n then $cexp$ is well formed and its translation is $aexp$. The *environment* or *context* B in which $cexp$ is translated is the abstract representation of the *symbol table*. Although for convenience, a context B is not always represented by a *finite mapping*, we assume that no id_i occurs twice in B and use B as a finite mapping.

Our contexts are sorted i.e. they are of the form $B_1 \times B_2 \times \dots \times B_n$. They can also be structured i.e. if $id \rightarrow a$ then a can contain itself a context. Identifiers are also called *simple semantic objects*. All other objects from the abstract syntax (including contexts) are also called *compound semantic objects*.

In the translation rules, phrases within single square brackets [] are called *first options*, and those within double square brackets [[]] are called *second options*. To reduce the number of rules we adopt the following convention:

Notation: In each instance of a rule, the first options must be either all present or all absent; similarly the second options must be either all present or all absent.

When we write $[cse \Rightarrow ase]_{def}$ then we assume that def is the default semantic value when the option is missing.

Chapter 2

Lexical Syntax

$\langle \text{spectext} \rangle ::= \{ \langle \text{lexeme} \rangle | \langle \text{whitespace} \rangle | \langle \text{comment} \rangle \}^*$
 $\langle \text{lexeme} \rangle ::= \langle \text{charconst} \rangle | \langle \text{num} \rangle | \langle \text{string} \rangle | \langle \text{alphanumeric} \rangle$
 $\quad | \langle \text{symbolid} \rangle | \langle \text{inf-alphanumeric} \rangle | \langle \text{inf-symbolid} \rangle | \langle \text{special} \rangle | \langle \text{reserved} \rangle$

Whitespace

$\langle \text{whitespace} \rangle ::= \langle \text{space} \rangle | \langle \text{newline} \rangle | \langle \text{carriage return} \rangle | \langle \text{tab} \rangle$
 $\quad | \langle \text{formfeed} \rangle | \langle \text{vtab} \rangle$

Syntactic Categories

$\langle \text{letter} \rangle ::= \mathbf{a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u}$
 $\quad | \mathbf{v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M}$
 $\quad | \mathbf{N|O|P|Q|R|S|T|U|V|W|X|Y|Z}$
 $\langle \text{digit} \rangle ::= \mathbf{0|1|2|3|4|5|6|7|8|9}$
 $\langle \text{sym} \rangle ::= \mathbf{[|]|!|#|%|&|$|*|+|-|/|<|>|=|?|@|^|~|'|`}$
 $\quad | \setminus \{ \langle \text{letter} \rangle \}^+$
 $\langle \text{spcl} \rangle ::= \mathbf{\{ | \} | (|) | . | ; | , | : | \}$
 $\langle \text{graph-spcl} \rangle ::= \mathbf{\forall | \nabla | \perp | \exists | \exists^+ | \lambda}$
 $\langle \text{graph-sym} \rangle ::= \mathbf{\delta | \perp | \sqsubseteq | \neg | \vee | \wedge | \rightarrow | \times | \Rightarrow | \Leftrightarrow | \neq}$
 $\langle \text{ext-graph-sym} \rangle ::= \langle \text{additional graphic symbols} \rangle$
 $\langle \text{symbol} \rangle ::= \langle \text{sym} \rangle | \langle \text{graph-sym} \rangle | \langle \text{ext-graph-sym} \rangle$
 $\langle \text{special} \rangle ::= \langle \text{spcl} \rangle | \langle \text{graph-spcl} \rangle$
 $\langle \text{alphanum} \rangle ::= \langle \text{letter} \rangle | \langle \text{digit} \rangle | \text{'|_}'$
 $\langle \text{any} \rangle ::= \langle \text{alphanum} \rangle | \langle \text{symbol} \rangle | \langle \text{special} \rangle | \langle \text{space} \rangle | \langle \text{tab} \rangle$

Comments

$\langle \text{line-comment} \rangle ::= \text{--} \{ \langle \text{any} \rangle \}^* \langle \text{line-end} \rangle$
 $\langle \text{line-end} \rangle ::= \langle \text{newline} \rangle | \langle \text{carriage return} \rangle | \langle \text{formfeed} \rangle | \langle \text{vtab} \rangle | \langle \text{eof} \rangle$
 $\langle \text{nest-comment} \rangle ::= (: \{ \langle \text{no-nest} \rangle | \langle \text{nest-comment} \rangle \}^* :)$
 $\langle \text{no-nest} \rangle ::= \{ \langle \text{any} \rangle \}^* \{ \langle \text{any} \rangle^* \{ (: ;) \} \langle \text{any} \rangle^* \}$
 $\langle \text{comment} \rangle ::= \langle \text{line-comment} \rangle | \langle \text{nest-comment} \rangle$

Character Constants

$\langle \text{char} \rangle ::= \langle \text{letter} \rangle | \langle \text{digit} \rangle | \langle \text{sym} \rangle | \langle \text{spcl} \rangle | _ | \langle \text{space} \rangle | \langle \text{escapes} \rangle$
 $\langle \text{escapes} \rangle ::= \backslash \mathbf{n} | \backslash \mathbf{t} | \backslash \mathbf{v} | \backslash \mathbf{r} | \backslash \mathbf{f} | \backslash \mathbf{a} | \backslash \backslash | \backslash ' | \backslash \backslash$
 $\langle \text{charconst} \rangle ::= \text{' } \langle \text{char} \rangle \text{'}$

String Constants

$\langle \text{string} \rangle ::= \text{" } \{ \langle \text{char} \rangle \}^* \text{"}$

Natural Numbers

$\langle \text{num} \rangle ::= \langle \text{digit} \rangle_{\{0\}} \{ \langle \text{digit} \rangle \}^*$

Identifiers

$\langle \text{alphanumeric} \rangle ::= \{ \langle \text{alphanum} \rangle \}^+ \{ \langle \text{char} \rangle | \langle \text{num} \rangle | \langle \text{reserved} \rangle \}$
 $\langle \text{symbolid} \rangle ::= \{ \langle \text{sym} \rangle \}^+ \{ \{ \langle \text{sym} \rangle \}^* \text{ -- } \{ \langle \text{sym} \rangle \}^* | \langle \text{reserved} \rangle \}$
 $\langle \text{inf-alphanumeric} \rangle ::= \cdot \langle \text{alphanumeric} \rangle \cdot$
 $\langle \text{inf-symbolid} \rangle ::= \cdot \langle \text{symbolid} \rangle \cdot$

Reserved Words

$\langle \text{reserved} \rangle ::=$ **enriches|export|in|hide|rename|SIG|to|let**
| **letrec|endlet|ALL|ALL+|EX|EX+|LAM**
| **if|then|else|endif|and|axioms|endaxioms**
| **data|strong|strict|total|freely|generated|by**
| **prio|sortsyn|class|subclass|of|sort|hidden**
| **param|body|via|!|[]**

Chapter 3

The Core Language

SPECTRUM consists of a language “in the small” and a language “in the large”. The first one is used for the design of a single specification unit, i.e. a basic component. The latter one is intended for structuring large system descriptions by combining and adapting specifications into more complex ones.

3.1 In the Small

3.1.1 Context Free Syntax

In this section we define the raw structure of our language “in the small” by giving a context-free grammar in EBNF style.

```
<specbody> ::= { <decls> }
<decls>    ::= {<signature> ;|<axioms> ;}*
```

Signatures

```
<signature> ::= class <id> [subclass of {<id> // ,}+ ]
| <id> subclass of {<id> // ,}+
| sort <sid> { <sid>* | :: <classexp> }
| {<sid> // ,}+ :: <classexp>
| sortsyn {<sid>}+ = <sortexp>
| SIG ( <specexp> )
| {<id> // ,}+ : [(<context>)] <sortexp> to <sortexp>
| {<inf-id> // ,}+ : [(<context>)] <sortexp> × <sortexp> to <sortexp> [(<prio>)]
| {<id> // ,}+ : [(<context>)] <sortexp>
| {<inf-id> // ,}+ : [(<context>)] <sortexp> × <sortexp> → <sortexp> [(<prio>)]
<classexp> ::= [( {<id> // ,}+ )] <id>
<prio>      ::= prio <num> [: {left|right}]
```

Sort Expressions

```
<sortexp> ::= <sortexp1>
| <sortexp1> → <sortexp> (Functional Sort)
```

$$\begin{aligned}
\langle \text{sortexp1} \rangle & ::= \langle \text{sortexp2} \rangle \\
& \quad | \langle \text{sortexp2} \rangle \{ \times \langle \text{sortexp2} \rangle \}^+ && \text{(Product Sort)} \\
\langle \text{sortexp2} \rangle & ::= \langle \text{asort} \rangle | \langle \text{sid} \rangle \{ \langle \text{asort} \rangle \}^+ \\
\langle \text{asort} \rangle & ::= \langle \text{sid} \rangle | (\langle \text{sortexp} \rangle)
\end{aligned}$$

Sort Contexts

$$\begin{aligned}
\langle \text{context} \rangle & ::= \{ \langle \text{scontext} \rangle // , \}^+ \Rightarrow \\
\langle \text{scontext} \rangle & ::= \{ \langle \text{sid} \rangle // , \}^+ :: \langle \text{id} \rangle
\end{aligned}$$

Axioms

$$\begin{aligned}
\langle \text{axioms} \rangle & ::= \mathbf{axioms} [\langle \text{varlist} \rangle] \{ \{ \langle \text{id} \rangle \} \langle \text{exp1} \rangle ; \}^* \mathbf{endaxioms} \\
& \quad | \langle \text{simplsorts} \rangle \mathbf{generated\ by} \langle \text{opns} \rangle \\
\langle \text{varlist} \rangle & ::= [\langle \text{context} \rangle] \{ \{ \forall | \forall^\perp \} \langle \text{opdecls} \rangle \}^+ \mathbf{in} \\
\langle \text{opdecls} \rangle & ::= \{ \langle \text{sopdecl} \rangle // , \}^+ | \{ \langle \text{id} \rangle // , \}^+ \\
\langle \text{sopdecl} \rangle & ::= \{ \langle \text{id} \rangle // , \}^+ : \langle \text{sortexp} \rangle \\
\langle \text{simplsorts} \rangle & ::= \{ \langle \text{sid} \rangle \{ \langle \text{sid} \rangle \}^* // , \}^+
\end{aligned}$$

Expressions

$$\begin{aligned}
\langle \text{exp1} \rangle & ::= \langle \text{exp2} \rangle \\
& \quad | \{ \forall | \forall^\perp | \exists | \exists^\perp \} \langle \text{opdecl} \rangle . \langle \text{exp1} \rangle \\
& \quad | \lambda \langle \text{pat} \rangle . \langle \text{exp1} \rangle \\
\langle \text{exp2} \rangle & ::= \langle \text{exp3} \rangle | \langle \text{exp2} \rangle \langle \text{id} \rangle [: \langle \text{asort} \rangle] \langle \text{exp1} \rangle \\
\langle \text{exp3} \rangle & ::= \langle \text{aexp} \rangle | \langle \text{exp3} \rangle \langle \text{aexp} \rangle \\
\langle \text{aexp} \rangle & ::= \langle \text{opn} \rangle | (\langle \text{exp1} \rangle) | (\langle \text{exp1} \rangle \{ , \langle \text{exp1} \rangle \}^+) | \langle \text{aexp} \rangle : \langle \text{asort} \rangle \\
\langle \text{pat} \rangle & ::= \langle \text{id} \rangle [: \langle \text{sortexp} \rangle] \\
& \quad | (\langle \text{id} \rangle [: \langle \text{sortexp} \rangle] \{ , \langle \text{id} \rangle [: \langle \text{sortexp} \rangle] \}^+)
\end{aligned}$$

Identifiers

$$\begin{aligned}
\langle \text{id} \rangle & ::= \langle \text{alphanumid} \rangle | \langle \text{symbolid} \rangle | \langle \text{num} \rangle | \langle \text{charconst} \rangle | \langle \text{string} \rangle \\
\langle \text{inf-id} \rangle & ::= \langle \text{inf-alphanumid} \rangle | \langle \text{inf-symbolid} \rangle \\
\langle \text{sid} \rangle & ::= \langle \text{id} \rangle \{ \rightarrow , \times , - , > , * \} \\
\langle \text{opn} \rangle & ::= \langle \text{id} \rangle | \langle \text{inf-id} \rangle \\
\langle \text{opns} \rangle & ::= \{ \langle \text{opn} \rangle // , \}^+
\end{aligned}$$

3.1.2 Semantic Objects

A basic specification consists of a signature and a set of axioms. Thus, its abstraction is defined as follows:

$$(\Sigma, Ax) \text{ or } sp \in \text{Specification} = \text{Sig} \times \text{Fin}(Ax)$$

We start with the description of the abstract signature Sig.

Signatures

A signature contains all global identifiers known to a specification. We use separate environments for the different kinds of identifiers.

$$(CE, SCE, SSE, ME, FE) \text{ or } \Sigma \in \text{Sig} = \text{ClassEnv} \times \text{SConEnv} \times \text{SSynEnv} \times \text{MapEnv} \times \text{FuncEnv}$$

The class environment contains all declared class identifiers and the defined subclass relation. We

use a finite set of tuples to describe this relation.

$$(IdSet, Subset) \text{ or } CE \in \text{ClassEnv} = \text{Fin}(\text{ClassId}) \times \text{Fin}(\text{ClassId} \times \text{ClassId})$$

In the sort constructor environment all declared sort constructor identifiers are registered. In addition, for each identifier the possibly overloaded type information is stored. It consists of the parameter classes and the result class. The number of parameter classes depends on the arity of the sort constructor.

$$(SConSet, SConType) \text{ or } SCE \in \begin{array}{l} \text{SConEnv} = \text{Fin}(\text{SConId}) \times \text{Fin}(\text{SconId} \times \text{ConType}) \\ \text{ConType} = \bigcup_{k \geq 0} (\text{ClassId}^k \times \text{ClassId}) \end{array}$$

The sort synonym environment contains all defined sort synonyms together with their assigned type expression abstractions.

$$\begin{array}{l} \text{SSynEnv} = \text{Id} \xrightarrow{fin} \text{TypeAbstr} \\ \text{TypeAbstr} = \bigcup_{k \geq 0} \text{TypeVar}^k \times \text{Type} \end{array}$$

Finally we need environments for the declared functions and mappings. They contain the sort information for the declared functions and mappings. In addition, for infix identifiers the priority (a natural number) and associativity information (left, right or none) is stored.

$$\begin{array}{l} FE \in \text{FuncEnv} = \text{Fin}(\text{TypeBind} \cup \text{TypeBind} \times \text{Prio} \times \text{Assoc}) \\ ME \in \text{MapEnv} = \text{Fin}(\text{TypeBind} \cup \text{TypeBind} \times \text{Prio} \times \text{Assoc}) \\ \text{TypeBind} = \text{Id} \times \text{TypeScheme} \\ p \in \text{Prio} = \text{subset of } \mathbb{N} \\ a \in \text{Assoc} = \text{left} \mid \text{right} \mid \text{none} \end{array}$$

The sort information for function and mapping identifiers consists of a binding for sort variables and an abstract sort expression. The sort variable binding assigns a class identifier to sort variables.

$$\begin{array}{l} \sigma \in \text{TypeScheme} = \text{SVarEnv} \times \text{Type} \\ SVE \in \text{SVarEnv} = \text{TypeVar} \xrightarrow{fin} \text{ClassId} \\ \tau \in \text{Type} = \text{abs}(\langle \text{sortexp} \rangle) \end{array}$$

Well Formed Signatures

Not all elements of the set Sig are legal signatures. Signatures must be *well formed*, i.e. they have to fulfil the additional conditions given below. To define these conditions we use the following notational shortcuts:

$$\begin{array}{l} MapSet = \{ id \mid \exists (id, \sigma) \in ME \vee \exists ((id, \sigma), p, a) \in ME \} \\ FuncSet = \{ id \mid \exists (id, \sigma) \in FE \vee \exists ((id, \sigma), p, a) \in FE \} \\ \text{Dom}(\Sigma) = IdSet \cup SconSet \cup MapSet \cup FuncSet \cup \text{Dom}(SSE) \\ FSV(\tau) = \text{the set of sort variables occurring freely in } \tau \end{array}$$

1. ClassEnv

- Each identifier occurring in *Subset* must be in *IdSet*. Formally:
 $\forall (x, y) \in Subset. (x \in IdSet) \wedge (y \in IdSet)$

2. SConEnv

- Each sort constructor identifier occurring in *SconType* must be in *SconSet*. Formally:
 $\forall (x, t) \in SconType. x \in SconSet.$
- The arity of each sort constructor occurring in *SconType* must be unique. Formally:

$\forall x \in SconSet. (x, (c_{11}, \dots, c_{1k}, c_1)) \in SconType \wedge (x, (c_{21}, \dots, c_{2l}, c_2)) \in SconType \Rightarrow k = l.$

- *SconType* must be coregular. Formally¹:
For each $id \in SconSet$ and $c \in IdSet$ the set $\{c^k \mid \exists (c, d) \in Subset^*. (id, (c^k, d)) \in SconType\}$ is either \emptyset or has a greatest element w.r.t. the extension of *Subset** on tuples.
- Each class identifier occurring in *SconType* must be in *IdSet*. Formally:
 $\forall (x, ((c_1, \dots, c_k), c)) \in SconType. c \in IdSet \wedge \{c_i \mid 1 \leq i \leq k\} \subseteq IdSet$

3. SSynEnv

- The identifiers $sid_i, 1 \leq i \leq n$, must be sort variables. Formally:
 $\forall (sid, ((sid_1, \dots, sid_n), \tau)) \in SSE.$
 $sid_i \notin SConSet \cup Dom(SSE), 1 \leq i \leq n.$
- Each sort variable occurring free in τ must be an identifier sid_i . Formally:
 $\forall (sid, ((sid_1, \dots, sid_n), \tau)) \in SSE.$
 $FSV(\tau) \subseteq \cup_{1 \leq i \leq n} \{sid_i\}$
- Sort synonyms must not be defined recursively. Formally:
 $\forall \tau \in Types. \exists \tau' \in Types. \tau \rightarrow_{\beta}^* \tau' \not\rightarrow_{\beta}^+ \tau''$
where $sid \tau_1 \dots \tau_n \rightarrow_{\beta} \{\alpha_1 \rightarrow \tau_1, \dots, \alpha_n \rightarrow \tau_n\} \tau$ iff
 $SSE(sid) = ((\alpha_1, \dots, \alpha_n), \tau)$
and \rightarrow_{β}^* is the reflexive, transitive extension of \rightarrow_{β} to sort expression $\tau \in Types$
and \rightarrow^+ is the transitive extension of \rightarrow_{β} to sort expression $\tau \in Types$
- The identifiers $sid_i, 1 \leq i \leq n$, must be pairwise disjoint. Formally:
 $\forall i, j. 1 \leq i \leq n \wedge 1 \leq j \leq n \wedge i \neq j \Rightarrow sid_i \neq sid_j.$

4. MapEnv

- *ME* must be a finite function w.r.t. the mapping identifiers. Formally²:
 $(i, m) \in ME \wedge (i, n) \in ME \Rightarrow m \stackrel{\alpha}{=} n$

5. FuncEnv

- *FE* must be a finite function w.r.t. the function identifiers. Formally:
 $(i, n) \in FE \wedge (i, n) \in FE \Rightarrow m \stackrel{\alpha}{=} n$

6. SVarEnv

- No identifier occurring in the signature is allowed as a bound identifier. Formally:
 $Dom(SVE) \cap Dom(\Sigma) = \emptyset$
- Each class identifier occurring in *SVE* is in *IdSet*. Formally:
 $\forall id. id \in Cod(SVE) \Rightarrow id \in IdSet$

7. Disjointness of Identifier Classes

- The sets of class identifiers, sort identifiers, mapping identifiers and function identifiers must be pairwise disjoint. Formally:
IdSet, SconSet, MapSet, FuncSet pairwise disjoint

¹*Subset** denotes the reflexive and transitive closure of the relation.

² $\stackrel{\alpha}{=}$ means equal up to renaming of sort variables.

Axioms

An axiom is either a formula block or a generation statement. A formula block consists of an environment and a set of formulae. The environment contains the local sort variables and variables. The generation axiom contains the constructed sorts together with their constructors.

$$\begin{array}{llll}
 \text{ax} & \in & \text{Ax} & = \text{FormAx} \cup \text{GenAx} \\
 (AE, \text{formulae}) & \in & \text{FormAx} & = \text{AxEnv} \times \text{Fin}(\text{Formula}) \\
 (\text{sois}, \text{cons}) & \in & \text{GenAx} & = \text{Fin}(\text{SconId}) \times \text{Fin}(\text{Id}) \\
 (\Delta, \Gamma) & \in & \text{AxEnv} & = \text{SVarEnv} \times \text{VarEnv} \\
 \Gamma & \in & \text{VarEnv} & = \text{Id} \xrightarrow{\text{fin}} \text{Type} \\
 \text{formula} & \in & \text{Formula} & = \text{abs}(\langle \text{exp1} \rangle)
 \end{array}$$

Well Formed Specifications

A specification is well formed if its signature is well formed and if the following condition holds:

- No identifier occurring in the signature is allowed as a bound identifier. Formally:
 $\text{Dom}(\Gamma) \cap \text{Dom}(\Sigma) = \emptyset$

3.1.3 Translation Rules

Now we give the translation rules for the language in the small. Context conditions that cannot be described conveniently with the rules are given separately. To increase readability we add the associated context free rules. In order to simplify the translation rules, the context free rules sometimes slightly differ from the productions given in Section 3.1.1. However, it is easy to prove that both variants always define the same language.

Specification Body

$$\begin{array}{ll}
 \langle \text{specbody} \rangle & ::= \{ \langle \text{decls} \rangle \} \\
 \langle \text{decls} \rangle & ::= \varepsilon \mid \langle \text{decl} \rangle \langle \text{decls} \rangle \\
 \langle \text{decl} \rangle & ::= \langle \text{signature} \rangle ; \mid \langle \text{axioms} \rangle ;
 \end{array}$$

$$\frac{\text{decls} \Rightarrow \Sigma}{\text{specbody} \Rightarrow \Sigma}$$

$$\frac{\text{decl} \Rightarrow \Sigma_1 \quad [\text{decls} \Rightarrow \Sigma_2]}{\text{decls} \Rightarrow \Sigma_1[\cup \Sigma_2]}$$

$$\overline{\text{axioms} \Rightarrow \emptyset}$$

The signatures of the individual declarations are unified. The axioms part does not contain any signature information.

Signature Declarations

Classes

$$\begin{array}{ll}
 \langle \text{signature} \rangle & ::= \text{class } \langle \text{id} \rangle [\text{subclass of } \{ \langle \text{id} \rangle // , \}^+] \\
 & \mid \langle \text{id} \rangle \text{ subclass of } \{ \langle \text{id} \rangle // , \}^+
 \end{array}$$

$$\overline{\text{class } id \Rightarrow ((\{id\}, \{(id, CPO)\}), \emptyset, \emptyset, \emptyset, \emptyset)}$$

$$\overline{\text{class } id \text{ subclass of } \{id_i //, \}^+ \Rightarrow ((\{id\}, \bigcup_i \{(id, id_i)\}), \emptyset, \emptyset, \emptyset, \emptyset)}$$

$$\overline{id \text{ subclass of } \{id_i //, \}^+ \Rightarrow ((\emptyset, \bigcup_i \{(id, id_i)\}), \emptyset, \emptyset, \emptyset, \emptyset)}$$

Each defined class identifier is stored in the class environment. In addition the subclass information is stored. By default, each class is a subclass of CPO.

Sort Constructors

$$\begin{aligned} \langle \text{signature} \rangle & ::= \text{sort } \langle \text{sid} \rangle \{ \langle \text{sid} \rangle^* \mid :: \langle \text{classexp} \rangle \} \\ & \quad \mid \{ \langle \text{sid} \rangle //, \}^+ :: \langle \text{classexp} \rangle \\ \langle \text{classexp} \rangle & ::= [(\{ \langle \text{id} \rangle //, \}^+)] \langle \text{id} \rangle \end{aligned}$$

$$\overline{\text{sort } sid \ sid_1 \dots sid_n \Rightarrow (\emptyset, (\{sid\}, \{(sid, \underbrace{(CPO, \dots, CPO)}_n), CPO\}), \emptyset, \emptyset, \emptyset)}$$

$$\overline{\text{sort } sid :: (id_1, \dots, id_n) id \Rightarrow (\emptyset, (\{sid\}, \{(sid, (CPO, \dots, CPO), CPO), (sid, (id_1, \dots, id_n), id)\}), \emptyset, \emptyset, \emptyset)}$$

$$\overline{\text{sort } sid :: id \Rightarrow (\emptyset, (\{sid\}, \{(sid, (), CPO), (sid, (), id)\}), \emptyset, \emptyset, \emptyset)}$$

$$\overline{\{sid_i //, \}^+ :: (id_1, \dots, id_n) id \Rightarrow (\emptyset, (\emptyset, \bigcup_i \{(sid_i, (id_1, \dots, id_n), id)\}), \emptyset, \emptyset, \emptyset)}$$

$$\overline{\{sid_i //, \}^+ :: id \Rightarrow (\emptyset, (\emptyset, \bigcup_i \{sid_i, (), id\}), \emptyset, \emptyset, \emptyset)}$$

The sort constructors are stored together with their class information. Each sort constructor can be applied to all sorts because each sort is an element of class CPO. Then the result class is always CPO.

Sort Synonyms

$$\langle \text{signature} \rangle ::= \text{sortsyn } \{ \langle \text{sid} \rangle \}^+ = \langle \text{sortexp} \rangle$$

$$\overline{\text{sortsyn } sid \ sid_1 \dots sid_n = \tau \Rightarrow (\emptyset, \emptyset, \{sid \rightarrow ((sid_1, \dots, sid_n), \tau)\}, \emptyset, \emptyset)}$$

Including a Signature

$$\langle \text{signature} \rangle ::= \mathbf{SIG} (\langle \text{specexp} \rangle)$$

$$\overline{\text{specexp} \Rightarrow \Sigma}$$

$$\mathbf{SIG} (\text{specexp}) \Rightarrow \Sigma$$

Mapping

$$\begin{aligned}
\langle \text{signature} \rangle & ::= \{ \langle \text{id} \rangle // , \}^+ : [\langle \text{context} \rangle] \langle \text{sortexp} \rangle \mathbf{to} \langle \text{sortexp} \rangle \\
& | \{ \langle \text{inf-id} \rangle // , \}^+ : [\langle \text{context} \rangle] \langle \text{sortexp} \rangle \times \langle \text{sortexp} \rangle \mathbf{to} \langle \text{sortexp} \rangle [\langle \text{prio} \rangle] \\
& \frac{[\text{context} \Rightarrow SVE]_{\emptyset}}{\{ \text{id}_i // , \}^+ : [\text{context}] \text{sortexp}_1 \mathbf{to} \text{sortexp}_2 \Rightarrow (\emptyset, \emptyset, \emptyset, \bigcup_i \{ \text{id}_i, SVE, \text{sortexp}_1 \rightarrow \text{sortexp}_2 \}, \emptyset)} \\
& \frac{[\text{context} \Rightarrow SVE]_{\emptyset} \quad [\text{prio} \Rightarrow (n, a)]_{(0, \text{none})}}{\{ \text{id}_i // , \}^+ : [\text{context}] \text{sortexp}_1 \times \text{sortexp}_2 \mathbf{to} \text{sortexp}_3 [\text{prio}] \Rightarrow \\
& (\emptyset, \emptyset, \emptyset, \bigcup_i \{ \text{id}_i, SVE, \text{sortexp}_1 \times \text{sortexp}_2 \rightarrow \text{sortexp}_3, n, a \}, \emptyset)}
\end{aligned}$$

In the mapping environment the syntactic **to** is replaced by the function space constructor. From a syntactic point of view there is no difference between these two identifiers. If the priority information is missing for infix identifiers we assume priority 0 and no associativity.

Functions

$$\begin{aligned}
\langle \text{signature} \rangle & ::= \{ \langle \text{id} \rangle // , \}^+ : [\langle \text{context} \rangle] \langle \text{sortexp} \rangle \\
& | \{ \langle \text{inf-id} \rangle // , \}^+ : [\langle \text{context} \rangle] \langle \text{sortexp} \rangle \times \langle \text{sortexp} \rangle \rightarrow \langle \text{sortexp} \rangle [\langle \text{prio} \rangle] \\
& \frac{[\text{context} \Rightarrow SVE]_{\emptyset}}{\{ \text{id}_i // , \}^+ : [\text{context}] \text{sortexp} \Rightarrow (\emptyset, \emptyset, \emptyset, \emptyset, \bigcup_i \{ \text{id}_i, SVE, \text{sortexp} \})} \\
& \frac{[\text{context} \Rightarrow SVE]_{\emptyset} \quad [\text{prio}] \Rightarrow (n, a)_{(0, \text{none})}}{\{ \text{id}_i // , \}^+ : [\text{context}] \text{sortexp}_1 \times \text{sortexp}_2 \rightarrow \text{sortexp}_3 [\text{prio}] \Rightarrow \\
& (\emptyset, \emptyset, \emptyset, \emptyset, \bigcup_i \{ \text{id}_i, SVE, \text{sortexp}_1 \times \text{sortexp}_2 \rightarrow \text{sortexp}_3, n, a \})}
\end{aligned}$$

Priority

$$\begin{aligned}
\langle \text{prio} \rangle & ::= \mathbf{prio} \langle \text{num} \rangle [: \{ \text{left} | \text{right} \}] \\
& \frac{}{\mathbf{prio} \text{ num} \Rightarrow (\text{val}(\text{num}), \text{none})} \\
& \frac{}{\mathbf{prio} \text{ num} : \text{left} \Rightarrow (\text{val}(\text{num}), \text{left})} \\
& \frac{}{\mathbf{prio} \text{ num} : \text{right} \Rightarrow (\text{val}(\text{num}), \text{right})}
\end{aligned}$$

Sort Contexts

$$\begin{aligned}
& \textit{Sort Contexts} \\
\langle \text{context} \rangle & ::= \{ \langle \text{scontext} \rangle // , \}^+ \Rightarrow \\
\langle \text{scontext} \rangle & ::= \{ \langle \text{sid} \rangle // , \}^+ :: \langle \text{id} \rangle \\
& \frac{}{\{ \text{sid}_i // , \}^+ :: \text{id} \Rightarrow \bigcup_i \{ \text{sid}_i \mapsto \text{id} \}}
\end{aligned}$$

$$\frac{scontext \Rightarrow SVE_1 \quad [context \Rightarrow SVE_2]}{scontext[context] \Rightarrow SVE_1[\cup SVE_2]} \{[dom(SVE_1) \cap dom(SVE_2) = \emptyset]\}$$

Note that the domains of the unified sort variable environments must be disjoint.

Generated By

$$\begin{aligned} \langle \text{axioms} \rangle & ::= \langle \text{simplesorts} \rangle \text{ generated by } \langle \text{opns} \rangle \\ \langle \text{simplesorts} \rangle & ::= \{ \langle \text{sid} \rangle \{ \langle \text{sid} \rangle^* // , \}^+ \\ \langle \text{opns} \rangle & ::= \{ \langle \text{opn} \rangle // , \}^+ \\ \langle \text{opn} \rangle & ::= \langle \text{id} \rangle \mid \langle \text{inf-id} \rangle \end{aligned}$$

$$\frac{\Sigma \vdash \text{simplesorts} \Rightarrow \text{sois} \quad \Sigma \vdash \text{opns} \Rightarrow \text{cons}}{\Sigma \vdash \text{simplesorts generated by opns} \Rightarrow (\text{sois}, \text{cons})} \{ \text{see context conditions given below} \}$$

$$\frac{}{\Sigma \vdash \{ \text{sid}_i \{ \text{sid}_{ij} \}^* // , \}^+ \Rightarrow \bigcup_i \{ \text{sid}_i \}}$$

$$\frac{}{\Sigma \vdash \{ [\cdot] \text{id}_i [\cdot] // , \}^+ \Rightarrow \bigcup_i \{ \text{id}_i \}}$$

© The identifier sid_i must be defined as a sort constructor. Formally:

$$\text{sid}_i \in SConSet$$

© The identifier sid_{ij} must be a sort variable. Formally:

$$\text{sid}_{ij} \notin \text{dom}(\Sigma)$$

© The identifier id_i must be defined as a function if the optional dots are missing. Formally:

$$(\text{id}_i, \sigma) \in FE \text{ for some } \sigma$$

© The identifier id_i must be defined as an infix function if id_i is included in \cdot 's. Formally:

$$(\text{id}_i, \sigma, n, a) \in FE \text{ for some } \sigma, n, a$$

© The result sort of the constructors in cons must be a sort of interest from sois . Formally:

$$\begin{aligned} \forall \text{id} \in \text{cons}. \exists \text{sid} \in \text{sois} \\ (\text{id} : (SVE, \tau \rightarrow \text{sid sid}_1 \dots \text{sid}_n) \in \Sigma \vee \\ \text{id} : (SVE, \text{sid sid}_1 \dots \text{sid}_n) \in \Sigma) \\ \wedge \text{sid}_i \notin SConSet \ 1 \leq i \leq n \end{aligned}$$

© All sort variables occurring in the argument sort of a constructor from cons must occur in the result sort of the constructor. Formally:

$$\begin{aligned} \forall \text{id} \in \text{cons}. \\ (\text{id} : (SVE, \tau_1 \rightarrow \tau_2)) \in \Sigma \\ \Rightarrow \text{FSV}(\tau_1) \subseteq \text{FSV}(\tau_2) \end{aligned}$$

© There must be a constructor in cons for each sort of interest from sois . Formally:

$$\begin{aligned} \forall \text{sid} \in \text{sois}. \exists \text{id} \in \text{cons} \\ (\text{id} : (SVE, \tau \rightarrow \text{sid sid}_1 \dots \text{sid}_n) \in \Sigma \vee \\ \text{id} : (SVE, \text{sid sid}_1 \dots \text{sid}_n) \in \Sigma) \\ \wedge \text{sid}_i \notin SConSet \ 1 \leq i \leq n \end{aligned}$$

- © Each sort of interest from *sois* is only allowed to occur at the uppermost level of the argument sort of a constructor. Formally:

$$\begin{aligned}
& \forall id \in cons . \forall sid \in sois \\
& (id : (SVE, \tau_1 \times \dots \times \tau_n \rightarrow \tau)) \in \Sigma, n \geq 1 \\
& \Rightarrow ((\exists \alpha_1, \dots, \alpha_m. \tau_1 = sid \alpha_1 \dots \alpha_m \wedge \alpha_j \notin SConSet \ 1 \leq j \leq m) \\
& \vee \neg \text{occurs}(sid, \tau_i)) \ 1 \leq i \leq n
\end{aligned}$$

3.1.4 The Sort Inference

SPECTRUM has a static sort system. However, a context free description as given in Section 3.1.1 cannot completely define a statically sorted language. Sorting constraints are typically context sensitive. Therefore we will define the well-sorted axioms using a logical calculus. To simplify the description of well-sortedness for axioms blocks we translate axioms blocks to equivalent ones in the following way:

axioms $[context] \ \{\forall|\forall^\perp\} \ opdecls_1 \dots \{\forall|\forall^\perp\} \ opdecls_n \ \mathbf{in}$
 $\{\{id_1\}\} \ exp_1;$
 \vdots
 $\{\{id_m\}\} \ exp_m;$
endaxioms

is translated to

axioms $[context] \ \mathbf{in} \ \{\forall|\forall^\perp\} \ opdecls_1 \dots \{\forall|\forall^\perp\} \ opdecls_n.$
 $exp_1 \wedge \dots \wedge exp_m;$
endaxioms

- © Each axioms block must be well-sorted. Formally:

- **axiom in exp endaxioms:** for some *SVE*. $SVE, \emptyset \triangleright_\Sigma \ exp :: \text{Bool}$
- **axiom context in exp endaxioms:** $SVE, \emptyset \triangleright_\Sigma \ exp :: \text{Bool}$
where $context \Rightarrow SVE$

Note that we use σ both for sort variable substitutions and for their extensions to sort expressions. $FI(exp)$ denotes the set of identifiers occurring freely in expression *exp*.

$$\begin{aligned}
& \frac{}{\Delta, \Gamma \triangleright_\Sigma \ id :: \Gamma(id)} \ (var) \\
& \frac{\forall \alpha \in dom(SVE). \ \Delta \vdash_\Sigma \ \sigma(\alpha) : SVE(\alpha)}{\Delta, \Gamma \triangleright_\Sigma \ id :: \sigma(\tau)} \ (const) \ \left\{ \begin{array}{l} FE(id) = (SVE, \tau) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \end{array} \right. \\
& \frac{\forall \alpha \in dom(SVE). \ \Delta \vdash_\Sigma \ \sigma(\alpha) : SVE(\alpha)}{\Delta, \Gamma \triangleright_\Sigma \ .id. :: \sigma(\tau)} \ (iconst) \ \left\{ \begin{array}{l} FE(id) = ((SVE, \tau), n, a) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \end{array} \right. \\
& \frac{\Delta, \Gamma \triangleright_\Sigma \ exp :: \tau}{\Delta, \Gamma \triangleright_\Sigma \ (exp) :: \tau} \ (paranthesis) \\
& \frac{\Delta, \Gamma \triangleright_\Sigma \ exp_i :: \tau \ 2 \leq i \leq n}{\Delta, \Gamma \triangleright_\Sigma \ (exp_1, \dots, exp_n) :: \tau_1 \times \dots \times \tau_n} \ (tuple) \\
& \frac{\Delta, \Gamma \triangleright_\Sigma \ exp :: \tau}{\Delta, \Gamma \triangleright_\Sigma \ exp : \tau :: \tau} \ (constrained) \\
& \frac{\Delta, \Gamma \triangleright_\Sigma \ exp_1 :: \tau_1 \rightarrow \tau_2 \quad \Delta, \Gamma \triangleright_\Sigma \ exp_2 :: \tau_1}{\Delta, \Gamma \triangleright_\Sigma \ exp_1 \ exp_2 :: \tau_2} \ (appl)
\end{aligned}$$

$$\begin{array}{c}
\frac{\Delta, \Gamma \triangleright_{\Sigma} \text{exp}_1 :: \tau_1 \quad \Delta, \Gamma \triangleright_{\Sigma} \text{exp}_2 :: \tau_2}{\Delta, \Gamma \triangleright \text{exp}_1 \text{ id}[: \sigma(\tau)] \text{exp}_2 :: \tau_3} \text{ (infix)} \left\{ \begin{array}{l} FE(id) = ((SVE, \tau), n, a) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \\ \forall \alpha \in \text{dom}(SVE). \\ \Delta \vdash_{\Sigma} \sigma(\alpha) : SVE(\alpha) \\ \tau = \tau_1 \times \tau_2 \rightarrow \tau_3 \end{array} \right. \\
\\
\frac{\Delta, \Gamma \text{id} \rightarrow \tau_1 \triangleright_{\Sigma} \text{exp} :: \tau_2}{\Delta, \Gamma \triangleright \lambda \text{id}[: \tau_1]. \text{exp} :: \tau_1 \rightarrow \tau_2} \text{ (abstr1)} \{ FI(\text{exp}) \cap \text{Dom}(ME) = \emptyset \\
\\
\frac{\Delta, \Gamma \text{id}_1 \rightarrow \tau_1. \dots \text{id}_n \rightarrow \tau_n \triangleright_{\Sigma} \text{exp} :: \tau}{\Delta, \Gamma \triangleright_{\Sigma} \lambda (\text{id}_1[: \tau_1], \dots, \text{id}_n[: \tau_n]). \text{exp} :: \tau_1 \times \dots \times \tau_n \rightarrow \tau} \text{ (abstr2)} \left\{ \begin{array}{l} \text{id}_i \text{ pairwise disjoint} \\ n \geq 2 \\ FI(\text{exp}) \cap \text{Dom}(ME) = \emptyset \end{array} \right. \\
\\
\frac{\Delta, \Gamma \text{id}_1 \rightarrow \tau_1, \dots, \text{id}_n \rightarrow \tau_n \triangleright_{\Sigma} \text{exp} :: Bool}{\Delta, \Gamma \triangleright_{\Sigma} Q \text{id}_1. \dots \text{id}_n. \text{exp} :: Bool} \text{ (quant}_1) \left\{ \begin{array}{l} Q \in \{\forall, \forall^{\perp}, \exists, \exists^{\perp}\} \\ \text{id}_i \text{ pairwise disjoint} \end{array} \right. \\
\\
\frac{\Delta, \Gamma \text{id}_1 \rightarrow \tau_1. \dots \text{id}_{n_1} \rightarrow \tau_1. \dots \text{id}_m \rightarrow \tau_m. \dots \text{id}_{n_m} \rightarrow \tau_m \triangleright_{\Sigma} \text{exp} :: Bool}{\Delta, \Gamma \triangleright_{\Sigma} Q \text{id}_1, \dots, \text{id}_{n_1} : \tau_1, \dots, \text{id}_m, \dots, \text{id}_{n_m} : \tau_m. \text{exp} :: Bool} \text{ (quant}_2) \\
\left\{ \begin{array}{l} Q \in \{\forall, \forall^{\perp}, \exists, \exists^{\perp}\} \\ \text{id}_{ij} \text{ pairwise disjoint} \end{array} \right. \\
\\
\frac{\Delta, \Gamma \triangleright_{\Sigma} \text{exp} :: \sigma(\tau_1)}{\Delta, \Gamma \triangleright_{\Sigma} \text{id exp} :: \sigma(\tau_2)} \text{ (mappl)} \left\{ \begin{array}{l} ME(id) = (SVE, \tau) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \\ \forall \alpha \in \text{Dom}(SVE). \\ \Delta \vdash_{\Sigma} \sigma(\tau) : SVE(\alpha) \\ \tau = \tau_1 \rightarrow \tau_2 \end{array} \right. \\
\\
\frac{\Delta, \Gamma \triangleright_{\Sigma} \text{exp} :: \sigma(\tau_1)}{\Delta, \Gamma \triangleright_{\Sigma} \text{id. exp} :: \sigma(\tau_2)} \text{ (imappl)} \left\{ \begin{array}{l} ME(id) = (SVE, \tau) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \\ \forall \alpha \in \text{Dom}(SVE). \\ \Delta \vdash_{\Sigma} \sigma(\tau) : SVE(\alpha) \\ \tau = \tau_1 \rightarrow \tau_2 \end{array} \right. \\
\\
\frac{\Delta, \Gamma \triangleright_{\Sigma} \text{exp}_1 :: \sigma(\tau_1) \quad \Delta, \Gamma \triangleright_{\Sigma} \text{exp}_2 :: \sigma(\tau_2)}{\Delta, \Gamma \triangleright_{\Sigma} \text{exp}_1 \text{ id}[: \sigma(\tau)] \text{exp}_2 :: \sigma(\tau_3)} \text{ (minfix)} \left\{ \begin{array}{l} ME(id) = ((SVE, \tau), n, a) \\ \sigma : FSV(\tau) \xrightarrow{fin} Type \\ \forall \alpha \in \text{dom}(SVE). \\ \Delta \vdash_{\Sigma} \sigma(\alpha) : SVE(\alpha) \\ \tau = \tau_1 \times \tau_2 \rightarrow \tau_3 \end{array} \right. \\
\\
\frac{\Delta, \Gamma \triangleright_{\Sigma} \text{exp} :: \tau_1}{\Delta, \Gamma \triangleright_{\Sigma} \text{exp} :: \tau_2} \text{ (sortsyn)} \{ \tau_1 \rightarrow_{\beta}^* \tau_2
\end{array}$$

The side condition in rules *(abstr1)* and *(abstr2)* ensures that no mapping is used in the body of a λ -abstraction. Otherwise a λ -abstraction may denote a non-continuous function. This is, however, not allowed in SPECTRUM.

3.1.5 Class Inference

In SPECTRUM sorts are classified by classes. Therefore we need a second calculus to define the judgement $\tau : C$, stating that sort τ belongs to class C .

$$\frac{}{\Delta \vdash_{\Sigma} \alpha : C} \text{ (svar)} \{ C \in \Delta(\alpha)$$

$$\frac{\Delta \vdash_{\Sigma} \tau_i : C_i \quad 1 \leq i \leq n}{\Delta \vdash_{\Sigma} \text{sid } \tau_1 \dots \tau_n : C} (\text{sconappl}) \left\{ \begin{array}{l} n \geq 0 \\ (\text{sid}, ((C_1, \dots, C_n), C) \in \text{SconType}) \end{array} \right.$$

$$\frac{\Delta \vdash_{\Sigma} \tau : C_1}{\Delta \vdash_{\Sigma} \tau : C_2} (\text{subclass}) \left\{ (C_1, C_2) \in \text{Subset} \right.$$

3.1.6 Infix Symbols

The context free syntax given in Section 3.1.1 is ambiguous with respect to expressions because of the user definable infix identifiers. Thus, parsing an expression possibly yields a set of semantic objects. The following context condition uniquely selects the object with the desired binding.

- © The binding of infix identifiers must be correct with respect to the given priority and associativity. Formally:

$$\forall \text{exp} \in \text{Formula} . \exists n \in \text{Prio} \cup \{\infty\}, a \in \text{Assoc} \\ \text{exp} \Rightarrow (n, a)$$

where $\text{exp} \Rightarrow (n, a)$ is inductively defined on the structure of exp in the following way:

$$\frac{\text{exp}_1 \Rightarrow (n_1, a_1) \quad \text{exp}_2 \Rightarrow (n_2, a_2)}{\text{exp}_1 \text{ id } \text{exp}_2 \Rightarrow (n, a)}$$

iff

$$\begin{aligned} & (FE(id) = (\sigma, n, a) \vee ME(id) = (\sigma, n, a)) \wedge \\ & (n_1 > n \wedge n_2 > n \vee \\ & n_1 = n \wedge n_2 > n \wedge a_1 = a = \text{left} \vee \\ & n_1 > n \wedge n_2 = n \wedge a_2 = a = \text{right} \vee \\ & n_1 = n = n_2 \wedge a_1 = \text{left} \wedge a = \text{none} \wedge a_2 = \text{right}) \end{aligned}$$

$$\frac{\text{exp}_1 \Rightarrow (n_1, a_1) \dots \text{exp}_m \Rightarrow (n_m, a_m)}{\text{exp} \Rightarrow (\infty, \text{none})} \left\{ \begin{array}{l} \text{for all other inductive cases,} \\ \text{where } \text{exp}_1 \dots \text{exp}_m, m \geq 0 \\ \text{are all direct} \\ \text{subexpressions of } \text{exp} \end{array} \right.$$

3.2 In the Large

3.2.1 Context Free Syntax

$$\begin{aligned} \langle \text{system} \rangle & ::= \langle \text{syspart} \rangle [\text{system}] \\ \langle \text{syspart} \rangle & ::= \langle \text{alphanumid} \rangle = \{ \langle \text{specexp} \rangle \mid \langle \text{sigmorph} \rangle \mid \langle \text{specabstr} \rangle \} \end{aligned}$$

Signature Morphisms

$$\begin{aligned} \langle \text{sigmorph} \rangle & ::= [\langle \text{rename-list} \rangle] \\ \langle \text{rename-list} \rangle & ::= \langle \text{rename} \rangle [, \langle \text{rename-list} \rangle] \\ \langle \text{rename} \rangle & ::= \langle \text{inf-id} \rangle \text{ to } \langle \text{inf-id} \rangle \mid \langle \text{id} \rangle \text{ to } \langle \text{id} \rangle \mid \langle \text{sid} \rangle \text{ to } \langle \text{sid} \rangle \\ \langle \text{morph} \rangle & ::= \langle \text{alphanumid} \rangle \mid \langle \text{sigmorph} \rangle \end{aligned}$$

Structured Specifications

$$\begin{aligned}
\langle \text{specexp} \rangle & ::= \langle \text{aspecexp} \rangle [+ \langle \text{specexp} \rangle] \\
& \quad | \quad \mathbf{abstr} \ (\langle \text{arg-lst} \rangle) \\
\langle \text{arg-lst} \rangle & ::= \langle \text{specexp} \rangle [\mathbf{via} \langle \text{morph} \rangle] [, \langle \text{arg-lst} \rangle] \\
\langle \text{aspecexp} \rangle & ::= \mathbf{rename} \ \langle \text{specexp} \rangle \ \mathbf{by} \ \langle \text{morph} \rangle \\
& \quad | \quad (\langle \text{specexp} \rangle) \\
& \quad | \quad \mathbf{hide} \ \langle \text{sigel-set} \rangle \ \mathbf{in} \ \langle \text{aspecexp} \rangle \\
& \quad | \quad \mathbf{export} \ \langle \text{sigel-set} \rangle \ \mathbf{in} \ \langle \text{aspecexp} \rangle \\
& \quad | \quad \{ \mathbf{enriches} \ \langle \text{specexp} \rangle; \langle \text{decls} \rangle \} \\
& \quad | \quad \langle \text{aspecexp} \rangle | \langle \text{specbody} \rangle \\
\langle \text{sigel-set} \rangle & ::= \langle \text{sigel} \rangle [, \langle \text{sigel-set} \rangle] \\
\langle \text{sigel} \rangle & ::= \langle \text{opn} \rangle | \langle \text{sid} \rangle | \mathbf{SIG} \ (\langle \text{specexp} \rangle)
\end{aligned}$$

Parameterized Specifications

$$\begin{aligned}
\langle \text{abstr} \rangle & ::= \langle \text{alphanumid} \rangle | \langle \text{specabstr} \rangle \\
\langle \text{specabstr} \rangle & ::= \mathbf{param} \ \langle \text{spec-list} \rangle \mathbf{body} \ \langle \text{aspecexp} \rangle \\
\langle \text{spec-list} \rangle & ::= \langle \text{alphanumid} \rangle = \langle \text{specexp} \rangle [, \langle \text{spec-list} \rangle]
\end{aligned}$$

3.2.2 Compound Semantic Objects

A specification text consists of a set of (parameterized) specifications and signature morphisms (the system parts) together with their declared names. Its abstraction is therefore a *specification environment*

$$c \in \text{SpecEnv} = \text{Alphanumid} \rightarrow \text{SysPart}$$

i.e. a finite mapping from identifiers to system parts.

The abstraction ρ of a signature morphism is again a finite mapping, from identifiers to identifiers. Abstract signature morphisms are used not only to rename specifications but also to implement hiding. A symbol to be hidden is renamed to a fresh name, not available at the user level. As a consequence, this symbol acts as if it was existentially bound: it cannot be used in any enclosing specification and does not collide with any other hidden symbol. Fresh names are taken from the set of identifiers *HiddenId* which is disjoint from the set of concrete syntax identifiers *VisibleId*. The set of all identifiers is *AllId*.

To generate fresh hidden symbols we keep track in the base

$$(HIS, C) \in \text{Base} = \text{HiddenIdSet} \times \text{SpecEnv}$$

of the set *HIS* of hidden symbols generated so far in the specification text. A hidden identifier *hid* is then fresh if $hid \notin HIS$. For each specification *SP*, the function $hids(SP)$ returns the set of hidden symbols in *SP*.

The abstraction of a specification was defined in the previous section. It is a pair consisting of a signature and a set of axiom blocks (a flat specification). The abstraction of a parameterized specification

$$SA \in \text{PSpecification} = \text{SpecParam} \times \text{Specexp}$$

is a tuple consisting of the list

$$SpP \in \text{SpecParam} = \text{List}(\text{Alphanumid} \times \text{Specification})$$

of formal parameter specifications and the specification expression of the body. We use a list instead of finite mapping for the formal parameters because the order of the actual parameter list

$$SpA \in \text{SpecArgs} = \text{List}(\text{Sigmorph} \times \text{Specification})$$

must match the order of the formal parameters. Actual parameters are required to match syntac-

tically the formal parameters modulo renaming i.e.

$$\text{if } (\rho_i, SpA_i) \in SpA \quad \text{then} \quad \rho_i(\Sigma(SpP_i)) \subseteq \Sigma(SpA_i).$$

The definition of compound semantic objects for structured specifications is summarized below:

$$\begin{aligned} (HIS, C) \text{ or } B &\in Base &= HiddenIdSet \times SpecEnv \\ C &\in SpecEnv &= Alphanumeric \xrightarrow{fin} SysPart \\ &SysPart &= SigMorph \cup Specification \cup PSpecification \\ \rho &\in SigMorph &= AllId \xrightarrow{fin} AllId \\ SA &\in PSpecification &= SpecParam \times Specexp \\ SpP &\in SpecParam &= List(Alphanumeric \times Specification) \\ SpA &\in SpecArgs &= List(Sigmorph \times Specification) \\ \\ VId &\in VisibleId &= Sid \cup Op \\ HId &\in HiddenId &= \text{a set disjoint from } VisibleId \\ AId &\in AllId &= VisibleId \cup HiddenId \\ \\ VIS &\in VisibleIdSet &= Fin(VisibleId) \\ HIS &\in HiddenIdSet &= Fin(HiddenId) \end{aligned}$$

We are now ready to give the translation rules. To increase their readability, each rule is given together with its associated context free production. Context checks which cannot be described conveniently with the rules are given separately.

3.2.3 Translation Rules

System Specification

$$\langle \text{system} \rangle ::= \langle \text{syspart} \rangle [\text{system}]$$

$$\langle \text{syspart} \rangle ::= \langle \text{alphanumeric} \rangle = \{ \langle \text{specexp} \rangle \mid \langle \text{sigmorph} \rangle \mid \langle \text{specabstr} \rangle \}$$

$$\frac{B \cup SY \vdash \text{syspart} \Rightarrow P \quad [B \cup P \vdash \text{system} \Rightarrow SY]}{B \vdash \text{syspart} [\text{system}] \Rightarrow P [\cup SY]} \left\{ \begin{array}{l} \text{no cyclic reference occurs} \\ \text{in } \text{syspart} [\text{system}] \end{array} \right.$$

The system part *syspart* can contain forward references to system part names from *system*. Therefore it is necessary to add the abstract representation *SY* of *system* to the current environment when translating *system*. However, references are not allowed to be cyclic.

$$\frac{B \vdash \text{specexp} \Rightarrow SP}{B \vdash \text{alphanumeric} = \text{specexp} \Rightarrow (hids(SP), \{\text{alphanumeric} \rightarrow SP\})} \{ \text{alphanumeric} \notin \text{dom}(B) \}$$

This rule, together with the previous one, assures the propagation of the hidden symbols into the base.

$$\frac{B \vdash \text{sigmorph} \Rightarrow \rho}{B \vdash \text{alphanumeric} = \text{sigmorph} \Rightarrow (\emptyset, \{\text{alphanumeric} \rightarrow \rho\})} \{ \text{alphanumeric} \notin \text{dom}(B) \}$$

$$\frac{B \vdash \text{specabstr} \Rightarrow SA}{B \vdash \text{alphanumeric} = \text{specabstr} \Rightarrow (\emptyset, \{\text{alphanumeric} \rightarrow SA\})} \{ \text{alphanumeric} \notin \text{dom}(B) \}$$

Signature Morphisms

$\langle \text{sigmorph} \rangle ::= [\langle \text{rename-list} \rangle]$

$\langle \text{rename-list} \rangle ::= \langle \text{rename} \rangle [, \langle \text{rename-list} \rangle]$

$\langle \text{rename} \rangle ::= \langle \text{inf-id} \rangle \mathbf{to} \langle \text{inf-id} \rangle \mid \langle \text{id} \rangle \mathbf{to} \langle \text{id} \rangle \mid \langle \text{sid} \rangle \mathbf{to} \langle \text{sid} \rangle$

$\langle \text{morph} \rangle ::= \langle \text{alphanumericid} \rangle \mid \langle \text{sigmorph} \rangle$

$$\frac{B \vdash \text{rename-list} \Rightarrow \rho}{B \vdash [\text{rename-list}] \Rightarrow \rho}$$

$$\frac{B \vdash \text{rename} \Rightarrow \rho_1 \quad [B \vdash \text{rename-list} \Rightarrow \rho_2]}{B \vdash \text{rename} [, \text{rename-list}] \Rightarrow \rho_1 [\cup \rho_2]} \{[\text{dom}(\rho_1) \cap \text{dom}(\rho_2) = \emptyset]\}$$

Abstractly, a signature morphism is a finite map. This motivates the side condition.

$$\overline{B \vdash \text{inf-id}_1 \mathbf{to} \text{inf-id}_2 \Rightarrow \{\text{inf-id}_1 \rightarrow \text{inf-id}_2\}}$$

$$\overline{B \vdash \text{id}_1 \mathbf{to} \text{id}_2 \Rightarrow \{\text{id}_1 \rightarrow \text{id}_2\}}$$

$$\overline{B \vdash \text{sid}_1 \mathbf{to} \text{sid}_2 \Rightarrow \{\text{sid}_1 \rightarrow \text{sid}_2\}}$$

$$\frac{\text{alphanumericid} \in \text{dom}(B)}{B \vdash \text{alphanumericid} \Rightarrow B(\text{alphanumericid})}$$

The name of a system part is translated to its definition.

Plus

$\langle \text{specexp} \rangle ::= \langle \text{aspecexp} \rangle [+ \langle \text{specexp} \rangle]$

$\langle \text{aspecexp} \rangle ::= \langle \text{alphanumericid} \rangle \mid \langle \text{specbody} \rangle$

$$\frac{B \vdash \text{aspecexp} \Rightarrow SP_1 \quad [B \vdash \text{specexp} \Rightarrow SP_2]}{B \vdash \text{aspecexp} [+ \text{specexp}] \Rightarrow SP_1 [\cup SP_2]} \{SP_1[\cup SP_2] \text{ is well formed}\}$$

The sum of two flat specifications is again a flat specification with signature and axioms the union of the signatures and axioms of the component specifications. Clearly, this union has to be well formed; otherwise it is rejected.

For *alphanumericid* one uses the rule given in the previous section.

Rename

$\langle \text{specexp} \rangle ::= \mathbf{rename} \langle \text{specexp} \rangle \mathbf{by} \langle \text{morph} \rangle$

Abstractly, a signature morphism ρ is a finite map from identifiers to identifiers. If $id \notin \text{dom}(\rho)$ we consider that $\rho(id) = id$. A finite map ρ is extended to expressions, sort expressions and environments in a trivial way which we do not further describe here. When a signature morphism ρ is applied to a specification SP , it is possible that $\text{dom}(\rho)$ contains identifiers outside the domain $\text{dom} \Sigma(SP)$ of signature identifiers. To avoid both the collision of these identifiers with variables bound in SP and the renaming of identifiers occurring in the standard signature³ Σ_S we restrict ρ to $\Sigma \setminus \Sigma_S$ before applying it to SP^4 . We ambiguously write this as $\rho|_{\Sigma}$.

$$\frac{C \vdash \text{specexp} \Rightarrow (\Sigma, Ax) \quad C \vdash \text{morph} \Rightarrow \rho}{C \vdash \mathbf{rename} \text{ specexp} \mathbf{by} \text{ morph} \Rightarrow \rho|_{\Sigma}(\Sigma, Ax)} \quad \{\rho|_{\Sigma}(\Sigma, Ax) \text{ is well formed}\}$$

The renamed specification has to be well formed. This is a very strong condition which excludes all erroneous renamings which occur when changing the attributes of an identifier (e.g. a sort identifier is renamed to an identifier which was declared as a function in the same specification or a function identifier is renamed to a function which occurs in the same specification with another signature).

Hide

$\langle \text{aspecexp} \rangle ::= \mathbf{hide} \langle \text{sigel-set} \rangle \mathbf{in} \langle \text{aspecexp} \rangle$

$\langle \text{sigel-set} \rangle ::= \langle \text{sigel} \rangle [, \langle \text{sigel-set} \rangle]$

$\langle \text{sigel} \rangle ::= \langle \text{opn} \rangle | \langle \text{sid} \rangle | \mathbf{SIG} (\langle \text{specexp} \rangle)$

As we already mentioned an identifier is hidden by renaming it to a fresh identifier from *HiddenId*. This fresh renaming is abstractly realized with an injective function

$$(\cdot)_H : \text{VisibleId} \rightarrow \text{HiddenId}$$

where $H \in \text{HiddenIdSet}$ and such that

$$id \in \text{VisibleId} \quad \Rightarrow \quad id_H \in (\text{HiddenId} \setminus H).$$

Given a set of visible identifiers *VIS* and a set of hidden identifiers *HIS*, we define their associated renaming morphism ρ_{HIS}^{VIS} as follows:

$$\rho_{HIS}^{VIS} = \{id \rightarrow id_{HIS} | id \in VIS\}$$

The visible identifiers of a signature have to build again a signature. As a consequence, hiding a class has to automatically hide all sorts, sort synonyms, maps and functions whose signatures contain this class. Similarly, when hiding a sort constructor and a sort synonym. Given an identifier *id* and a signature Σ we denote by $\text{scons}(id, \Sigma)$, $\text{ssyns}(id, \Sigma)$, $\text{maps}(id, \Sigma)$ and $\text{fncs}(id, \Sigma)$ the set of sort constructors, the set of sort synonyms, the set of maps and respectively the set of functions which contain *id* in their signatures.

³See [BFG⁺93a, BFG⁺93b] for the definition of the standard signature.

⁴Remember, that the set of bound identifiers in a specification *SP* is disjoint from the set of identifiers declared in the signature or *SP*.

If $\Sigma = (CE, SCE, SSE, ME, FE)$ then the above sets are defined as follows:

$$\begin{aligned}
& \text{scons, ssyns, fncs, maps} : \text{VisibleId} \times \text{Sign} \rightarrow \text{Fin}(\text{VisibleId}) \\
& \text{scons}(id, \Sigma) = \{scid \mid id \text{ occurs in } SCE(scid)\} \\
& \text{ssyn}(id, \Sigma) = \{ssid \mid id \text{ occurs in } SSE(ssid)\} \\
& \text{fncs}(id, \Sigma) = \{fid \mid id \text{ occurs in } FE(fid)\} \\
& \text{maps}(id, \Sigma) = \{mid \mid id \text{ occurs in } ME(mid)\}
\end{aligned}$$

The set of all identifiers depending on id — the downward closure of id — is written as $d(id, \Sigma)$. It is defined as follows:

$$\begin{aligned}
id \in IdSet & \Rightarrow d(id, \Sigma) = \{id\} \cup \text{fncs}(id, \Sigma) \cup \text{maps}(id, \Sigma) \cup d(\text{scons}(id, \Sigma), \Sigma) \\
id \in SconSet & \Rightarrow d(id, \Sigma) = \{id\} \cup \text{fncs}(id, \Sigma) \cup \text{maps}(id, \Sigma) \cup d(\text{ssyns}(id, \Sigma), \Sigma) \\
id \in dom(SSE) & \Rightarrow d(id, \Sigma) = \{id\} \cup \text{fncs}(id, \Sigma) \cup \text{maps}(id, \Sigma) \cup d(\text{ssyns}(id, \Sigma), \Sigma) \\
id \in dom(FE) & \Rightarrow d(id, \Sigma) = \{id\} \\
id \in dom(ME) & \Rightarrow d(id, \Sigma) = \{id\}
\end{aligned}$$

This function is extended point-wise to sets of identifiers. We also forbid to hide identifiers from the standard signature Σ_S .

Note: The relation \leq between classes can be considered as an axiom. So hiding a class automatically hides this axiom but it *does not affect* the other classes. Similarly hiding a function does not affect the other functions which use the hidden one in their definition.

$$\frac{HIS, C \vdash \text{sigel-set} \Rightarrow VIS \quad HIS, C \vdash \text{aspecexp} \Rightarrow (\Sigma, Ax)}{HIS, C \vdash \mathbf{hide} \text{ sigel-set in aspecexp} \Rightarrow \rho_{HIS}^{d(VIS, \Sigma)}(\Sigma, Ax)} \{d(VIS, \Sigma) \subseteq dom(\Sigma \setminus \Sigma_S)\}$$

$$\frac{B \vdash \text{sigel} \Rightarrow VIS_1 \quad [B \vdash \text{sigel-set} \Rightarrow VIS_2]}{B \vdash \text{sigel}[, \text{sigel-set}] \Rightarrow VIS_1[\cup VIS_2]}$$

$$\overline{B \vdash \text{opn} \Rightarrow \{\text{opn}\}}$$

$$\overline{B \vdash \text{sid} \Rightarrow \{\text{sid}\}}$$

$$\frac{B \vdash \text{specexp} \Rightarrow (\Sigma, Ax)}{B \vdash \mathbf{SIG}(\text{specexp}) \Rightarrow dom(\Sigma)}$$

Export

$$\langle \text{aspecexp} \rangle ::= \mathbf{export} \langle \text{sigel-set} \rangle \mathbf{in} \langle \text{aspecexp} \rangle$$

Export is similar to hiding. However, in this case the visible identifiers are given in the export list. As before, we must close this list, to a valid, visible signature. This closure operation is written as $u(id, \Sigma)$ — upward closure of id with respect to Σ — and it is defined as below.

Export–Closure Building

$$\begin{aligned}
id \in IdSet &\Rightarrow u(id, \Sigma) = \{id\} \\
id \in SConSet &\Rightarrow u(id, \Sigma) = \{id\} \cup flat(CE(id)) \\
id \in dom(SSE) &\Rightarrow u(id, \Sigma) = \{id\} \cup u(flat(SSE(id)), \Sigma) \\
(id, t) \in FE &\Rightarrow u(id, \Sigma) = \{id\} \cup u(flat(t), \Sigma) \\
(id, t) \in ME &\Rightarrow u(id, \Sigma) = \{id\} \cup u(flat(t), \Sigma)
\end{aligned}$$

Given an expression t , then $flat(t)$ builds the set containing all the class identifiers, sort constructors and sort synonyms which occur in t . Sort variables are ignored. Denote by $dom(\Sigma)$ the set of all identifiers occurring in the signature. Then we define the complement set of $u(VIS, \Sigma)$ as follows:

$$c(VIS, \Sigma) = dom(\Sigma) \setminus (dom(\Sigma_S) \cup u(VIS, \Sigma))$$

The semantic rule for export is then as follows:

$$\frac{HIS, C \vdash sigel\text{-}set \Rightarrow VIS \quad HIS, C \vdash aspecexp \Rightarrow (\Sigma, Ax)}{HIS, C \vdash \mathbf{export} \ sigel\text{-}set \ \mathbf{in} \ aspecexp \Rightarrow \rho_{HIS}^{c(VIS, \Sigma)}(\Sigma, Ax)} \{VIS \subseteq dom(\Sigma \setminus \Sigma_S)\}$$

Parameterized Specifications

$$\begin{aligned}
\langle \text{abstr} \rangle &::= \langle \text{alphanumid} \rangle \mid \langle \text{specabstr} \rangle \\
\langle \text{specabstr} \rangle &::= \mathbf{param} \ \langle \text{spec-list} \rangle \ \mathbf{body} \ \langle \text{aspecexp} \rangle \\
\langle \text{spec-list} \rangle &::= \langle \text{alphanumid} \rangle = \langle \text{specexp} \rangle \ [, \ \langle \text{spec-list} \rangle] \\
\langle \text{specexp} \rangle &::= \langle \text{abstr} \rangle \ (\ \langle \text{arg-lst} \rangle \) \\
\langle \text{arg-lst} \rangle &::= \langle \text{specexp} \rangle \ [\mathbf{via} \ \langle \text{morph} \rangle] \ [, \ \langle \text{arg-lst} \rangle]
\end{aligned}$$

A parameterized specification $(SpP, specexp)$ can be understood as a “macro definition” where

$$SpP = \{id_1 \rightarrow Sp_1\} : \dots : \{id_n \rightarrow Sp_n\}$$

is the list of evaluated formal parameters and $specexp$ is the unevaluated body.

This specification can be applied to a list of actual parameters

$$(\rho_1, SP'_1) : \dots : (\rho_n, SP'_n)$$

only if:

- the signatures of the actual parameter include (modulo renaming) the signatures of the formal parameters i.e. if $(\rho_i|_{\Sigma_i})(\Sigma_i) \subseteq \Sigma'_i$
- the union $\rho = (\rho_1|_{\Sigma_1}) \cup \dots \cup (\rho_n|_{\Sigma_n})$ is again a signature morphism

In that case, the renamed body $\rho(specexp)$ is evaluated in a context where the actual parameters replace the formal ones.

$$\frac{B \vdash spec\text{-}list \Rightarrow SpP \quad B + SpP \vdash aspecexp \Rightarrow SP}{B \vdash \mathbf{param} \ spec\text{-}list \ \mathbf{body} \ aspecexp \Rightarrow (SpP, aspecexp)}$$

$$\frac{B \vdash specexp \Rightarrow SP \quad [B \vdash spec\text{-}list \Rightarrow SpP]}{B \vdash alphanumid = specexp[, spec\text{-}list] \Rightarrow \{alphanumid \rightarrow SP\}[: SpP]} \left\{ \begin{array}{l} [alphanumid \notin \\ dom(SpP)] \end{array} \right.$$

Note: The environment SpP overrides B . We therefore use $B + SpP$ instead of $B \cup SpP$.

$$\begin{array}{l}
B \vdash \mathit{abstr} \Rightarrow \{id_1 \rightarrow SP_1 : \dots : id_n \rightarrow SP_n, \mathit{aspecexp}\} \\
B \vdash \mathit{arg-lst} \Rightarrow (\rho_1, SP'_1) : \dots : (\rho_n, SP'_n) \\
B + \{id_1 \rightarrow SP'_1 : \dots : id_n \rightarrow SP'_n\} \\
\vdash (\rho_1|_{\Sigma_1} \cup \dots \cup \rho_n|_{\Sigma_n})(\mathit{aspecexp}) \Rightarrow SP \\
\hline
B \vdash \mathit{abstr}(\mathit{arg-lst}) \Rightarrow SP
\end{array}
\left\{ \begin{array}{l}
(\rho_i|_{\Sigma_i})(\Sigma_i) \subseteq \Sigma'_i \\
id \in \mathit{dom}(\rho_i) \cap \mathit{dom}(\rho_j) \\
\Rightarrow \rho_i(id) = \rho_j(id)
\end{array} \right.$$

$$\frac{B \vdash \mathit{specexp} \Rightarrow SP \quad [B \vdash \mathit{morph} \Rightarrow \rho] \quad [[B \vdash \mathit{arglst} \Rightarrow SpA]]}{B \vdash \mathit{specexp}[\mathbf{via} \mathit{morph}][[, \mathit{arglst}]] \Rightarrow (\{ \} [+ \rho], SP)[[: SpA]]}$$

Acknowledgments

The authors would like to thank all colleagues that were concerned with the development of SPECTRUM. In particular we thank Rudi Hettler, Franz Regensburger and Oscar Slotosch for stimulating discussions and for their comments to this report. Special thanks go to Manfred Broy for supervising the whole development of SPECTRUM.

Bibliography

- [BFG⁺93a] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language SPECTRUM. An Informal Introduction. Version 1.0. Part I. Technical Report TUM-I9311, Technische Universität München. Institut für Informatik, May 1993.
- [BFG⁺93b] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language SPECTRUM. An Informal Introduction. Version 1.0. Part II. Technical Report TUM-I9312, Technische Universität München. Institut für Informatik, May 1993.
- [GR94] Radu Grosu and Franz Regensburger. The Logical Framework of SPECTRUM. Technical Report TUM-I9402, Institut für Informatik, Technische-Universität München, 1994.

Appendix A

Context Free Syntax

A.1 In the Small

In this section we define the raw structure of our language “in the small” by giving a context-free grammar in EBNF style.

```
⟨specbody⟩ ::= { ⟨decls⟩ }
⟨decls⟩    ::= { ⟨signature⟩ ; | ⟨axioms⟩ ; }*
```

Signatures

```
⟨signature⟩ ::= class ⟨id⟩ [subclass of { ⟨id⟩ // , }+]
              | ⟨id⟩ subclass of { ⟨id⟩ // , }+
              | sort ⟨sid⟩ { ⟨sid⟩* | :: ⟨classexp⟩ }
              | { ⟨sid⟩ // , }+ :: ⟨classexp⟩
              | sortsyn { ⟨sid⟩ }+ = ⟨sortexp⟩
              | SIG ( ⟨specexp⟩ )
              | { ⟨id⟩ // , }+ : [⟨context⟩] ⟨sortexp⟩ to ⟨sortexp⟩
              | { ⟨inf-id⟩ // , }+ : [⟨context⟩] ⟨sortexp⟩ × ⟨sortexp⟩ to ⟨sortexp⟩ [⟨prio⟩]
              | { ⟨id⟩ // , }+ : [⟨context⟩] ⟨sortexp⟩
              | { ⟨inf-id⟩ // , }+ : [⟨context⟩] ⟨sortexp⟩ × ⟨sortexp⟩ → ⟨sortexp⟩ [⟨prio⟩]
⟨classexp⟩  ::= [( { ⟨id⟩ // , }+ )] ⟨id⟩
⟨prio⟩     ::= prio ⟨num⟩ [ : { left | right } ]
```

Sort Expressions

```
⟨sortexp⟩  ::= ⟨sortexp1⟩
              | ⟨sortexp1⟩ → ⟨sortexp⟩ (Functional Sort)
⟨sortexp1⟩ ::= ⟨sortexp2⟩
              | ⟨sortexp2⟩ { × ⟨sortexp2⟩ }+ (Product Sort)
⟨sortexp2⟩ ::= ⟨asort⟩ | ⟨sid⟩ { ⟨asort⟩ }+
⟨asort⟩    ::= ⟨sid⟩ | ( ⟨sortexp⟩ )
```

Sort Contexts

```
⟨context⟩ ::= { ⟨scontext⟩ // , }+ ⇒
```

$\langle \text{scontext} \rangle ::= \{ \langle \text{sid} \rangle // , \}^+ :: \langle \text{id} \rangle$

Axioms

$\langle \text{axioms} \rangle ::= \mathbf{axioms} [\langle \text{varlist} \rangle] \{ \{ \langle \text{id} \rangle \} \langle \text{exp1} \rangle ; \}^* \mathbf{endaxioms}$
 $\quad | \langle \text{simplesorts} \rangle \mathbf{generated\ by} \langle \text{opns} \rangle$
 $\langle \text{varlist} \rangle ::= [\langle \text{scontext} \rangle] \{ \{ \forall | \forall^\perp \} \langle \text{opdecls} \rangle \}^+ \mathbf{in}$
 $\langle \text{opdecls} \rangle ::= \{ \langle \text{sopdecl} \rangle // , \}^+ | \{ \langle \text{id} \rangle // , \}^+$
 $\langle \text{sopdecl} \rangle ::= \{ \langle \text{id} \rangle // , \}^+ : \langle \text{sortexp} \rangle$
 $\langle \text{simplesorts} \rangle ::= \{ \langle \text{sid} \rangle \{ \langle \text{sid} \rangle \}^* // , \}^+$

Expressions

$\langle \text{exp1} \rangle ::= \langle \text{exp2} \rangle$
 $\quad | \{ \forall | \forall^\perp | \exists | \exists^\perp \} \langle \text{opdecl} \rangle . \langle \text{exp1} \rangle$
 $\quad | \lambda \langle \text{pat} \rangle . \langle \text{exp1} \rangle$
 $\langle \text{exp2} \rangle ::= \langle \text{exp3} \rangle | \langle \text{exp2} \rangle \langle \text{id} \rangle [: \langle \text{asort} \rangle] \langle \text{exp1} \rangle$
 $\langle \text{exp3} \rangle ::= \langle \text{aexp} \rangle | \langle \text{exp3} \rangle \langle \text{aexp} \rangle$
 $\langle \text{aexp} \rangle ::= \langle \text{opn} \rangle | (\langle \text{exp1} \rangle) | (\langle \text{exp1} \rangle \{ , \langle \text{exp1} \rangle \}^+) | \langle \text{aexp} \rangle : \langle \text{asort} \rangle$
 $\langle \text{pat} \rangle ::= \langle \text{id} \rangle [: \langle \text{sortexp} \rangle]$
 $\quad | (\langle \text{id} \rangle [: \langle \text{sortexp} \rangle] \{ , \langle \text{id} \rangle [: \langle \text{sortexp} \rangle] \}^+)$

Identifiers

$\langle \text{id} \rangle ::= \langle \text{alphanumid} \rangle | \langle \text{symbolid} \rangle | \langle \text{num} \rangle | \langle \text{charconst} \rangle | \langle \text{string} \rangle$
 $\langle \text{inf-id} \rangle ::= \langle \text{inf-alphanumid} \rangle | \langle \text{inf-symbolid} \rangle$
 $\langle \text{sid} \rangle ::= \langle \text{id} \rangle_{\{ \rightarrow, \times, -, >, * \}}$
 $\langle \text{opn} \rangle ::= \langle \text{id} \rangle | \langle \text{inf-id} \rangle$
 $\langle \text{opns} \rangle ::= \{ \langle \text{opn} \rangle // , \}^+$

A.2 In the Large

$\langle \text{system} \rangle ::= \langle \text{syspart} \rangle [\text{system}]$
 $\langle \text{syspart} \rangle ::= \langle \text{alphanumid} \rangle = \{ \langle \text{specexp} \rangle | \langle \text{sigmorph} \rangle | \langle \text{specabstr} \rangle \}$

Signature Morphisms

$\langle \text{sigmorph} \rangle ::= [\langle \text{rename-list} \rangle]$
 $\langle \text{rename-list} \rangle ::= \langle \text{rename} \rangle [, \langle \text{rename-list} \rangle]$
 $\langle \text{rename} \rangle ::= \langle \text{inf-id} \rangle \mathbf{to} \langle \text{inf-id} \rangle | \langle \text{id} \rangle \mathbf{to} \langle \text{id} \rangle | \langle \text{sid} \rangle \mathbf{to} \langle \text{sid} \rangle$
 $\langle \text{morph} \rangle ::= \langle \text{alphanumid} \rangle | \langle \text{sigmorph} \rangle$

Structured Specifications

$\langle \text{specexp} \rangle ::= \langle \text{aspecexp} \rangle [+ \langle \text{specexp} \rangle]$
 $\quad | \mathbf{abstr} (\langle \text{arg-lst} \rangle)$
 $\langle \text{arg-lst} \rangle ::= \langle \text{specexp} \rangle [\mathbf{via} \langle \text{morph} \rangle] [, \langle \text{arg-lst} \rangle]$
 $\langle \text{aspecexp} \rangle ::= \mathbf{rename} \langle \text{specexp} \rangle \mathbf{by} \langle \text{morph} \rangle$
 $\quad | (\langle \text{specexp} \rangle)$
 $\quad | \mathbf{hide} \langle \text{sigel-set} \rangle \mathbf{in} \langle \text{aspecexp} \rangle$
 $\quad | \mathbf{export} \langle \text{sigel-set} \rangle \mathbf{in} \langle \text{aspecexp} \rangle$

		{ enriches <specexp>; <decls> }
		<aspecexp> <specbody>
<sigel-set>	::=	<sigel> [, <sigel-set>]
<sigel>	::=	<opn> <sid> SIG (<specexp>)

Parameterized Specifications

<abstr>	::=	<alphanumid> <specabstr>
<specabstr>	::=	param <spec-list> body <aspecexp>
<spec-list>	::=	<alphanumid> = <specexp> [, <spec-list>]