Master Thesis

Syntax–Directed Editor for RSL

Michael Suodenjoki

Department of Computer Science
Danish Technical University

28 February, 1994

[0]**RSL** and **RAISE** are trademarks of Computer Resources International A/S.
UNIX is a registered trademark of UNIX Systems Laboratories.
Microsoft Windows 3.1 and **Win32s** are trademarks of Microsoft Corporation.
Symantec C++ Professional is a trademark of Symantec Corporation.
Synthesizer Generator and **GrammaTech** are trade names of GrammaTech Inc.

Preface

This report presents my Master Thesis in obtaining the Master of Science Degree at Danish Technical University. The report describes a syntax-directed editor for the RAISE[1] Specification Language.

The master's thesis reflects six months of work from the 1st of September 1993 to the 28th of February 1994.

The report is written in English in the hope that it will reach a wider audience. Even though English is not my native language, I hope that the reader will bear with eventually ill-formulated sentences.

Acknowledgements

I would like to thank all who have contributed to produce LaTeX [Lamport 86] styles, in which this report is written: Trevor J. Darrell for Postscript Figure environment psfig, Van Jacobsen and Jerry Leichter for the C++ language style environment lgrind, Piet van Oostrum for the fancy headings style environment fancyheadings and CRI A/S for the RSL environment rslenv.

I thank Thomas H. Hansen for a program to capture Windows screens and saving them to Postscript.

I thank Annemette Overgaard for her careful reading of the manuscript.

I thank Chris George and Peter Haff at CRI for letting me see their RSL syntax input grammar for YACC [Johnson 79].

I also want to thank all those who have been communicating in a very friendly way on the USENET News Net and have helped me getting different kind of useful information.

Finally I thank my two supervisors Bo Stig Hansen and Hans Bruun.

Lyngby, Copenhagen, February 1994

Michael Suodenjoki

[1] RAISE - Rigorous Approach for Industrial Software Engineering, ESPRIT RAISE (315) and LaCoS (5383)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report describes a kernel for a syntax-directed editor for the RAISE Specification Language (RSL). The editor is implemented in Windows 3.1 and is called RSLED, which is an acronym for RAISE Specification Language EDitor.

A syntax-directed editor is an editor which recognizes the underlying language of text, which can be edited. The underlying language can be anything which can be written with text e.g. programming languages, specification languages or even natural text. The main advantage of using such an editor is that it can tell whenever the user is typing something wrong or even not allowing the user to type something wrong.

The reader should have a computer science background, know RSL, the C language, and the basic concepts in object oriented programming in C++. Furthermore, knowledge of programming Windows should be an advantage.

1.1 Notation

In the report the following notations are used:

- roman type style is used for normal text,
- *italics type style* is used for introducing / defining new concepts, or for emphasizing certain words,
- sans serif type style is used to refer to RSL syntax,
- **bold type style** is used to refer to functions and variables in specifications and source code,
- `typewriter type style` is used to show user input or program output/source.

I have used the term *grammar* for general context free grammars [Chomsky 56], which can be described in BNF *Backus Naur Form* [Naur 63].

In RSL specifications all maps are considered to be deterministic, e.i. an element in the domain of the map does *not* map to different elements in the range.

When referring to types or functions in RSL specifications, the notation (*x.yy*) is used to refer to specification *x* and line number *yy*.

Sometimes a range of functions, with the same prefix in the name, is referred to as **function_name_prefix_XX**.

In the report Windows, with capital W, is used to refer to Microsoft Windows. Windows in lowercase refer to rectangular areas on the screen.

1.2 The project

Background

The project is an off-spring of an area in which I am very interested: Transformational Programming. For using transformational programming effective user environments should be available. These environments include syntax-directed editors. In my perspective, this project is just an examination of capabilities of how to program syntax-directed editors.

Another reason, why especially a syntax-directed editor for RSL is wanted, is that users at Danish Technical University want to be able to edit RSL specifications at home at their own personal computers. Currently, the students use the syntax-directed editor *eden* on the Unix operating system. Eden is programmed by CRI A/S, using the Cornell Synthesizer Generator (CSG) [TeitelBaum, Reps 89] from the firm GrammaTech Inc. [GrammaTech 92]. Since this editor exists a large part of RSLED is inspired by it.

Two previous syntax-directed editors has been developed at Department of Computer Science [Hurvig 85] and [Ekner, Hørlyck 87]. Those projects are very similar to this and RSLED are also inspired by them.

The intention have been to develop, on one hand a fully running syntax-directed editor for RSL, and on the other hand also to give a full specification of it at first. These two logical parts are each large enough to be two separate projects, so they cannot be fully examined/developed in this project.

Preferably, a stand-alone project, specifying the complete (generic) syntax-directed editor system, succeeded with a stand-alone project that implements the hole system would have been preferred.

Demands

This leads to the technical demands of the project:

- A specification (in RSL) of the basic kernel in an syntax-directed editor is wanted - either generic or specific to RSL.
- Basic unparsing with the possibility of extended unparsing structures like *elision* and *adaptive formatting* are wanted.
- The kernel should be portable to other environments e.g. Unix.

- The user interface should be graphical using windows, dialogs, menus and buttons. Preferably it should be compatible with *eden*.
- The editor should be implemented under Microsoft Windows 3.1 using Win32s.
- The editor should be implemented in the C++ programming language.

Programming under DOS/Windows can be difficult due to certain inexpediencies in the construction of the hardware (the chip). These a mainly attached to a memory limit of 64kB of data or code. Multiple data and code segments are possible when one compiles under different memory models. The large model should allow this. As these inexpediencies exist, it is decided that the editor should be programmed under Win32s, which is a application interface (API) that supports full 32-bit memory addressing and have a flat memory model. This would also lead to easier transformation to other 32-bit platforms such as Unix and OS/2.

The use of the C++ language arises several design problems. Should the system be developed using object oriented methods or pure C ? Using Windows and C++ leads the thoughts to an object oriented user interface. Several class libraries exist on the market. Should they be used ?

At an early moment during the project, it was the opinion that a lot could be learned of using the basic elements in the available languages and in the available application interface. It was decided that OOP should be used to program the basic kernel in the editor, while the user interface should be programmed in pure C. At a later stage, it is much easier to convert to an object oriented user interface than to convert the basic kernel from pure C to C++. This also calls for a natural modularization in kernel and user interface.

In practical, the project was handled by using a simple form of time schemes which stated time intervals of different parts of the projects. A simplification of this is:

2 weeks	for introduction and literature studying
9 weeks	for specification
2 weeks	for prototype development
8 weeks	for implementation, verification and error correction
4 weeks	for writing the report

The compiler used for programming RSLED was Symantec C++ Professional, which supports both Windows and Win32s development. Comments on this compiler is located in appendix D.

1.3 History of syntax-directed editors

In the early sixties when computers began to gain so much power that they could present their output of calculations on teletype (TTY) terminals, the need for user friendly interfaces arised. Programming began not only to be a question of inputting language sentences to an interpreter, but also of inputting of collective language sentences to a compiler. The compiler needed a text file which could be edited by a text editor. These editors was at first very simple *line editors* , but was rapidly developed to *character editors* and *screen editors*, where the user could move all around the screen with a text cursor, insert characters, delete characters and move pages.

Even though the editor was used for editing files intended to be given to a compiler, the editor could also be used as a document text processor. As the number of high-level programming languages grew, the need for language-based editors was recognized [Allison 83],[Hansen 71]. A *language-based editor* is an editor which recognizes the structure of the underlying language. The

editor pays attention to the syntax of the language and could as such be called a *syntax-directed editor*. The main difference between a syntax-directed editor and a normal text editor is the internal representation of the edited text. In a text editor, the text is often represented in a sequence of characters and in a syntax-directed editor the text (the program) is often represented by a *tree*, which reflects the syntactic structure of the edited program. Existing editors are either *structure editors* that manipulate trees of a language's abstract syntax, such as Emily [Hansen 71], MENTOR [Donzeau-Gouge et al. 80], and the Gandalf editor [Habermann, Notkin 86], *text editors* that employ an incremental parser, such as the CAPS editor [Wilcox et al. 76], or *hybrid editors* that combines both techniques, such as the Cornell Synthesizer Generator [TeitelBaum, Reps 89].

Editor	Year(s)	Type	Language	Reference
Emily	1971	structure	PL/1	[Hansen 71]
Emacs		text	several	
POE	1979-1984		Pascal	[Fisher et al. 84]
Mentor	1975	structure	Pascal	[Donzeau-Gouge et al. 80]
CAPS	1976	text		[Wilcox et al. 76]
Gandalf	1979-1981	structure		[Habermann, Notkin 86]
COPE	1981			[Archer, Conway 81]
PECAN	1984		alg. lang.	[Reiss 84]
PSG	1985			[Bahlke, Snelting 85]
GENIE	1985	hybrid, programmable	several	[Hurvig 85]
CSG	1984-88	hybrid, programmable	several	[TeitelBaum, Reps 89]
Centaur	1990	hybrid programmable	several	[Borras, Clément 90], [Clément 90]

Table 1.1: Overview over some language-based editors.

The general tendency for syntax-directed editors is that they have more and more advanced features such as semantics checking, elision and graphical user interfaces.

1.4 Views on syntax-directed editors

Are syntax-directed editors helpful ?

This is a question which is always necessary when building software environments. For the developers of these editors, the question may be answered with a big Yes, as the argument would be that they would never have started the development if they had not believed in the usefulness. But for the users of such environments the answer would properly be different. Trivial tasks in syntax-directed editors are often "expensive". New users must learn to perceive the edited text as a tree and understand how to move around in the tree. Some think this is tedious, when they are used to move freely around in the edited text. This can be mitigated by allowing textual input.

Important aspects for users are: easy functionality, graphical interfaces using mouse and easy text editing. A menu with all command reachable, helps learning of the system.

Syntax-directed editors are by some considered as a good tool for learning the syntax and structure of new languages. For more experienced users syntax-directed editors are irreplaceable. Often they offers more complex transformations which support program development.

Several industrial products exist on the market, but none of them has reached a widespread success. At least not in the lower end of the market - on the personal computers. At universities and firms related to universities syntax-directed editors are more common. It is considered that it is just a question of time (2-5 years), before syntax-directed editors dominate the market. Several programming environments already use some features from syntax-directed editors, such as using different colors for different syntactic constructs.

1.5 Structure of the report

The report is divided into three logical parts:

PART I: Specification Chapter 2-5.
: This part specifies the abstract syntax and the abstract-syntax tree. Movement and basic editing in abstract-syntax trees are also specified. Basic concepts and specification of unparsing are also described. Furthermore, parsing and textual input are considered.

PART II: Implementation Chapter 6-11.
: This part describes the implementation of RSLED. The interconnection and the structure of the units are explained. Each unit in the program is explained in details in each chapter.

PART III: User manual Chapter 13.
: This part contains the manual of how to use RSLED.

Furthermore, several appendixes contain RSL language specific informations like syntax and precedence of operators. The report ends with appendixes of source files.

The conclusion of the project, future improvement and extensions and final remarks are located in chapter 12.

Part I

Specification

Chapter 2

Abstract Syntax

2.1 Abstract Syntax

The abstract syntax is the underlying data structure for the entire editor. It abstracts away from keywords and other terminal symbols. It should reflect the language used in the editor, and as it can be seen in appendix A, the RAISE Specification Language can be described by an extended form of BNF notation. The extension reflects the possibility to describe optionals and lists in different forms. These two extensions should also be captured by the specification of the abstract syntax.

At this point one design decision to consider, is however the specification of the abstract syntax should be a general one e.i. one which cover all grammars[1] (languages), or one which is specific to a certain grammar (language). Obvious the program would be much more adaptive to new languages if the specification is general, but the specific specification could be much more precise and easier to understand. Furthermore, the specific specification have the advantage of being specified in an abstract way using algebraic constructs. All though the specific specification has these advantages, the general solution must be considered the best one, due to the possibility to extend the hole editor system to a generic editor.

Normally a language is described by a grammar[2] which consists of an *alphabet* and a set of *productions*. The *alphabet* consists of *terminal* and *nonterminal* symbols. *Productions* maps *nonterminals* over in sequence of *symbols*, but the same *nonterminal* can be mapped over in different symbol-sequences each named uniquely by a *production identifier*. This conform to variant productions in grammars. A *Symbol* could either be a *terminal* or a *nonterminal*.

However, this specification does not capture optional-, list- and optional list-constructs satisfactory, even though they can be modelled using empty productions and recursive references. These constructs are important in syntax-directed editors.

Another possibility is to describe the abstract syntax in a way inspired by CSG[3]. In CSG the editor programmer specify the abstract syntax with a collection of productions of the form:

$$x_0 : op(x_1 x_2 \ldots x_n);$$

[1] LALR(1) grammars
[2] BNF inspired
[3] this is again inspired by term algebra.

where *op* is an *operator* name and each x_i is a nonterminal of the grammar, or, as they call it, the name of a *phylum*. For each production it can be specified if the phylum x_0 on the left-hand side represent a list, an optional element or an optional list (explained later).

This can be modelled as:

Is_wf_Productions (2.28) state that all phyla in the productions should be defined e.i. be in the set of allowable phyla, and that two different phyla must not map over in the same two (Operator $\overrightarrow{m}$ Production) map.

If we compare the two proposals to the abstract syntax (scheme AS0 and AS1), we notice that *operators* conform to *production identifiers*, and *phyla* nearly conform to *symbols*, but with the big difference that no terminal symbols are represented. The terminal symbols are totally hidden as expected with a real abstract syntax.

The abstract syntax for the (RSL) language should also capture incomplete syntaxes e.i. unfinished specifications containing *placeholders*. Placeholders identify locations at which additional components can be inserted.

The abstract syntax defined in scheme AS1 is used in RSLED.

2.2 Abstract-syntax tree

When handling the edited text in the editor it is convenient to represent the text as a *tree*[4]. Normally trees are *empty* (the null tree) or consists of some *nodes* containing information (the node info) and some subtrees. Subtrees which are empty are called *leafs*.

Subtrees are named after genealogical trees where a subtree in general is called a child. Each child have a parent and some siblings. Below in figure 2.1 one can see the connection between children, parents and siblings.

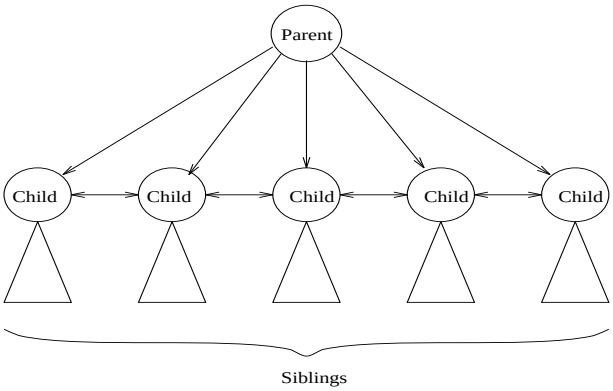

Figure 2.1: Figure of interconnections in a genealogical tree

The node contents in the tree is, at this point, not fully known. It is known that each node should be uniquely identified by a node identification. The identificator uniquely represent a production rule in the grammar. Due to this relation to the abstract syntax the tree are called an *abstract-syntax tree*.

[4] This is recognized by several other structure editors, see table 1.1

Since at this point it is unknown how nodes are represented the uncertainty can be captured by 'minimizing' what is said about nodes. Furthermore it is stated that the destructor nodeid will return the node identifier.

A node in the tree can in some situations be: an *optional node* e.i. a node which dynamically are added or deleted to the syntax tree; a *list node* e.i. the node could consist of one or more sub-nodes, or an *optional list node* e.i. the node could consist of zero, one or more sub-nodes.

Furthermore a node could be selected by the user. It is also noticed that a node not always should be selected e.g. if the tree consist of a range of nodes with only one subtree, it would be teedius for the user to move through all these nodes. This can be illustrated by an example. In the figure 2.2 below, a part of the RSL syntax are drawn. The formal_array_parameter is represented by a node, which only has one subtree - the brackets are only drawn for help understanding - and if the user while traversing through the tree each time should stop at the node for formal_array_parameter, he must make another move to traverse to the node for typing-list. The notation of a resting place

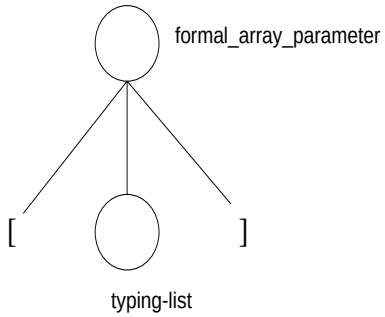

Figure 2.2: An example of a node representing formal_array_parameter.

are therefore invented (all though inspired by CSG). A resting place is a node which can be selected e.i. a node where a traversal can stop.

All this can be captured by the following specification of a node:

All this can be captured by the following specification of the abstract-syntax tree[5]:

The uncertainty of how one represents trees are captured by using the wildcard constructor [6].

The root of the entire syntax tree is kept in the variable **root** (5.5).

The wildcard constructor is again used to capture uncertainty of how trees are constructed. It is not in the interest to generate the induction axiom, because it tells something about limitation of trees. An induction axiom restricts the variant types (null and _) to contain only values generated by the constructors, mentioned in the variant definition. If one does not omit the induction rule, it would be impossible to write down elements of the type !

Furthermore, the wildcard variant is used. This means more formally that no value definition of a (tree) constructor is generated and as consequence no axioms for the destructors are generated.

The subtrees, which at this (implementation) level of abstraction, could be comprehended as

[5] Several more abstract forms can be chosen (see [Hansen 71] and [Hurvig 85]), but since we are not concerned with different abstraction levels of specifications, we chose one which are lying close to implementation

[6] This mean more formally that no value definition are defined and that no induction axiom is generated [George et al. 92, see pp. 98].

pointers[7] to other trees. This means, for example, that a child of a parent should contain (have a pointer to) the child. This gives reason to define when a abstract syntax tree is well-formed. Below the well-formedness function **is_wf** uses a utility function **isintree**, which examines if one tree contains another tree.

The lines from (6.24) to (6.35) could be comprehended as appropriate destructor axioms (recall that these were not defined, because the wildcard constructor was used). The lines from (6.35) to (6.38) ensure the right understanding of the destructors as pointer models e.g. the **parent(firstchild(n)) = n** states that if for a tree you fetch the first child and then its parent, then we end up with the same tree.

The well formedness function does also state, that if a node is selected, it is also a resting place (6.42). Furthermore, all list nodes does at least has a minfixed value of 1 (6.43).

Optional nodes

Optional structures in the grammar are captured by having information for each node whether they represent optional structures. In the syntax tree optional nodes are nodes which automatically can be expanded or deleted.

List nodes

List constructs in the syntax are captured by having information for each node whether they representing list structures. In the syntax tree there are two different ways of representing lists - by a flat structure or by a binary structure (see figure 2.3). The flat structure is easy to manipulate

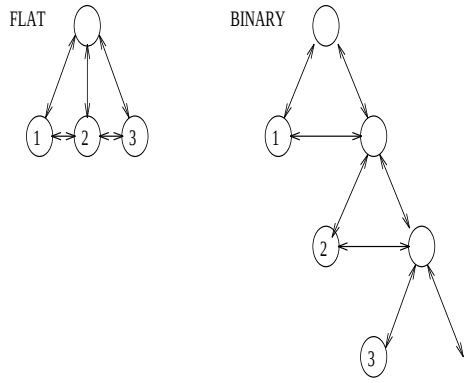

Figure 2.3: Flat contra binary list structure.

e.g. insertion of a new node, and it is easy to define movement between them. The binary structure is not so easy to manipulate e.g. a hole subtree must be manipulated[8].

To control the minimum amount of elements in a list, the node carry information about the minimum number of fixed list elements. This captures that some syntactic structure, representing lists, in e.g. RSL always has at least 2 elements e.g. name_or_wildcard-choice2. For other languages it is easy to extend to other fixed lengths.

[7] known from imperative languages
[8] In the implementation of RSL the flat structure is choosed

Chapter 3

Syntax-tree Manipulation

In this chapter it is specified how to move around in the abstract-syntax tree and how to modify the abstract-syntax tree.

When specifying the basic kernel of RSLED it have been in the mind that the implementation should use pointers and C++ classes. This had influenced the specification to capture as much of the object oriented paradigm as possible. It have been naturally to use RSL classes as representing different C++ classes, but since RSL does not have a dynamical object feature build into the language e.g. one have to specify a program using dynamical objects, then one has to model a kind of heap and functions like new and delete on the heap. This could have been done, but it is lying outside the frame of this report[1]. Even though this not have been done several object oriented features can be captured. One of the most important ones is known as virtual functions (or late binding). A virtual function, in a class, is a function, which at a later stage, can be redefined - overwritten in another class. Calls to the function in the first class would call the appropriate (overwritten) function in inherited classes. Such virtual functions is in the specification just specified with an signature, but not further defined, since they would/could be defined (overwritten) by subsequent classes. The basic inherit mechanism is easily modelled by using the RSL possibility of extending classes.

Informally, it is understood that all functions specified in the following sections, all maintain the well-formness of trees after their execution.

3.1 Basic manipulation

This section deals with the lower levels of functions to manipulating the tree.

At first are defined some lower level operations of inserting, deleting and comparing trees. Most of these changes the global tree which is represented by the variable **root** (5.5).

The two inserting functions **insert_tree_first** (7.7) and **insert_tree_last** (7.10) both insert new children either as first child or last child. In the function signature the first parameter (Tree) represents the tree in which the child (the second parameter) should be inserted. The other two inserting functions, (7.8) and (7.9), inserts siblings as either as left (before) or right sibling (after).

[1] A proper examination of the object oriented paradigme would have been necessary

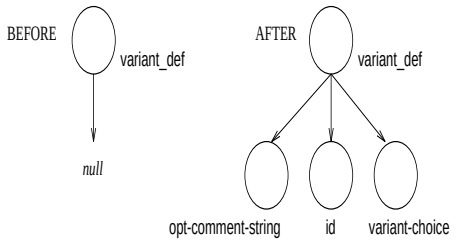

Figure 3.1: Example of calling **make_subtree** on a variant_def.

The two deleting functions **delete_XX**, (7.11) and (7.12), respectively delete a complete subtree or only a single node in the tree.

Two functions dealing with cutting and pasting are defined (7.13) and (7.14). They are later used for specifying cutting and pasting of trees.

The two compare functions (7.16 and 7.17) compare respectively trees and nodes.

Two very important functions **make_subtree** (7.18) and **make_child** (7.19), are underspecified. The function **make_subtree** should extend a tree with (appropriate) children. The children should correspond to the right syntactic constructs in the abstract syntax, e.g. if the tree given as parameter corresponds to the syntactic construct variant_def (see figure 3.1), then **make_subtree** would be defined as creating and inserting three children corresponding to the three syntactic constructs for opt-comment-string, id and variant-choice. The function **make_child** is used when handling lists and/or optional nodes, e.i. whenever a new child of a list node and/or an optional node should be created then **make_child** is called. For example, a node would represent the syntactic construct scheme_def-list have **make_child** defined as returning a new node representing a scheme_def (see also RSL syntax in appendix A). So multiple calls of **make_child** would give multiple 'instances'[2] of child nodes.

Another function **count_children** (7.15) calculate the number of children of a node. It is primarily used when handling lists.

3.2 Movement operations

Movement in the syntax tree is one of the most used operations by the user and reflects cursor movement as in normal flat editors. In a syntax-directed editor the user does not move a cursor but a selection. The selection represents the current position in the tree.

A movement operation in the tree is characterized as not changing the tree. Only the selection is changed.

3.2.1 Specification of movement operations

To explain the different possibly movements, the notation of genealogical trees is used e.i. parent, siblings and children. In figure 3.2 below a subtree, where node 4 is selected (the current selection), is presented. Node 1 is the parent, nodes 2,3, 5 and 6 are siblings and nodes 7,8 and 9 is children. A

[2]this expression is used even though memory not is modelled.

filled circle represents an optional node. Below in table 3.1 one can see which node each command leads to.

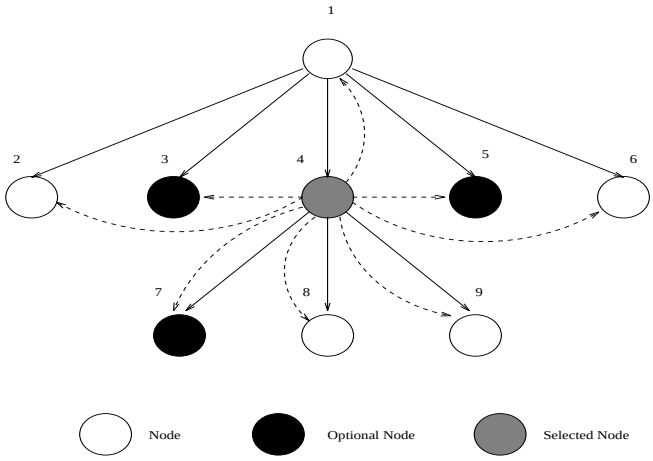

Figure 3.2: Tree movement.

Command	Leads to node
ascend-to-parent	1
backward-preorder	9
backward-sibling	2
backward-sibling-with-optionals	3
backward-with-optionals	1
forward-preorder	8
forward-sibling	6
forward-sibling-with-optionals	5
forward-with-optionals	7

Table 3.1: Table of movements from selected node 4 in the figure above.

The two main strategies to traverse the tree are *forward preorder* and *backward preorder*[3]. The forward preorder traversal is a left-to-right, depth-first strategy and backward preorder are a right-to-left, depth-first strategy. These two traversals are usually used in syntax-directed editors (see e.g. [Hurvig 85] or [TeitelBaum, Reps 89]).

The final goal in this section is to specify movement operations in the tree. First some basic functions must be specified, which could be used when specifying the movement operations.

The first ones are dealing with optional nodes. To ease the understanding of the specification the parameters of one function are briefly explained. The function **insert_optional** (8.7,8.12) has 3 parameters: a **tree** which is the node to possibly extend, a boolean parameter **opt** which is true whenever optional nodes should be inserted and a boolean parameter **flag**, which should be true if the (root) tree in any way is changed during a movement, e.i. new nodes are inserted or some nodes are deleted.

Insertion of children of optional nodes must only be done under certain conditions: **opt** must be true, the node to consider must be optional and it must not have any children[4]. That ensures

[3] both strategies are known from other areas of software engineering e.g. searching
[4] other trivial conditions must also be true

that the tree can be expanded. If all conditions are true then the node is expanded with a new child which is fetched by calling **make_child** (7.19).

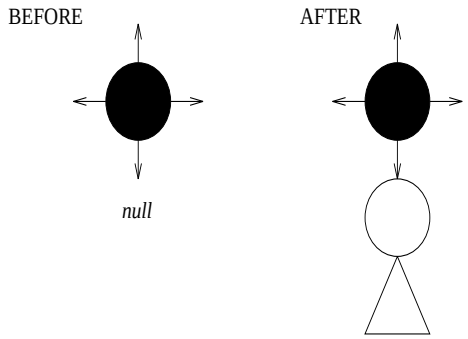

Figure 3.3: Insertion of a child of a optional node.

There is also functions dealing with list nodes. Since it should be possible to insert children of list nodes in between each other, several insertion function must be defined. Below in (9.7-9.9) three insertion function are defined. The parameters have the same meaning as the parameters for **insert_optional**. New children of list nodes are only inserted if certain conditions are true. For example it must not be possible to insert a child of a list node in between two children of list nodes if one of the two already is a 'unused' child. An unused child is comprehended as a tree (the one returned by (make_subtree)) which is not further expanded[5]. To examine if that is the case the specification uses both calls to **make_subtree** and **compare_tree**.

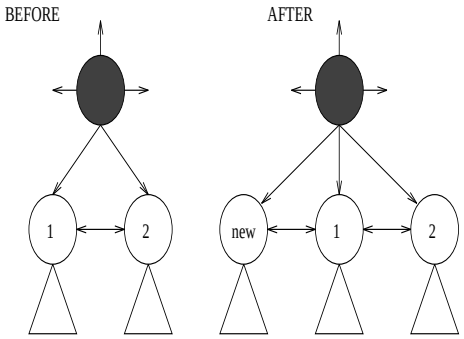

Figure 3.4: Insertion of a (first) child of a list node.

The deletion of children of list nodes (**delete_listnode** (10.55)) must ensure that lists (not optional lists) are non-empty, or at least have the minimum number of children required by **min-fixed**. The minimum fixed number of children of a list node are carried in the node information. This can be controlled by using **count_children** in conjunction with comparison of trees.

Next to define is a variable **selected**, which refers to the currently selected tree. To select and un-select a tree two functions **select** (10.11) and **unselect** (10.10) are defined.

It is now ready to specify the two general traversal algorithms, forward_preorder_traversal and backward_preorder_traversal. They control how one traverse through the syntax tree while possibly expanding or pruning it. The two parameters **opt** and **bypass** has special meanings. The **opt** (optional) parameter specifies if optionals or new (children of) list nodes should be inserted. The **bypass** parameter specify if the traversal should jump over children e.i. move to next/previous sibling.

[5] in CSG they call it a completing term.

The **forward_preorder_traversal** function is explained. It is expected that the traversal functions are called with the current selection, since the user issue movement commands from the selection. First is the selection unselected. Then a loop is entered which traverses through the tree. The loop stops whenever a reached node is a resting place or it is an optional node. Inside the loop it is first examined if the three should be expanded with a new child (10.52), because it is optional or a list node. Then it is examined in sequence, if it possible to move to the first child (10.53) and then to the next sibling (10.57). If this was not possible, the traversal continues back up in the tree (through parents) and tries each time to move to the next sibling (10.65). After a movement, the old position may need to be deleted and new optional nodes may need to be inserted. This is done by calling **deleting**, which is a local utility function, and by calling **insert_optional**. After the loop is finished, the reached node is selected.

The movement operations are "specified" first informally:

forward-preorder Change the selection to the next resting place in a forward preorder traversal of the abstract-syntax tree. Do not stop at nodes for optional constituents.

backward-preorder Change the selection to the previous resting place in a forward preorder traversal of the abstract-syntax tree. Do not stop at nodes for optional constituents.

forward-with-optionals Change the selection to the next resting place in a forward preorder traversal of the abstract-syntax tree. Stop at nodes for optional constituents.

backward-with-optionals Change the selection to the previous resting place in a forward preorder traversal of the abstract-syntax tree. Stop at nodes for optional constituents.

forward-sibling Bypass all resting places contained within the current selection and advance to the next sibling in a forward preorder traversal of the abstract-syntax tree. If there is no next sibling, ascend to the enclosing resting place and advance to its sibling, etc. Do not stop at nodes for optional constituents.

backward-sibling Bypass all resting places contained within the current selection and advance to the previous sibling in forward preorder traversal of the abstract-syntax tree. If there is no next sibling, ascend to the enclosing resting place and advance to its previous sibling, *etc.* Do not stop at place-holders for optional constituents.

forward-sibling-with-optionals Same as **forward-sibling**, but stopping, in addition, at nodes for optional constituents.

backward-sibling-with-optionals Same as **backward-sibling**, but stopping, in addition, at nodes for optional constituents.

ascend-to-parent Change the selection to the closest enclosing resting place.

Now the movement functions are ready to be specified. They use heavily the two traversal functions from TREE5.

3.3 Modify operation

Another central ability in the editor is to modify the syntax tree e.g. expansion and pruning of the abstract-syntax tree.

3.3.1 Transformation schemes

To expand the abstract-syntax tree one uses *transformation schemes* telling how nodes in the tree can be expanded with an entire subtree.

The transformations are kept in a map, with the domain of allowable node identifications, e.i. those nodes which can be transformed. These nodes are typical nodes representing variant productions in the abstract syntax. It is captured by the well-formness of the transformation map.

A transformation is just a question of calling **insert_tree_first** with the right tree, which is fetched from the transformation table (12.23).

3.3.2 Sub-tree manipulation

In modifying the abstract-syntax tree operations, like copying, pasting and deleting must be present so that the editor is user-friendly and compatible with CSG generated editors. Deletion (pruning) is already specified in (7.11).

Structural Editing

Structured modification follows a cut-and-paste paradigm. Only whole, well-formed substructures can be removed and inserted.

Informally explanation of cut and paste functions

cut-to-clipped Move the selection of the current buffer to clipbuffer. The new selection becomes a place-holder at the point from which the selection was removed. The previous contents of clipbuffer are lost.

copy-to-clipped Copy the selection of the current buffer to the clipbuffer. The previous contents of CLIPPED are lost.

paste-from-clipped Copy the contents of clipbuffer into the buffer at the current selection, which must be a place-holder. In clipbuffer, a place-holder term replaces the previous contents.

Chapter 4

Unparsing & Windowing

Even though the central data structure is the abstract syntax-tree, a main purpose of the editor is to show plain text. This chapter considers this aspect of the editor. First are considered some basic concepts before the actual specification is presented.

When the editor should display the contents of the edited text, it must extract information from the abstract syntax tree. This is done by visiting all nodes in preorder. This process is called *unparsing* [1].

Several decisions must be considered due unparsing:

- How should the text be formatted, e.i. with line breaks and indentions ?
- Should keywords, placeholders and comments occur in different fonts ?
- What about special symbols like $\overrightarrow{m}$ and $\overset{\sim}{\rightarrow}$?
- Should it be possible to do adaptive formatting ?
- What about window widths ?
- What about elision and holophrasting ?
- Should the editor allow the user to change formatting ?

All these questions are answered in this chapter.

Unparsing is one of the major aspects of building an effective syntax-directed editor. An effective unparsing strategy leads to an editor which is more user-friendly with respect to speed, because unparsing should be done nearly all the time.

4.1 Basic unparsing

In the literature and in several syntax-directed editors, unparsing is often done through so-called *templates,unparsing schemes* or *unparsing declarations*, e.i. forms of how to format the text.

[1] the opposite term *parsing* is the process of building the syntax tree [Aho, Sethi & Ullman 86].

Unparsing schemes

An unparsing scheme contains information about how to unparse a syntactic construct. For example could the unparsing scheme for an if-statement look like:

```
"if" @ " then " @
```

The children of the syntactic construct are represented by a @, thus the first @ is the if-statements expression and the second @ is the if-statements statement.

Often, an unparsing scheme does not only contain the keywords, but also *formatting characters*. Formatting characters are like special control codes, that have information about where to break lines, indention, which font to use, etc. As an example, consider the if-then-else-statement:

```
"if" @ " then " "%t%n"  @ "%b%n" "else%t%n" @ "%b"
```

In this example the formatting characters %t, %b and %n are respectively move left margin one indention unit to the right, move left margin one indention unit to the left, and break line and move to the left margin of the next line. When unparsing the above example the output would be[2]:

```
if a > 7 then
  a := 7
else a := 0
```

By the choice of formatting characters, one can control the unparsing more or less flexible to the environment in where the text should be printed.

Fixed versus adaptive formatting

The term *fixed formatting* covers the concept of unflexible unparsing schemes. An unflexible unparsing scheme contains only formatting characters for fixed line breaks and indentions. The term *adaptive formatting* covers the concept of flexible unparsing schemes. A flexible unparsing scheme can contain optional formatting characters like optional line break. An optional line break takes only effect under certain conditions w.r.t. to width of the output device (e.g. a window on a screen).

A user friendly editor should use unflexible unparsing.

Fencing

Although unparsing schemes could be flexible, another problem occurs when unparsing some (most) abstract syntaxes, e.g. the value infix expression $7 * 3 + 2$ could be represented by two different abstract syntax trees (see figure 4.1 below), but when doing unintelligent unparsing

[2] indention unit is here 2 characters

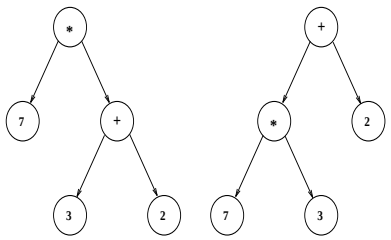

Figure 4.1: Example of two different infix expression syntax trees

those trees shows exact same output e.i. $7 * 3 + 2$. The left tree should have been unparsed to $7 * (3 + 2)$.

The problem is that the syntax-tree carry more information than can be conveyed to the output. Another thing to note is that the unparsing scheme cannot contain the missing information, e.g. the parentheses - at least not in the unparsing schemes discussed presently. If the parentheses should occur in the unparsing scheme, they would always be written, which of course not is the most user friendly way to do it - the screen will quickly be full of unnecessary parentheses.

The problem can be solved by using the concept of *fencing*. Fencing means surrounding of the unparsed node by a new pair of unparsing schemes called *fencing strings*. The fencing strings should only take effect under certain conditions. In the above example of the infix expression the parentheses should only be displayed when the root node was a multiplication expression. This leads to a general solution where operator priority and operator associativity are used to determine when the fencing strings should be displayed.

Another way to solve the problem can be seen in CSG [TeitelBaum, Reps 89], which have a more advanced feature build into the generated editors. CSG uses attribute grammars. Attributes, inherited or synthesized of user defined types, can carry semantic information around in the tree, and they can be referred to in unparsing schemes. In that way unparsing schemes in CSG are very flexible. In the example above CSG would need two attributes, one inherited (the precedence of an operator) and one synthesized attribute (the local level of precedence for an operator)[3]. At a point the two attributes can be compared for deciding which unparsing scheme to use. The technique allows more sophisticated editors with type control and error messages.

Elision & holophrasting

Syntactic elision enables large programs to be cut down in display size by replacing subtrees with elision markers, e.i. symbols indicating that a syntactic structure has been replaced by an abbreviated representation. Elision is in some editors called folding. In these *folding editors* one can mark a block of text and then fold it into a single line of text describing what is hidden.

The term holophrasting[4] is an extended form of meaningful elision with varying degrees of abstraction. The markers are called holophrasts. A holophrasting parameter (see figure 4.2 below) may be used for determining the extent of detail to be represented.

Holophrasting can be perceived as execution of elision on different levels in the syntax-tree.

Elision and holophrasting are typical implemented by allowing several unparsing schemes for a

[3] see pp. 132 in [TeitelBaum, Reps 89] for example
[4] first used by [Hansen 71]

H=1
class ... **end**

H=2
class type ... **value** ... **end**

H=3
class type Numbers = **Int value** f: **Int** → **Int end**

Figure 4.2: Example of holophrasting using holophrasting parameter H.

syntactic construct. One unparsing scheme could show the entire syntactic construct and another could show part of the syntactic construct and/or elision markers. For example, the normal unparsing scheme for a RSL type expression would typical look like:

```
"type " @
```

and for the elided unparsing scheme look like:

```
"type ..."
```

Another way to implement elision is to have information for each node, that controls whether elision is enable or disabled. Whenever enabled the unparsing simply displays a fixed elision mark.

The selection of the different unparsing schemes could be done for the entire edited object (holophrasting) or for individual syntactic construct alone (elision) allowing each user to independently choose the form of formatting.

4.2 Windowing

Even though the unparsing scheme could be very adaptive, they will never cover all extremes, e.i. the text must be shown on a limited size - a window on the screen or on a printer with fixed paper width. This problem can be solved by using windowing.

Windowing refers to the concept of only showing a part of the edited text. The edited text in a window is called a *view*. Several different views could be allowed in different windows.

What characterizes a view ? It reflects a rectangle, e.i. it has a (x,y)-position, a width and a height. This can be captured by the specification in (15.43):

type
  Pos :: x : **Int** ↔ setx y: **Int** ↔ sety,
  View ::
    pos : Pos ↔ setpos
    width : **Nat**
    height : **Nat**

Some terms are used in the following sections:

Virtual window: a window of infinite width and height.

Window: the actual window on screen, where the edited text can be seen.

View: the part of the edited text that can be seen in the window. The entire view can (of course) always fit into the virtual window.

4.3 Unparsing in RSLED

This section describes the specification of how to unparse the abstract syntax-tree in RSLED.

4.3.1 Input/Output

Because unparsing physical should write output to a device, the first step is to specify some functions which can help by doing this.

An IO class is presented which is intended to be an interface between unparsing and the actual user interface.

The class consists of a type definition (14.6) for fonts for several of those constructs which normally are represented in grammars, e.i. keywords and terminal symbols, and a font for special nodes in the abstract syntax-tree e.g. a placeholder (see table 4.1 below). Which physical fonts

Name	Explanation	Example
Normal	Style for normal text.	
Placeholder	Style for placeholders	<:*identifier*:>.
Symbol	Style for symbols	$\overrightarrow{m}$.
Keyword	Style for keywords	**class end**.
Comments	Style for comments.	
Texts	Style for texts	`"Hello out there"`.

Table 4.1: Kinds of font styles.

actually are represented by these styles are of no concern for now. Special fonts for symbols are used because those often are physical defined in a special font file.

To decide which physical device should be used, a type definition is also used (14.14). A file is not further specified.

The class then consists of the following functions, which are not further specified:

width_dev should return the width of the device, e.g. a screen has a limitation of a number of pixels/characters and a file has (often) a limitation of 80 characters at each line.

write_physical which do the physical writing of a text on a device. If all went ok the return value should be true.

select_font should change the font which is used.

get_previous_font should get the previous used font.

get_width which should return the width of a text in respect on the used font, e.i. with one font the text has a width which could be different when using another font.

get_height which should return the height of a font. The height of written text is always the same, even though they, when actually displayed, seems to have different height.

4.3.2 Unparsing schemes

In RSLED each syntactic construct has one (or more) unparsing scheme for each view. Each unparsing scheme could contain formatting characters (see table 4.2 below) which should be compatible with formatting characters in CSG generated editors and special characters representing children (see table 4.3). Several other formatting characters could be defined [GrammaTech 92, pp. 43].

Formating command	Meaning
%t	move the left-margin one indention unit to the right
%b	move the left-margin one indention unit to the left
%n	break the line and return to the current left-margin
%o	optionally, break the line and return to the current left-margin
%c	same as %o, but either all or no %c in a group are taken
%{	beginning of an unparsing group
%}	end of an unparsing group
%[	same as %t%{
%]	same as %}%b
%S(*fontname* :	enter the named font
%S)	revert to the previous font
%%	display a %

Table 4.2: Kinds of formatting commands in RSLED.

This can be modelled by a variant definition (15.8)

At first the specification of an unparsing scheme is considered. At this level of abstraction a unparsing scheme consists of a sequence of unparsing symbols (15.7)[5]. The unparsing symbols abstract away the use of formatting characters. The definition gives reason to define when unparsing schemes are well-formed. This is done informally by telling that the mk_fontbegin and mk_fontend symbols are coming in pairs and they are not nested, e.i. it should be illegal to define

[5]Several more abstract forms can be chosen e.g. one which capture that font change symbols always are coming in pairs. This is chosen because it is close to implementation

Special character	Meaning
@	represents a child
%@	represents a special terminal child

Table 4.3: Kinds of special characters in RSLED.

⟨ mk_fontbegin,mk_fontbegin,mk_fontend,mk_fontend ⟩

The grouping symbols should also occur in pairs. It is legal for them to be nested.

The if-then-else statement from a previous section will in this notation be:

⟨ mk_text(″`if`″),mk_child,mk_text(″ `then` ″),mk_indent,
mk_newline,mk_child,mk_unindent,mk_newline,mk_text(″ `else` ″),
mk_indent,mk_newline,mk_child,mk_unindent ⟩

To capture which unparsing scheme to use the definition of a UnparseType has been defined (15.31). It reflects the possibility to unparse the normal unparsing scheme, the unparsing scheme for separators between list elements and the unparsing schemes for elided constructs. More unparsing schemes are in the future allowed by use of the wild card variant construct.

The unparsing scheme are located in a table modelled by a map (15.34).

When actually doing the unparsing, a function **get_unparse_scheme** is specified (15.37). It returns the appropriate unparse scheme for a node in the abstract syntax-tree.

4.3.3 The unparsing algorithm

The parsing of the unparsing schemes is not an easy task. Because some of the formatting characters only should take effect when the window width is small, the unparsing must know the position and size of each node in the abstract syntax-tree. Therefore each node in the abstract syntax-tree also must have a position and size (a view) associated.

Another problem is that when the user sees a part of the edited text, then then screen may contain output which is derived from unparsing schemes for nodes much further up in the syntax tree, e.g. the user is at the bottom of the edited text and can see and **end**, but cannot see the matching **begin**. This means that there is no obvious way out of unparsing the unparsing scheme containing the matching words, e.g. here **begin** and **end**. In fact a typical situation is that the user is editing at the bottom of the text and can see the final **end**. In RSL specification this means that the entire syntax tree must be unparsed.

What are primarily interested in is the exact size of the node, but at the same time make it so flexible that it can change size if the window view is changed. That means that for some unparsing symbols like optional newline, the unparsing must know if the remainder of the unparsing can fit into the window view. If that is not the case then line break should be inserted, affecting the height of the node.

A solution is to put the remainder of the unparsing into a buffer. To calculate if the buffer can fit into the window we must attach a size (width and height) to the buffer. The buffer solution is not the best one because for large programs many buffers should be created (because group unparsing symbols could be nested), affecting both speed and memory size of the application. At worst case, nearly the complete text should be inserted into a buffer. Furthermore, the buffer content cannot be reused directly when line breaks should be inserted.

So what is the alternative ?

Well the problem is to calculate the size in the remainder of the unparsing. How can this be

done without knowing what is in the remainder ? A solution is to calculate the size of the remainder without storing the rest in a buffer. Instead one could do two (nearly identical) unparsings which could slow down the process. It is also observed that it is not needed to calculate the view size for all of the remainder, just until the next enforced newline.

Unparsing of lists

Since there is only one unparsing scheme for a list node, e.g. the unparsing scheme for a type_def-list is just "@" and not "@,@, ..., @". The unparsing scheme cannot know how many elements exist in the list. The algorithm in RSLED simply reuses the unparsing scheme for all children of a list node.

Unparsing info

When unparsing each node in the syntax-tree some unparsing information (15.44) are needed:

device the current device to which output should be directed,

us the current sequence of unparsing symbols to unparse,

output determine if output should occur, e.i. is true or false. It is used to control whether the remainder of the unparsing scheme just are parsed because its size are calculated.

tree the next tree to unparse,

view the current view of the tree,

margin the current left margin indention,

font the current font,

group the number of nested groups.

To unparse an abstract syntax-tree a function **unparse** has been defined (15.59). As parameters it takes the tree to unparse, the device to which unparsing should be directed and finally a position where to print the tree. The position has meaning both when doing output to screen (the pixel/character coordinates) or to a file (the current line). The definition of **unparse** is simply a call to the **unparse_scheme** function, with a initialization of the unparsing information.

The **unparse_scheme** function (15.105-119) first examines if the tree to unparse is elided and if so, then the unparsing information should be adjusted to contain the unparsing scheme for the appropriate elided tree. Then it iteratively (and recursively) traverses through the unparsing symbols in the unparsing scheme and calls the **unparse_symbol** function.

The **unparse_symbol** function(15.123-229) is defined for each unparsing symbol:

mk_text call a utility function **do_write** which should do the actually outputting and adjust the view appropriately.

mk_fontbegin justs set the right font.

mk_fontend sets the font to the previous font used before mk_font was called.

mk_indent adjusts the left margin.

mk_unindent adjusts the left margin.

mk_newline adjusts the current position to be at left margin. If grouping is in effect and output is false, then the rest of the unparsing scheme does not need to be parsed. A newline when the device is a file, is also physical writing the amount of spaces which equals the left margin. This makes the output more "pretty".

mk_optnewline starts unparsing (recursively) of the remainder of the unparsing scheme. The **output** is set to false. The result is information about the width of the remainder. This is compared to the width of the device and if the device width is greater then a newline is inserted.

mk_groupnewline checks whether grouping is in effect, e.i. **group** number greater than zero. If this is the case then a newline is inserted.

mk_groupbegin calls recursively the **unparse_scheme** function with the remainder of the unparsing scheme and output to false. This results is information about the width of the view for the remainder of the unparsing scheme, which can be compared to the width of the current device. If the remainder of the unparsing scheme could not fit into the device then increase the group counter which instructs mk_groupnewline to insert newlines.

mk_groupend decreases the **group** counter.

mk_indentgroupbegin are a combination of mk_indent and mk_groupbegin.

mk_indentgroupend are a combination of mk_unindent and mk_groupend.

mk_child calls **unparsing_scheme** recursively for unparsing the first child in the tree. Afterwards is examined if the current tree is a list node and in such cases should the unparsing scheme for its parent listseperator be parsed. The next child to unparse are the next sibling - if any.

Another more ambitious unparsing strategy, could be called *incremental unparsing*. Incremental unparsing should only unparse the necessary nodes which is needed to be displayed. That means that a kind of connection between the syntax-tree and the upper left corner of the screen must be adapted. When the view is changed, the syntax-tree should be traversed in different 'directions' to find out when nodes are outside the view. Previous view information about the views for nodes which are 'displayed' above the current view, should be taken under consideration. The traversals should stop when it is seen that the views in the tree no longer need to be changed.

RSLED does not use this strategy, but it would certainly be interesting if it can be adapted.

4.3.4 Scrolling

Another problem is how one controls the view. If the selection moves outside the view (window) then the view should be updated so that the selection again appears inside the view. If the selection is to large to fit into the view, only the upper (left) part of the selection should be inside the view.

The view should not only be scrolled whenever the selection moves outside view, but also when the user wants to see another part of the edited text. The user can choose to scroll left, right, up or down. If the user, after a number of scrolls, selects[6] a new selection, this selection should be displayed in the view.

[6] either by mouse or keyboard

Whenever a unparsing take place the actual output from the unparsing is always done to the virtual screen. What actual can be seen in the window (the view) is controlled by choosing where to place the view in the virtual window[7].

To control the scroll position[8] one must know the exact number of lines of text and the maximum width of a line of text in the virtual window. The scroll bars has a scroll range which for the vertical scroll bar should be from 0 to the number of lines in the virtual buffer. For the horizontal scroll bar the scroll range should be 0 to maximum width of a line in the virtual buffer. Whenever the virtual buffer changes either in number of lines or in maximum width of a line, the scroll ranges also should change. The scroll position is controlled by holding the current line number and the approximately column. The values are available through the current selection - e.i. through the selection's view and through the root's view.

NOTE: A system where the upper left point is (0,0) and that the x-coordinate is increasing to the right and the y-coordinate are increasing down is used. This seems to be a logical orientation of the axes because one reads text from left to right and from top to bottom.

[7] this can be done by a graphic system command (such as SetViewportOrg or SetWindowOrg in Windows) or by a self constructed algorithm.

[8] in Windows terminology, the thumb position

Chapter 5

Parsing & Text input

In the previous chapters we have seen a specification of an editor, which when implemented works fine. In the history of syntax-directed editors the acknowledgement of pure syntax-directed editors is not quite accepted. The reason is that users of pure syntax-directed editors find it to slow, e.g. they must enter expressions in a way very different from normal text editors.

This chapter is dealing with specification of a parsing system, which for certain syntactic constructs (maybe all), can parse a piece of text (edited with a limited text editor) into a piece of abstract-syntax tree. This tree can then be inserted in the abstract-syntax tree.

Another good reason to allow this, is that the editor then could parse text files into abstract-syntax trees affecting the system to be compatible with other editor systems.

5.1 Parsing

Parsing is normally divided into a sequence of two components *lexical analysis*[1] and *syntax analysis*[2]. The lexical analysis scans the input text and split it up into recognizable components, e.g. keywords, identifiers, numbers etc. The syntactical analysis knows the syntactic structure of the underlying grammar, often represent by BNF, and uses this knowledge to parse the components given by the lexical analysis into groupings representing the different syntactic constructs for the underlying language. Often the groupings are represented by a *parse tree*, which, in this case, is identical with the abstract-syntax tree.

Parsing for a syntax-directed editor is not entirely the same as parsing for e.g. a compiler, although the general principles are the same. The main problem is that depending on which part of the text is changed, the parser do not need to parse the entire text, e.g. the user has only changed an expression in a statement. This problem is, in syntax-directed terminology, called *incremental parsing*.

Another problem is if the new text is syntactic wrong. What should happen if this occurs ? One obvious solution is to not allow the user to continue until the edited text is syntactic correct. It should here be stated that syntactic correctness could be maintained, even though the edited

[1] also called scanning
[2] also called parsing

text contain incomplete constructs e.i. placeholders. Placeholders should be a part of the grammar to allow incomplete constructs.

5.2 Incremental parsing

The term *incremental parsing* cover the problems in parsing individual constructs in a grammar e.i. the choice of how to start /end the parser and how to parse incomplete constructs.

One way to solve the problem is to extend the grammar with a special starting symbol and ending symbol, which controls which part of the grammar should be used to parse the text (see sec. 5.2.4).

Incomplete constructs are captured by having special symbols (placeholders) for incomplete structures (see sec. 5.2.5).

5.2.1 Lexical analysis

The lexical analysis is intended to be done with use of a tool called Flex++ [Coëtmeur 93a], [Coëtmeur 93b], which is a compiler program that, given a certain input file, produces a full functional scanner. The Flex++ program supports C++ classes and is based on Flex. The input file consist of regular expression and C++ code (rules).

The scanner is implemented as a *deterministic finite automata* DFA.

5.2.2 Syntax analysis

The syntax analysis is intended to be done with use of a tool called Bison++[Donnelly, Stallman 90], which is a compiler program that, given a certain input file, produces a full general-purpose parser that converts a grammar description for an LALR(1) context-free grammar into a C++ program to parse that grammar. The Bison++ program supports C++ classes and is based on Bison[3]. The input file consists of BNF production rules and C++ code, that should be executed when the corresponding rule is recognized.

The parser is implemented as a *deterministic finite automata* (DFA), with a so-called *parser stack*. Pushing on the stack is called *shifting* and popping is called *reduction*. The stack elements are tokens[4] which is received from the lexical analyzer (here Flex++).

5.2.3 Ambiguity

If the specified grammar in the input file is not a LALR(1) grammar, Bison++ will detect *parsing conflicts*. Such parsing conflicts are usually caused by inconsistencies and ambiguities in the grammar. Two conflicts can arise:

[3] which again is compatible with YACC [Johnson 79].
[4] and attributes/semantic values

Shift/Reduce conflict occurs when it is unclear to shift or to reduce. The well known example of the dangling-else in if-statements are such a conflict. The conflicts can often be eliminated by rewriting the grammar.

Reduce/Reduce conflict occurs when the parser cannot decide to reduce one rule in favor for reducing another rule. These conflicts can be a little tricky to solve - often a large piece of the grammar needs to be rewritten[5]

Conflicts in the RSLED parser should (and can) be completely avoided by rewriting the RSL grammar to take precedence of operators in consideration.

An example of this is the RSL prefix_expr:

```
prefix_expr ::=
  axiom_prefix_expr |
  universal_prefix_expr |
  value_prefix_expr
```

If just typed in as a production rule to Bison, both shift/reduce and reduce/reduce conflicts arises, because the three prefix expressions all contain terminal symbols which have different precedence. The one with lowest precedence (see app. B) is the universal_prefix_expr because it contain the universal state quantification operator $\Box$(see app. A). The other two prefix expression have the highest precedence, because they contain operators (the prefix connective $\sim$ and the prefix operators **abs**,...,**rng**) with the same precedence.

The above rule can then be rewritten as:

```
prefix_expr ::=
  prefix_expr1 |
  prefix_expr14
prefix_expr14 ::=
  universal_prefix_expr
prefix_expr1 ::=
  axiom_prefix_expr |
  value_prefix_expr
```

The number after the prefix_expr refer to precedence level.

When specifying the Bison++ input file all RSL constructs should be checked with respect to precedence, such that ambiguities are avoided.

5.2.4 Start symbol

To control which part of the grammar which should be parsed the parser are started by given it a *start token*. The start token makes the parser algorithm go directly to the desired construct.

As an example lets consider that the user are editing a type definition. When the user has entered the text, which should be parsed as a type definition, the program do some additional

[5] actually Bison++ can produce mysteriously reduce/reduce conflicts. The reason is that the grammar is not a LR(1)/ LALR(1) grammar (see [Donnelly, Stallman 90, pp. 70]).

actions that adds some kind of special characters at the start of the text. These characters are recognized by the lexical analyzer which will return them as a start token to the parser. In the parser definition it would look like :

```
start :
  TYPE_DEF_START type_def
  ;
```

where the non-terminal start is the start production. The terminal symbol TYPE_DEF_START are the start token for type definitions. By adding more start tokens to the parser definition, all allowable constructs can be covered. Furthermore the absence of a start token (XX_START) can start the parser for the complete syntax e.g. :

```
start :
  TYPE_DEF_START type_def |
  ... /* other rules for starting the parser */
  module_decl
  ;
```

The syntactic constructs which should be allowable to edit are represented by nodes in the syntax tree, which are resting places e.i. nodes which corresponds to placeholders for variant productions.

5.2.5 Placeholders

It should be possible to enter incomplete constructs e.g. constructs which have placeholders included. It is very simple to incorporate this in the scanner and parser. The scanner should recognize special groupings if characters, which represent placeholders e.g. <:*placeholder*:> and the parser should include them as possible constructs in the grammar. For an example see below (next section), where a token PH_MODULE_DECL represents the placeholder for a module declaration.

5.2.6 Actions

As stated in the introduction of this chapter the parser should generate a syntax tree, which could be inserted in the edited tree.

To control how the parser should generate the syntax tree Bison++ offers a way to control the semantic value of tokens on the parser stack. If this value[6] is selected to be the type for syntax tree, then new trees/nodes simply can be controlled.

At each rule C++ actions can be placed, and by using the special $x notation, the code can refer to different semantic values in the parser stack. Example:

```
module_decl :
  PH_MODULE_DECL { $$=new ModuleDecl } |
```

[6] the yystype

```
scheme_decl { $$=build(new ModuleDecl,$1) } |
object_decl { $$=build(new ObjectDecl,$1) }
;
```

When for instance a scheme declaration are successfully parsed and recognized its action should be executed. To assign the semantic value to the module declaration the action statement refer to $$[7]. The **build** function should be a function with a variable number of parameters, that builds a new syntax tree with the root as the first parameter and the children as the rest of the parameters. The $1 notation refers to the semantic value (the syntax tree) for a scheme declaration.

A special case arises for list constructs. Example:

```
module_decl_string :
  module_decl { $$=build(new ModuleDeclString,$1) } |
  module_decl_string module_decl { $1->insert_tree_last($2); $$=$1 }
  ;
```

In the sample above one rule is left recursive. That means that when the first module declaration in a list is recognized the action for the first rule are executed and the tree for a module_declaration_string is builded. When successive module declarations are recognized they (the semantic values) are inserted in the list with the function **insert_tree_last** (7.10). This method is necessary due to the use of flat lists. If binary lists were used the **build** function could have been used.

5.3 Text editor

To allow the user to edit some piece of text, as in a normal text editor, a small editor must be developed, which can do some of the basics of text editors. It should be possible to move around in the edited text with cursor keys (or with the mouse), delete and insert characters.

The editor consist of a buffer, which contain the edited text as a sequence of characters. A variable contain information about where the cursor is in the buffer

The constructs which should be allowed to textual edit are (all) those which can be selected e.i. resting places which includes placeholders.

One of the mayor problems connected to the text editor is how it is presented to the user. Should it be placed at the same spot as the selection or should be placed at a total separate place e.g. a new window. The most user-friendly is to place it at the selection. This would also make it compatible with CSG generated editors.

[7] these are, by Bison++, translated to a array variable, which represents the parser stack. For a complete description of the notation see the Bison manual [Donnelly, Stallman 90].

Part II

Implementation

Chapter 6

Implementation

One of the main objectives has been to develop a running version of RSLED. The following chapters describe the implementation.

Since the program has to be developed in a period of time of 6 months, and at least 1/2 of these has been used to background studies and specification, there has only been little time to revise specifications and/or program design. The implementation was first concentrated on the basic tree manipulation and movement operations (chapter 7). And then, with decreasing priority: unparsing (in sec. 7.4), user interface (chapter 10), parsing and text input.

Little time have been spent on making the implementation real effective with respect to both space and time. Operations which are critical to time efficiencies have been programmed (more or less effective) and it is only noticed that they are time critical. Future work should improve these.

RSLED have been implemented in the C++ programming language using the C++ compiler from Symantec Corporation (see appendix D).

6.1 Modularity

RSLED is naturally divided into several logical units, which take care of different parts of the program. The unit structure is not something which was planned thorough. They just seemed natural to divide them into parts which communicate as little as possible and into parts which has common functionality.

The basic functionality of the program is very much independent of the user interface. Below is a short description of the five units and their source files (see also figure 6.1 for an overview). The next chapters discuss them in more detail.

Trees This part consists of tree units which implements a basic syntactic tree node, a generic n-ary tree and a complete syntactic abstract syntax tree node.

- **synnode.h/synnode.cpp** specify and implement basic syntactic tree nodes.
- **narytree.h/narytree.cpp** specify and implement generic n-ary trees.
- **syntree.h/syntree.cpp** specify and implement a general abstract syntactic tree node.

Userinterface This part takes care of the user interface. It consists of four units, each one taking care of different windows representing the user interface to the user.

- **rsledwin.h** specify the uniform interface between the four units (windows).
- **rsledwin.cpp** implements the main window. Takes care of initializing Windows. Creates the three sub windows. React on menu commands.
- **editwnd.cpp** implements the edit window.
- **poswnd.cpp** implements the position window.
- **transwnd.cpp** implements the transformation window.

To this part also the resource file containing the menu, dialog, icons etc. are considered. The filenames are:

- **rslmenu.h** file containing defines for menu command identifiers.
- **rsledwin.res** resource file containing dialogs, menu definition and the RSLED icon.
- **commdlg32.lib** a library containing common dialogs for file I/O e.i. file open, save as and save dialogs. The file is distributed along with Symantec C++.

IO This part consists of 3 units which works as interface between the Userinterface unit and the Tree unit.

- **io.h/io.cpp** specify and implements general output routines for text, selection of fonts etc. General file I/O routines.
- **mystring.h/mystring.cpp** specify and implements an interface between Windows string-functions and string-functions known from standard C libraries.
- **def.h** specify definitions of global constants and global defines, which is used throughout the entire program.

Language This part consists of modules which is language dependent, e.i. definitions of nodes for RSL, unparsing schemes, scanning and parsing modules.

- **nodeids.h** defines uniquely all identifiers for each node in the abstract-syntax tree.
- **nodeinfo.cpp** defines unparsing schemes and legal transformations.
- **rsl.h** declares all nodes for RSL.
- **rsl.cpp** implements how subtrees are initial constructed.
- **rsl1.cpp** implements the constructors for nodes.
- **rsl2.cpp** implements how subtrees are copied (cloned).
- **rsl3.cpp** also implements how subtrees are initial constructed.
- **scanner.l/scanner.h/scanner.cpp** specify and implement the lexical scanner for RSL.
- **parser.y/parser.h/parser.cpp** specify and implement the parser for RSL.

Other This part consists of different standard modules which are used throughout the program.

- **memory.h/memory.cpp** specify memory routines.
- **stdlib.h/stdlib.cpp** is the C standard library.
- **string.h/string.cpp** is the C standard string library.
- **windows.h** is the general standard Windows library.

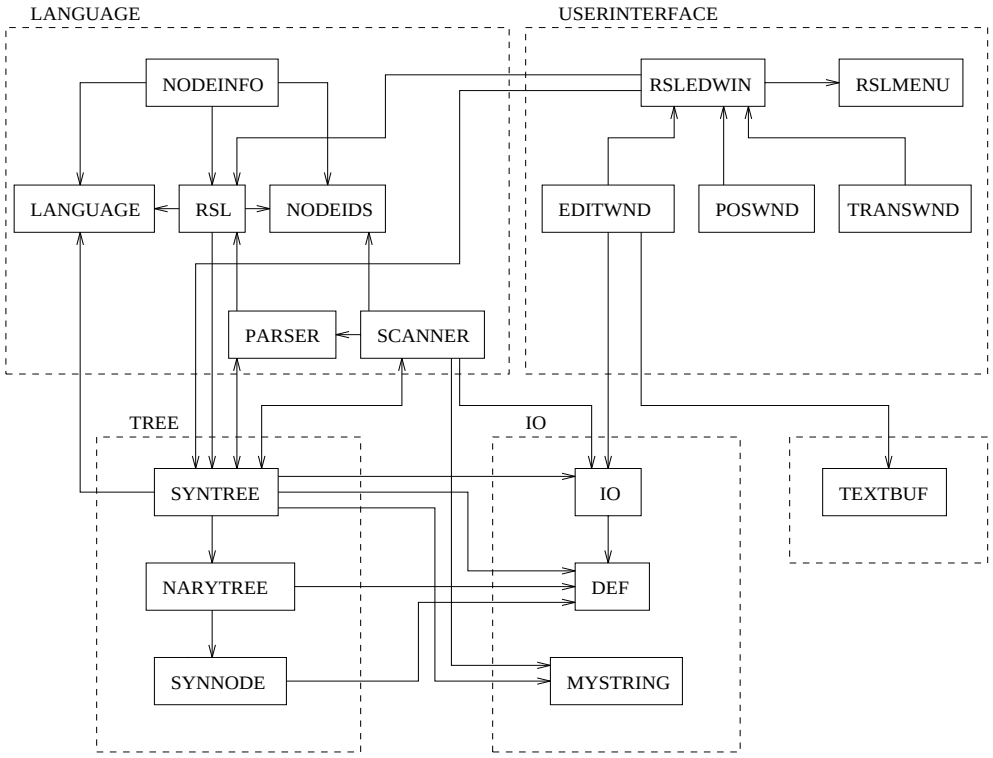

Figure 6.1: Overview of units and their dependency.

Chapter 7

Tree

This chapter describes the most important data structure in the entire editor - the syntax tree.

As discussed in section 2.2 a tree consists of nodes which is connected to subtrees which itself is nodes etc.

Implementation of trees should be:

- efficient with respect to memory consumption, e.i. each node should only contain enough information as necessary,
- efficient with respect to speed of movement operations in the tree,
- and the code should be easily to extend and maintain.

A syntax tree can of course be of any size, as the user could extend the tree indefinitely. This means that a dynamical allocation of the nodes is necessary and as such it is convenient to use pointers.

How should the pointers be connected to each other to model the tree ? Several proposals have been discussed both in [Hurvig 85] and in [Ekner, Hørlyck 87]. In this implementation of syntax trees the trees are conceived as nodes with pointers to the nodes parent, first child, last child, previous (left) sibling and to next (right) sibling as can be seen below in figure 7.1.

This structure favors speed of movement instead of memory consumption. It has been chosen partly because it is very important that the users sees a effective an quick editor and partly because it is easy to implement, e.i. there is no special routines for finding parents or left siblings[1].

The third item which was very important was the possibility to extend and maintain the tree. Due to this an object oriented structure has been chosen to implement a tree.

[1] In some implementations this is necessary

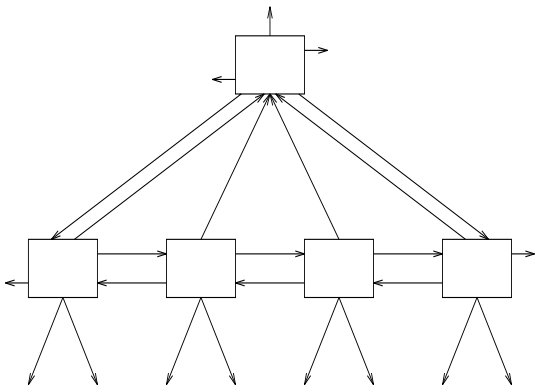

Figure 7.1: Pointer structure in implemented tree.

7.1 Object Oriented Design

One of the most essential questions regarding implementation is how the class structure should be designed. One way of design can be seen in figure 7.2 below. In this figure a general base NODE is at the top. A class NARYTREE should implement general n-ary trees[2]. From each base node a general syntactic node SYNNODE inherits basic properties. A syntactic node should contain basic properties for nodes in a syntax tree. A syntax tree node SYNTREE inherits (multiple) from both a general n-ary tree and from a general basic syntax node since a syntax tree here is perceived as an encapsulated node and tree.

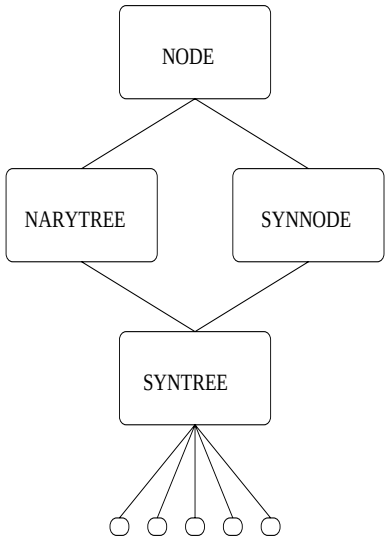

Figure 7.2: One class design.

In RSLED the class design is quite straightforward. A class models the basic concepts of a syntactic node SYNNODE. A class models the basic concepts of a n-ary tree NARYTREE and finally a class models the basic concepts of syntax tree SYNTREE. How these inherits from each other can be seen in figure 7.3.

Another idea, which first at a later stage was recognized, was that the class representing a syntactic node in the grammar, could be subclassed into classes representing different productions

[2]trees with nodes that could have more than 2 subtrees.

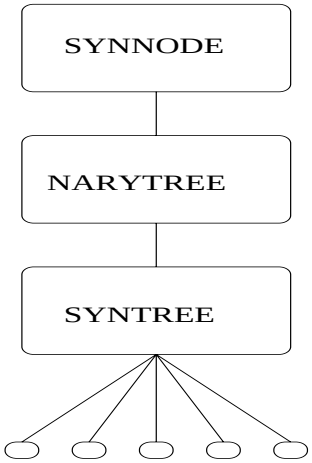

Figure 7.3: Class design in RSLED.

in a grammar. The idea is illustrated in figure 7.4 below. The classes represents different nodes in

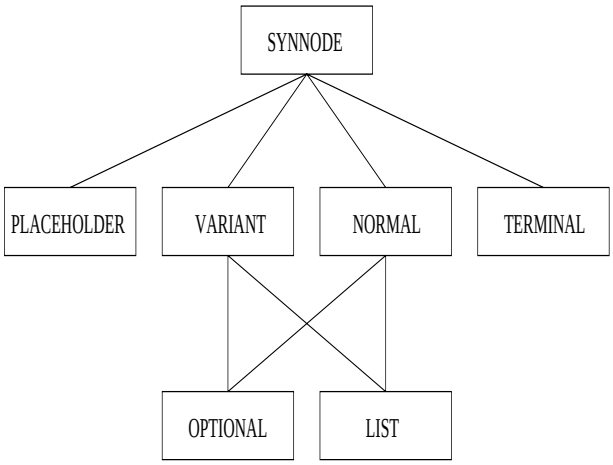

Figure 7.4: Subclassing of SYNNODE.

the syntax tree, e.g. there is a class representing placeholder nodes, variant production nodes etc. Some of the nodes can be subclassed into list nodes or optional nodes. In fact the idea is nearly implemented, but in RSLED it is not done by subclassing, but with use of macros (see sec. 8.3). In the future subclassing could be implemented.

7.2 SYNNODE

The SynNode class captures the node information in the tree (see figure 7.5).

```
// Prototype for class
class SyntaxTreeNode;

typedef SyntaxTreeNode* PSYNNODE;

// Class Specification
```

```
class SyntaxTreeNode {
  NODEID        nodeid;
  BOOL          resting_place;
  BOOL          listnode;
  BOOL          optional;
  BOOL          selected;
  unsigned char minfixed;
public:
  VIEW view;

  SyntaxTreeNode();

  void set_nodeid(NODEID nid) { nodeid=nid; }
  void set_resting_place(BOOL rp) { resting_place=rp; }
  void set_listnode(BOOL ln) { listnode=ln; }
  void set_optional(BOOL op) { optional=op; }
  void set_minfixed(const unsigned char mf) { minfixed=mf; }

  BOOL is_selected(void) const { return selected; }
  BOOL is_resting_place(void) const { return resting_place; }
  BOOL is_listnode(void) const { return listnode; }
  BOOL is_optional(void) const { return optional; }

  unsigned char get_minfixed(void) const { return minfixed; }

  void select(void) { selected=TRUE; }
  void de_select(void) { selected=FALSE; }

  void set_view(const int,const int,const unsigned int,const unsigned int);
  VIEW get_view(void) const;
  BOOL isinview(const int,const int);
  BOOL view_overlaps_view(VIEW);
  BOOL view_isin_view(VIEW);

  NODEID get_nodeid(void) const { return nodeid; }

  virtual BOOL compare(const PSYNNODE,const PSYNNODE) const;
};
```

Figure 7.5: List of SyntaxTreeNode specification (SYNNODE.H)

The private attributes are those given from specification (4.0-4.13)[3], e.i. the node information. Functions to set and retrieve the state of the information are specified[4]. Furthermore, functions to examine views are defined.

The most important function is the virtual function **compare** which is intended to compare two nodes and return true, whenever they contain the exact same information. The function is intended to be overwritten for special nodes with extra node information, e.g. terminal nodes having a string containing a identifier.

7.3 NARYTREE

The NARYTREE unit specifies and implements "general" n-ary trees (see figure 7.6). As it inherits information from SyntaxTreeNode, it is not completely general, e.i. works for all nodes. This could be considered a disadvantage of the class design.

[3] I would have liked that some of the attributes was constants as they cannot be redefined when first initialized.
[4] and in fact also implemented through inline functions definitions.

```
// Class prototype
class NaryTree;

typedef NaryTree* PNARYTREE;

// Class Specification
class NaryTree: public SyntaxTreeNode {
  PNARYTREE parent;
  PNARYTREE firstchild;
  PNARYTREE lastchild;                                                    10
  PNARYTREE prevsibling;
  PNARYTREE nextsibling;
public:
  NaryTree();

  void insert_tree_first(PNARYTREE);
  void insert_tree_last(PNARYTREE);
  void insert_tree_before(PNARYTREE);
  void insert_tree_after(PNARYTREE);
                                                                          20
  void delete_subtree(PNARYTREE);
  void delete_treenode(PNARYTREE);

  void cut_subtree(void);
  void cut_siblings(void);
  void paste_subtree(const PNARYTREE);
  void null(void) {
    parent=firstchild=lastchild=prevsibling=
    nextsibling=NULL;
  }                                                                       30

  BOOL compare_trees(PNARYTREE,PNARYTREE) const;
  virtual BOOL compare_nodes(PNARYTREE,PNARYTREE) const;

  PNARYTREE get_parent(void) const { return parent; }
  PNARYTREE get_firstchild(void) const { return firstchild; }
  PNARYTREE get_lastchild(void) const { return lastchild; }
  PNARYTREE get_prevsibling(void) const { return prevsibling; }
  PNARYTREE get_nextsibling(void) const { return nextsibling; }
                                                                          40
  unsigned int count_childs(PNARYTREE) const;

};
```

Figure 7.6: List of NaryTree specification (NARYTREE.H)

The class consists of the five attributes, which are five pointers to other trees, and of functions to manipulate the tree. These functions are all specified in scheme TREE2 (pp. 3.1). Functions to retrieve the pointers are also defined **get_XX**.

Four **insert_tree_XX** functions inserts subtrees at different places: as first child or last child, before the tree, e.i. left sibling and after the tree, e.i. right sibling.

Two functions **delete_XX** delete respectively a complete subtree and a node.

The **cut_subtree** simply removes the 'connections' to the subtrees, e.i. first child and last child.

The **paste_subtree** inserts a tree as a subtree.

Two functions **compare_XXX** takes care of comparison of trees.

A finally a function **count_children** counts the number of children of a node.

7.4 SYNTREE

The SyntaxTree class contains primarily functions to move around in the tree and the unparsing function (see figure 7.7).

```
// Class prototype
class SyntaxTree;

typedef SyntaxTree* PSYNTAXTREE;

// Class specification
class SyntaxTree: public NaryTree {
  static PSYNTAXTREE clip_buffer;
  static BOOL        mult_clipped; // is multiple listnodes in clipbuffer
public:                                                                    10
  SyntaxTree();

  void insert(PSYNTAXTREE,PSYNTAXTREE = NULL,PSYNTAXTREE = NULL,
              PSYNTAXTREE = NULL,PSYNTAXTREE = NULL);

  PSYNTAXTREE ascend_parent(BOOL&);
  PSYNTAXTREE forward_preorder(BOOL&);
  PSYNTAXTREE forward_preorder_with_optionals(BOOL&);
  PSYNTAXTREE forward_sibling(BOOL&);
  PSYNTAXTREE forward_sibling_with_optionals(BOOL&);                       20
  PSYNTAXTREE backward_preorder(BOOL&);
  PSYNTAXTREE backward_preorder_with_optionals(BOOL&);
  PSYNTAXTREE backward_sibling(BOOL&);
  PSYNTAXTREE backward_sibling_with_optionals(BOOL&);

  void de_select(const int,const int,BOOL&);
  void de_select_mult(BOOL&);

  PSYNTAXTREE belongs_to(const int,const int);
                                                                           30
  void unparse(OUTPUT,FILEHANDLE);
  BOOL parse_file(NODEID,FILEHANDLE);

  virtual PSYNTAXTREE make_child(void);

  void cut_to_clipped(BOOL,BOOL);
  BOOL paste_from_clipped(BOOL&);

private:
  PSYNTAXTREE forward_preorder(const BOOL,const BOOL,BOOL&);               40
  PSYNTAXTREE backward_preorder(const BOOL,const BOOL,BOOL&);

  void insert_optional(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_first(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_before(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_after(const PSYNTAXTREE,const BOOL,BOOL&);
  void delete_optional(const PSYNTAXTREE,BOOL&);
  void delete_listnode(const PSYNTAXTREE,BOOL&);

  SYMBOL nextsymbol(const char *,unsigned int &);                          50
  void unparse_symbol_to_screen(const char *,PSYNTAXTREE,int&,int&,BOOL,
                                unsigned int,unsigned int);
  BOOL unparse_symbol_to_file(FILEHANDLE,const char *,PSYNTAXTREE,int&,int&,BOOL,
```

```
                                    unsigned int,unsigned int);
  void unparse_symbol_to_buffer(const char *,PSYNTAXTREE,int&,int&,BOOL,
                                    unsigned int,unsigned int);

  PSYNTAXTREE copy_tree(void);
  virtual PSYNTAXTREE clone(void) const;
};                                                                          60
```

Figure 7.7: List of SyntaxTree specification (SYNTREE.H)

The SyntaxTree class consists of two attributes - one containing the clip buffer and one boolean value that contains information whether multiple nodes are part of the clip buffer.

7.4.1 Movement

The nine movement functions are implemented very similar to the specification (see scheme TREE6, pp. 3.2.1). They all have a reference parameter, which is a boolean value that is set to true, whenever a movement has changed the syntax tree, e.i. either a node is inserted or deleted.

The traversal functions, which was specified in scheme TREE5 (pp. 3.2.1), is implemented as private member functions **forward_preorder** and **backward_preorder**. They use the utility functions **insert_XX** and **delete_XX**, which were specified in scheme TREE3 (pp. 3.2.1) and TREE4 (pp. 3.2.1).

7.4.2 Un-selection

Whenever a node should be un-selected, because the user has changed the selection, two functions is available **de_select** and **de_select_mult**. An un-selection of a node can imply deletion of some nodes, because the node which should be un-selected could be an optional node. To find out if any nodes should be deleted the **de_select** function has three parameters: two which represent a (x,y) coordinate and one which is a boolean reference. The coordinate should represent the mouse position of the *new* selection. This is used to examine if the new selection is part of (a subtree of) the old one - see **belongs_to** function below. The boolean reference parameter is set to true if any nodes are deleted.

The **belongs_to** function finds a certain node. The parameters should represent a (x,y)-coordinate. By traversing through the syntax tree the function finds that node which has the view, that captures the coordinate. It always tries to find the smallest view first. The function is used when the user makes a new selection with the mouse.

7.4.3 Unparsing

When the system should unparse the tree it calls **unparse** with information about which device to unparse to. If screen is chosen the private function **unparse_symbol_to_screen** are called. If file is chosen the private function **unparse_symbol_to_file** are called. They have a functionality very similar to the specification of unparsing of symbols (15.37-15.233). The recursive function **unparse_scheme** (15.115) is enclosed into the **unparse_symbol_to_XX** function, as a simple

while-loop. The parsing of the unparsing schemes are handled by a private function **nextsymbol**, which given the current position in the unparsing scheme and the unparsing scheme, returns the next unparsing symbol. If the unparsing symbol represent simple text, a global variable **symbtxt** contains the text.

In the specification some unparsing information was used. This is implemented through different parameters to the **unparse_symbol_to_XX** function.

A special function **select_symbol** (see syntree.cpp in G.13), writes special RSL symbols in either ASCII or special graphic symbols.

The **unparse_special** function is one which only should be overwritten for special terminal nodes, e.i. nodes with carry extra information, which when unparsed should return the unparsing of the extra information.

7.4.4 Parsing

The SyntaxTree class also have a member function **parse_file**. As parameter it has a file handle defining the file to parse.

In the implementation file (syntree.cpp) the function creates an instance of a parser and calls the actual parser function **yyparse**, defined by Bison++ (see sec. 8.4 and sec. 8.5):

```
// Initialize parser with file
MyParser myparser(file);
// Call parser
return myparser.yyparse();
```

The parser is implemented as a class MyParser, which inherits from the PARSER class generated by Bison++. Inside the MyParser class a instance of a MyScanner class are declared. MyScanner inherits from the SCANNER class defined by Flex++. An overview of the class structure can be seen in figure 7.8.

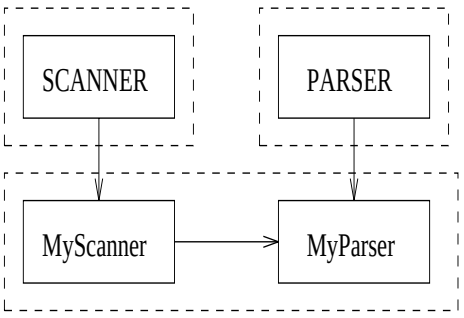

Figure 7.8: Scanner/Paser class structure.

7.4.5 Virtual member functions

The SyntaxTree class declares three virtual functions. These functions may all be overwritten by subsequent classes, which inherits from SyntaxTree.

The **make_subtree** function is a function that creates the right subtree for the node. It is intended that each node in the syntax tree, e.i. each syntactic construct in the language, should overwrite this function. Each node knows exactly which subtrees to create (it is defined in the grammar of the language).

The function **unparse_special** is explained in sec. 7.4.3.

The function **make_childs** is intended to be overwritten by subsequent classes, which inherits from SyntaxTree. It should return a new instance of classes representing optional and list constructs.

7.4.6 Cut & Paste

The private **copy_tree** function is used to implement cutting and pasting e.i. used by the function **cut_to_clipped** and **paste_from_clipped**.

The functions must consider whether multiple nodes are selected. This makes the situation a bit harder. If multiple nodes not are selected the tree representing the selection simply are copied, by using **copy_tree**, into the **clip_buffer** (see figure 7.9). The selection should change to be a

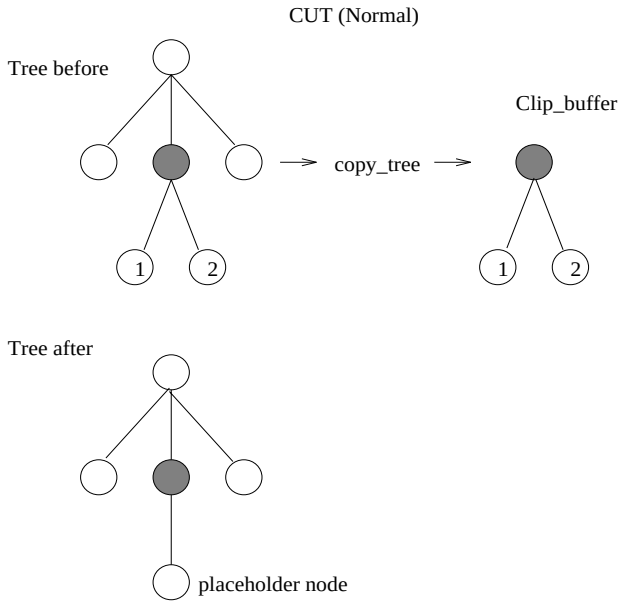

Figure 7.9: Cutting of tree, with only one selection.

placeholder. This is handled by removing the connections (pointers) to the subtree, and then call **make_subtree**, which automatically creates the new subtree representing the placeholder. If multiple node were selected, then they are part of a list, and the hole list is first copied. Then are the un-selected node(s) removed and the copied tree are inserted into the **clip_buffer** (see figure 7.10).

Pasting is almost opposite of cutting. It is first checked if it is legal to insert the clipbuffer at the selection - the node identificators are compared - and then the selection, which could be hole subtree, are deleted. Next are the clip buffer copied and pasted into the the syntax tree.

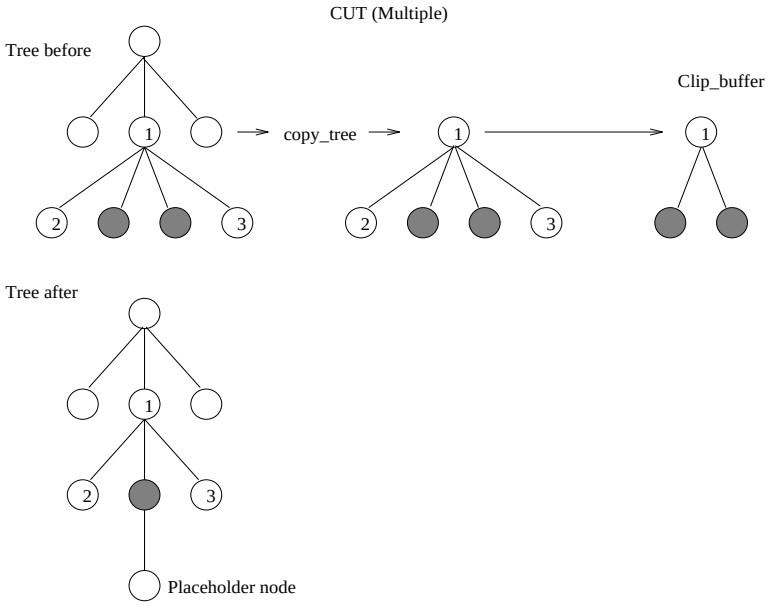

Figure 7.10: Cutting of tree, with multiple selections.

Chapter 8

Language

This chapter describes the language dependent part of RSLED, e.i. the class definitions of each syntactic construct in the language, the unparsing schemes, the transformation function, the scanner and the parser.

8.1 NODEINFO

The file contains two important tables and one function which carry information of each node.

A large array **NodeInfoTable**[1] contains information about each nodes syntactic name, unparsing scheme for the node and the unparsing scheme for separators. Several entries in the array are just the NULL pointer since those values never are used. For example would the entry for a scheme declaration looks like:

```
{ "Scheme_Decl", "%(keyword%:scheme%)%t%n@%b",NULL }
```

where the first field is the syntactic name (are used to displayed in the position window (see sec. 10.2.3). The second field is the unparsing scheme and the third is the unparsing scheme for a separator. As a scheme declaration is not a list node, this is just NULL.

A large table **TransformationTable**[2] defines all the legal transformations. For each constructs which can be transformed, a number of names is defined. The names correspond to those constructs the node can be transformed into. The names (which are strings) are primarily used to display the right buttons in the transformation window (see sec. 10.2.4). To actually transform the node, a function called **MakeIt** are defined. Given a number (which corresponds to a button value) a new instance of the right syntactic construct are returned. This can then be inserted into the syntax tree at appropriate place.

[1] the table are declared in LANGUAGE.H
[2] the table are declared in LANGUAGE.H

8.2 NODEIDS

This file simply defines a unique node identifier to each operator in the syntax tree. An example is the node identifier for the scheme declaration:

```
#define NID_SCHEMEDECLARATION       13
```

Also node identificators for placeholders are defined. Their names end with NULL e.g.

```
#define NID_MODULEDECLNULL          9
```

The values are used in the constructor of each class representing a syntactic node (see next section).

8.3 RSL files

The RSL.H file defines over 300 classes each representing a syntactic production in the grammar. All classes inherit from the SyntaxTree class (sec. 7.4) as they should be allowable nodes in the syntax tree.

Each node in the syntax tree is represented by its own class, which inherits from SyntaxTreeNode. A typical example is the class for a module declaration:

```
// Class specification

class ModuleDecl: public SyntaxTree {
public:
  ModuleDecl();
  void make_subtree(void);
  PSYNTAXTREE clone(void) const;
};

// Class implementation                                                    10

ModuleDecl::ModuleDecl() {
  set_nodeid(NID_MODULEDECL);
  make_subtree();  // Expand initially the tree with approriate children
}

void ModuleDecl::make_subtree(void) {
  insert_tree_last(new ModuleDeclNull);  // ModuleDeclNull represents the
                                         // placeholder for a module
                                         // declaration                    20
}

PSYNTAXTREE ModuleDecl::clone(void) const {
  return new ModuleDecl (* this);  // Clone tree and initialize attributes
                                   // to the same as what 'this' points at
```

Special macros defines different properties of a syntactic node. The typical declarations for the class are (see also the class definition above):

Constructor All constructors of the classes is initialized with the right node identificator (NID_XX). The default values for the node information set to (by the constructor for SyntaxTreeNode (in SYNNODE.CPP)): selected=false, listnode=false, optional=false, resting_place=true, nodeid=0, elided=0 and view members are initialized to zero. In this constructor these values can be changed, e.g. is the node an optional, a list or a resting place.

Make_subtree member function, which should be defined to extend the node with appropriate chidren (see also 3.1).

clone member function, which clones (make an exact copy) of the node.

The implementation of the different member functions in the class is located in different files. The **make_subtree** functions is implemented in **rsl.cpp** and **rsl3.cpp**, the constructors is implemented in **rsl1.cpp**, the **clone** functions is implemented in **rsl2.cpp**.

In table 8.1 there is an overview of how the macros differ with respect to initializing the different attributes in the node. It is also possible to see whether **make_subtree** function is called under construction and whether **make_child** is defined. The initial value, which is set by the constructor in SyntaxTreeNode are displayed below the different attribute names.

Macroname	listnode	minfixed	optional	resting place	make_subtree called	make_child defined
	false	?	false	true	no	no
ChoiceCl	true	1		false	yes	yes
Choice2Cl	true	2		false	yes	yes
Class				false	yes	
ClassRest					yes	
LexTerminal				false		
ListCl	true	1		false	yes	yes
ListClRest	true	1			yes	yes
List2Cl	true	2		false	yes	yes
NullCl				false		
Optional			true	false		yes
OptList	true		true	false	yes	yes
OptString	true		true	false		yes
Product2Cl	true	2		false	yes	yes
StringCl	true	1		false	yes	yes
Terminal				false		
VarCl					yes	

Table 8.1: Table of difference in macro constructors.

Example: Class Macro

A normal node in the syntax tree is defined by that it is a resting place. An example would be a scheme declaration:

```
scheme_decl ::=
  SCHEME scheme_def-list
```

which would have the class constructor:

```
SchemeDecl::SchemeDecl() {
  set_nodeid(NID_SCHEMEDECL); set_resting_place(FALSE); make_subtree();
}

void SchemeDecl::make_subtre(void) {
  insert_tree_last(new SchemeDefList);
}
```

The NullCl macro corresponds to placeholder nodes. The OptXX macros corresponds to optional nodes. The VarCl macro corresponds to variant productions. It automatically calls the macro for terminalnodes (the Null Macro), as a variant node have a placeholder attached. The ListCl,ListClRest,StringCl and ChoiceCl macros all initialize the list node to true and sets minfixed to one. They also declares the **make_child** function. The List2Cl,Choice2Cl and Product2 macros all initialize the list node to true and sets minfixed to two. They also declare the **make_child** function.

LexTerminal macro

The LexTerminal macro are used to defined nodes which represent special terminal nodes with extra information. The class definition is:

```
class Identifier: public SyntaxTree {
private:
  char str[IDLENGTH];
public:
  Identifier();
  Identifier(char *);
  PSYNTAXTREE clone(void) const;
  virtual char *unparse_special(void) const;
  virtual BOOL compare(const PSYNNODE,const PSYNNODE) const;
}
```

As could be seen it have a special attribute **str**, which should contain additional information e.g. an identifier or a value. It also have an extra constructor member function, which could initialize the **str** attribute. Furthermore two virtual functions is defined:

- **unparse_special** which is called when an unparsing scheme contains the special unparsing symbol %@. The function should return the **str** attribute.
- **compare** which overwrites the normal compare function. It always return false, because no two lexical terminal nodes could be equal[3].

8.4 SCANNER

The scanner - the lexical analyzer provides the parser (see next section) with lexical tokens. The scanner analyzes character input from a file and divides the characters into tokens after special

[3] the function must be redefined, if the system is extended with an attribute grammar.

rules defined by the programmer. To define the scanner Flex++ was previous discussed (sec. 5.2.1).

Flex++ defines a scanner class with attributes and member functions.

The input file to Flex++ consists of a four parts: declarations, definitions rules and user code.

To actually run Flex++ see appendix E.

Declarations

These are special definitions which can change the property of the scanner class. In RSLED the scanner class are defined to be the name SCANNER. It is stated that a class is wanted and that a new public attribute theLine is wanted. The variable should contain the current line of the input to the scanner. Further it is stated that two member functions should be pure, e.i. the SCANNER class cannot be used before another class inherits SCANNER and overwrites these two functions. All in all the definitions makes the class to be declared as:

```
class SCANNER {
  // private attributes and member functions
public:
  int theLine;
  SCANNER () : theLine(1);
  virtual int yy_input(char *,int &,int) = 0;
  virtual void yy_fatal_error(char *) = 0;

  // other public member functions and attributes
}
```

Furthermore the declarations consist of some source code which declares some functions which are defined in the user code part.

Definitions

This part consists of definitions which simplifies the scanner specification. The definitions are defined[4] as a

name definition

where the name then can be referred to in the rules section. An example would be the definition of an ascii_letter which is:

ascii_letter [a-zA-Z]

The section ends with two declaration of two start condition symbols - **comment** and **error**. These are used as a mechanism for conditionally activating rules.

Most of the definitions is derived from the RSL syntax in appendix A.3.

[4] for a description of these see [Paxson 90, pp. 3].

Rules

The rule section consists of rule definitions of the type:

pattern action

where pattern is regular expressions and action are C++ source code.

The section starts with rules for recognizing comments. A special boolean variable **incomment** is set to true whenever the scanner is scanning a comment. As long as the variable is true, the scanner is conditionally started to recognize comment rules:

```
"/*"                          { incomment=TRUE; return PARSER::COMMENTSTART; }
"*/"                          { incomment=FALSE; return PARSER::COMMENTEND; }
<comment>"*/"                 { incomment=FALSE; return PARSER::COMMENTEND; }

<comment>{comment_char}+      return PARSER::COMMENT_TEXT;
<comment>\n                   return PARSER::COMMENT_NEWLINE;
```

Next are four rules concerning recognition of (semantic) errors in the input file e.i. {! ...!} constructs. This feature is implemented, so that RSLED is compatible with the *eden* editor:

```
"{!"                          BEGIN(error);
<error>[^!]*                  /* Eat anything that's not a '!' */
<error>"!"+[^!}]*             /* Eat up '!'s not followed by '}' */
<error>"!"+"}"                BEGIN(INITIAL);
```

Next follows rules for recognizing identifiers, keywords, placeholders, literals, start symbols and a hole range of rules recognizing the different RSL special terminal symbols:

```
{letter}{ldup}*               return id_or_keyword((char *)yytext);

"<:"{letter}{ldup}*":>"       { int i=placeholder((char *)yytext);
                                if(i==-1) yyterminate;
                                return i;
                              };

{digit}+                      return PARSER::INT_LITERAL;
```

⋮

```
"*"                           return PARSER::AST;
```

⋮

The rules section ends with definitions for recognizing spaces, tabs, newlines and end of file:

```
[ \t]                          ;
\n                             theLine++;
.                              return PARSER::FEOF;
<<EOF>>                        return PARSER::FEOF;
```

User code

The user code part consists of C++ source code. Two tables are declared: one is used for recognizing placeholders **PTable** and one is used for recognizing keywords **KTable**. Two functions search through the tables in a binary search fashion. That means the tables must be ordered.

A function **startsymbol** recognizes the right start symbols which then are returned to the parser.

8.5 PARSER

The parser input file consists of four parts: declarations, bison declarations, grammar rules and additional C++ code.

Declarations

Like the declarations for Flex++ the definitions for Bison++ define how the parser class should look like. With the definitions defined the class will look like:

```
class PARSER {
  // static const declaration of tokens
public:
  PARSER ();
  virtual void yyerror(char *) = 0;
  virtual int yylex() = 0;

  // other public member functions
};
```

10

The two pure functions (those with = 0), state that the class is an abstract class and cannot be used, before another class has inherited it and overwritten the two functions.

The section ends with some C++ code which states that the parser should use an extern defined variable **EditTree** of the type PSYNTAXTREE. This variable contains, when the parser successfully has parsed input, the abstract-syntax tree of the parsed text. Furthermore, a function **build** are declared.

Bison declarations

This section declares all those terminal and non terminal symbols being used by the grammar rules for the RSL language.

It is divided into five parts: tokens for start symbols XX_START, tokens for placeholders PH_XX, tokens for keyword KW_XX, tokens for RSL symbols and a part defining tokens for special terminal symbols like identifiers, literals and comments. Furthermore, a token defining file end of file (FEOF) is declared.

Grammar rules

This part defines the RSL language by a BNF kind of style[5]. This part is the largest and for some examples see sec. 5.2.6 or the source code in H.2.

Additional C++ code

This section only defines the **build** function which simply insert children into a syntax tree, by calling **insert_tree_last**.

[5] see [Donnelly, Stallman 90].

Chapter 9

IO

This chapter describes the interface between the functional part of RSLED and the user interface in RSLED. The main objective have been to make it easy to convert to other platforms.

9.1 IO

The IO unit (see appendix F.2), consists mainly of lower level functions to handle the screen and functions to read/write from/to files. They are implemented partly through several Win32s Windows functions and partly through the standard C library.

Before any of the screen functions in the unit can be used, a call to the **set_output** function must have been done. The parameter is a handle to a device context (HDC), which determine the window that actually should retrieve output.

Colors

RSLED uses only two colors, beyond the Windows default ones: BLACK and WHITE. In future implementations RSLED could be extended with the possibility of user-defined colors for keywords, placeholders etc.

Three functions are declared for use with colors:

- **set_bk_color** sets the background color. Is mainly used to reverse the video when displaying the selection.
- **set_text_color** sets the text (foreground) color.
- **get_text_color** retrieves the current text color.

Fonts

To use different fonts in RSLED, several font handles have been defined. In IO they are declared as having special names, which corresponds to the use of them, e.i. one font handle is named as KEYWORD_FONT. These font handles can then be used with the function **select_font** to select a new font. To get the current font use the function **get_font**.

A font also determine the current size of the characters on the screen. To retrieve information about the current width and height of characters use the two functions **get_char_width** and **get_char_height**. If the width of a complete string is wanted use the function **sizeof_text**.

File I/O

Two functions are defined for file input and output. They operate on a FILEHANDLE, which is a Windows file handle, *not* a standard C file handle [1].

- **writefile** writes a number of bytes from a buffer to a file. If the operation went fine the function return true.
- **readfile** writes a number of bytes into a buffer from a file. If the operation went fine the function returns true.

Other functions

To retrieve information about the width and height of the current window, use the two functions **get_window_width** and **get_window_height**.

To change the shape of the mouse two functions are available: **show_mouse_busy**, which displays the busy mouse shape, and **show_mouse_normal**, which displays the normal mouse shape. The functions should be used if a program operation could take some time.

A function to write to the window is **text_out**, which display a string at a specified (x,y)-pixel coordinate.

To assist displaying error a function **error** is available. It displays a message window on the application, with the error as parameter.

9.2 MYSTRING

The MYSTRING unit contains five string functions e.g. copying and comparison between strings. They are used the same way as the standard C functions, but some of them are implemented through Win32s Windows functions. Since they are a part of the Windows library, they does not fill up memory space, because the Windows library already are linked into the program.

[1] unfortunately they are defined with equal names, but the type structure is different

9.3 DEF

The small unit DEF (see appendix F.1), defines some basic types: a boolean type BOOL, a view VIEW, a node-id NODEID, a OUTPUT type for use in unparsing, and it declared some basic constants: FALSE and TRUE and the NULL pointer.

Chapter 10

User Interface

One important element in constructing programs is the user-interface design. This will be described in this chapter.

In the 80'es, the acknowledgement of user interfaces has grown. In many programs the code to communicate between user and program consists of more than 80% of the entire source code. That means that a lot of time is invested in programming the interface. This has lead to more abstract methods/tools such as object oriented interfaces, dialog designers and complete visual window designers.

Several of these object oriented interfaces such as Borlands Object Windows, Visual C++ and Microsoft Foundation Classes (MFC) are all abstractors, e.i. they abstract from the 'tedious'[1] C-code style of programming windows. The member functions in the objects do the necessary calls to the windows functions. Because objects are used, it is easy to extend and maintain them.

In RSLED the decision about which application interface to use was determined to be Microsoft Windows Win32s[Win32]. Win32s support most aspects of developing applications to exploit the power of 32-bits operating systems. These includes window management, graphic device interface, memory management and file I/O. Win32s is however not an objected oriented interface and libraries such as MFC cannot be used[2]. That means that all programming must be done in the old C-style fashion.

RSLED is heavily inspired by CSG generated editors, so the user interface will be that too. These editors are divided into 3 areas: the main edit area, a area showing allowable transformation and a area which tells the name of the current selection. Furthermore, do editors which use CSG from GrammaTech[GrammaTech 92], have a menu bar. From the menu bar several different commands are possible: file I/O commands, edit commands, view commands, options commands, structure commands, text commands and help commands.

10.1 Introducing Win32s

When programming to Win32s every Windows-based application creates at least one window, called the *main window* that serves as the main window for the application. The application does

[1] some think its tedious

[2] At least not in Symantec C++ 6.1

not call explicit function calls to obtain input (such as **getchar()**). Instead, they wait for Windows to pass input to them. Input comes in form of *messages* which are created by other applications or input events e.g. input from the keyboard or by moving the mouse. Messages are received in a special window function, called the *window procedure*, which processes the input and returns control to the Windows operating system. Each window procedure has four parameters (see figure 10.1): a window handle, a message identifier, and two 32-bit values called *message parameters*. The window handle identifies the window for which the message is intended. Message parameters specify data or the location of data used by a window procedure when processing a message. The meaning and value of the message parameters depend on the message.

```
LONG APIENTRY WndProc(hwnd,uMsg,wParam,lParam)
HWND hwnd;
UINT uMsg;
WPARAM wParam;
LPARAM lParam;
{
  RECT rcClient;

  switch(uMsg) {
     case WM_CREATE: /* Code during creation of window */                    10
       /* ... */
       return 0;

     /* Process other messages */

     case WM_SIZE: /* Code if window size changes */

       GetClientRect(hwnd,&rcClient);

       /* Move child windows */                                              20

       return 0;

     case WM_DESTROY: /* Code when destroying window */

       /* User code to destroy */

       PostQuitMessage(0);
       return 0;
  }                                                                          30
  return DefWindowProc(hwnd,uMsg,wParam,lParam);
```

Figure 10.1: Example of window procedure

10.1.1 Messageloop

An application must process messages which is send to it by other applications. This is handled by a message loop, which retrieves messages from other application, and sends them to the appropriate windows (child windows) procedure for processing. The normal message loop look like:

```
MSG msg;

while(GetMessage(&msg,(HWND) NULL,0,0)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
return msg.wParam;
```

Figure 10.2: Example of message loop

10.1.2 Handles

Every object an the screen is referenced through a so-called handle, which essential is a pointer to that object. Handles exists for windows, dialogs, files, buttons, fonts and all other kinds of objects. When doing output do e.g. a window a handle to a device context (HDC) must be obtained. This uniquely represent the window which should retrieve output. When output are finished the device context must be released.

10.2 Windows in RSLED

RSLED consists of four windows - the main window and three child windows: the edit window, the position window and the transformation window.

All four window classes are created during startup in the initial WinMain function, by calling a function **InitRSLEDWIN**. This function registers the four window classes (see figure 10.3) and makes the initial main window creation (see figure 10.4).

The four window procedures, which processes the messages resides in four different source files: **rsledwin.cpp**, **editwnd.cpp**, **poswnd.cpp** and **transwnd.cpp**.

10.2.1 Main Window Procedure

The main window procedure **WndProc** (see appendix G.11) processes, beyond the default ones, input from the menu bar (WM_COMMAND with wParam as the menu command IDM_xx) and keyboard input messages WM_CHAR and WM_KEYDOWN, which is processed by simply sending them to the edit child window.

When IDM_OPEN, IDM_SAVE and IDM_SAVEAS are received appropriate calls to the common FileOpen or SaveAs dialogs are entered.

10.2.2 Edit Window Procedure

The edit window procedure is the central nerve in RSLED. It receives all keyboard and mouse input[3]. It paints the central editor text and controls the scroll bars.

[3] of course it only receives mouse input when the mouse is within the edit window

```
HINSTANCE hInstance;
static char szAppName = "RSLED";
.
.
.

WNDCLASS      wndclass;

// Registers the window
                                                                    10
wndclass.style              = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc        = WndProc; /* Name of window procedure */
wndclass.cbClsExtra         = 0;
wndclass.cbWndExtra         = 0;
wndclass.hInstance          = hInstance;
wndclass.hIcon              = LoadIcon(hInstance,szAppName);
wndclass.hCursor            = LoadCursor(NULL,IDC_ARROW);
wndclass.hbrBackground      = GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName       = "RSLMENU";
wndclass.lpszClassName      = szAppName;                            20

if(!RegisterClass(&wndclass)) return FALSE;
```

Figure 10.3: Example of window registration

Window management

A major topic is how the contents in the windows are updated. Whenever some part of the window has been overwritten a WM_PAINT command are received with information of which area to repaint. The information of what should be displayed is contained in the syntax tree and in the unparsing schemes.

An optionally solution is to make the updating as quick and efficient that only the needed part of the screen is rewritten.

In RSLED a very simple, and not very efficient solution, has been used. Whenever the window needs to be updated the complete syntax tree is unparsed to the screen. Windows control which part of the text which is rewritten, by using the repaint area information. This means that the user does not see any flickering of the screen.

Scrolling

Scrolling is implemented through two variables, (**ViewOrgX** and **ViewOrgY**, holding the current origin of the window view. By using the Window function **SetWindowOrgEx** information about which part of the virtual window to display.

Whenever the user activates the scroll bars, two variables are updated, and a repainting message are issued. Two variables **XPos** and **YPos** contains the current horizontal and vertical scroll position. Since the scroll bars should reflect the size of the edited text, e.i. whenever the scroll position is positioned in the middle of the vertical scroll bar, the view in the window should be in the middle of the edited text, a control mechanism is needed. Fortunately, Windows also offers

```
HINSTANCE     hInstance;
HWND          hwnd;
.
. /* Register the window class for the window */
.

/* Create the window */

hwnd = CreateWindow(
  szAppName,              /* class name */                          10
  "Window Name",          /* window name */
  WS_OVERLAPPEDWINDOW | /* overlapped window style */
    WS_HSCROLL |          /* horizontal scroll bar */
    WS_VSCROLL,           /* vertical scroll bar */
  CW_USEDEFAULT,          /* default horizontal position */
  CW_USEDEFAULT,          /* default vertical position */
  CW_USEDEFAULT,          /* default width */
  CW_USEDEFAULT,          /* default height */
  (HWND) NULL,            /* no parent or owner window */
  (HMENU) NULL,           /* class menu used */                     20
  hInstance,              /* instance handle */
  NULL);                  /* no window creation data */

if(!hwnd) return FALSE;

// Show window (send WM_PAINT) and update it

ShowWindow(hwnd,SW_SHOWDEFAULT);
UpdateWindow(hwnd);
                                                                    30
```

Figure 10.4: Example of window creation

this by the **SetScrollRange** function. In the vertical area the scroll range are continuely updated whenever a new line is inserted or deleted in the edited text. This information is fetched from the movement commands and from the view of the complete syntax tree (the **root**).

The actual scrolling is done by scrolling the part of the view which is on the screen (using Windows **ScrollWindowEx** function) and then invalidate the area in which new text should be displayed.

Multiple Node Selection

Selection of multiple node is handled by controlling the mouse buttons together with a variable **MultListnodesSelected**. Whenever the mouse is moved, while the left button is pressed, it is checked if the mouse points at a new selection. If the (new) selection is outside the previous selections view, then it is examined if it is a child of a list node. If it is then **MultListnodesSelected** variable is set to true.

The multiple nodes selected, all have their respective **selected** equal to true in the node information.

When the user issues cut or pasting commands the system checks (by checking **MultList-**

nodesSelected) whether multiple nodes should be deleted or inserted in the tree.

As soon as the user changes the selection to a new node which is not a child of a list node all multiple nodes are un-selected.

Text input

Textual editing are activated as soon the user starts typing normal characters. A variable **DoingTextEdit** are set while the user are in textual editing mode. Appropriate functions in the TextBuffer class (see chapter 11) are called, when the user issues keys for moving the caret, deleting a character etc.

To stop the textual editing mode by issuing a movement command.

10.2.3 Position Window Procedure

The position window is the smallest window on the screen and its function is to display the (syntactic) name of the current selection.

It receives only WM_PAINT messages from the other windows.

The information to display is received from the **NodeInfoTable**, which contains the current selections syntactic name.

10.2.4 Transformation Window Procedure

The transformation window procedure control the button controls which when 'pressed' makes the appropriate transformation (expansion) of the syntax tree. Every time the selection changes a WM_PAINT, message should be received, so that the allowable buttons can be displayed. Actually all buttons are active at all time, but if the non-allowable buttons are moved outside the transformation window, the user cannot activate them.

When a button is activated the syntax tree first are expanded, by calling **makeit** and **insert_tree_first**, and then a message to the other windows to update their contents are transmitted.

At start-up (WM_CREATE) all buttons are created. Information about which text to put on the button is received from the **TransformationTable** in NODEINFO.CPP. Each button is initialized with a number which correspond the the right entrance in the **MakeIt** function (see NODEINFO.CPP).

The displayment (WM_PAINT) of the right transformation buttons is controlled by examine the current selection. The selections node identifier is used to search through the domain of the **TransformationTable**. When the node identifier matches the number in the table the first transformation are found. Subsequent transformation is then placed subsequently until the nodeid again is unmatched.

To save time when displaying the buttons a variable **oldnodeid**, contains information about which transformation are currently displayed. If the user at any time changes the selection to a

node which has the same node identification as the old one, then there is no need to redisplay the transformation buttons.

Chapter 11

TEXTBUF

This chapter explains the implemented textbuffer. The textbuffer is used as a part of small editor. It is implemented as a class (see figure 11.1 below).

```
#define MaxPos 255

class TextBuffer {
  char *buffer;
  int charPos,charMaxPos;
  VIEW bufview;
  HWND hwnd;
public:
  TextBuffer();
  TextBuffer(HWND,VIEW,char *);                                    10
  ~TextBuffer();
  char *get_buffer(void);
  int get_buflen(void) { return charMaxPos; }

  void insert_char(char);
  void delete_char(void);
  void delete_prev_char(void);

  void left(void);
  void right(void);                                                20

  void paint(void);
  void update_caret(void);
  void erase(int);
};
```

Figure 11.1: Implemented text buffer in RSLED

The attributes consits of: a character pointer **buffer**, which points at the characters in the buffer, variables for handling the current length of the buffer **charMaxPos** and the current (cursor/caret) position in the buffer **charPos**, a view represent location and size in the window **bufview** and a handle **hwnd** representing the window in which the buffer are located.

The member functions are divided into five parts: constructor and destructors, information retrieval functions, character insertion and deletion functions, caret movement functions and display functions.

The maximum buffer length is in this implementation is defined to be 255 characters. Each time a buffer is needed, the buffer is allocated. In later implementations it should be a dynamically defined textbuffer.

The functions are used in the edit window procedure (see sec. 10.2.2), where the binding between user input and text buffer functions are defined.

There has not been so much time to find a more appropiate class design of the textbuffer, but a more elegant solution would be to first develop one class which represent a buffer, then a class which represent a editor, and finally a class which represent how an editor are displayed (see figure 11.2).

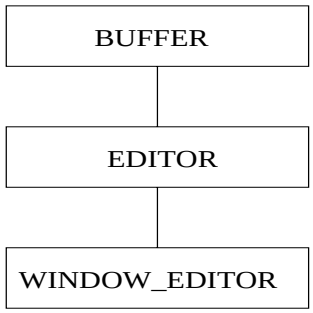

Figure 11.2: Proposal for new text buffer class hierarchy.

Chapter 12

Conclusion

12.1 Project results

A (generic) basic kernel for a syntax-directed editor is specified and a specific editor for the RSL language has been implemented. The editor is in alfa revision, meaning that it can be used, but errors may occur. Below is a list of features implemented and features which is missing.

Is implemented	Is not implemented
Basic kernel	Holophrasting
Basic unparsing	Fencing
Elision	Advanced text buffer
Parsing	
File I/O	
Graphic user interface	
Windowing	
Scrolling	
Multiple node selection	
Font selection	

Table 12.1: Table of features implemented/not implemented.

RSLED do have that advantage that fencing not is implemented. This means for example that the user must insert appropriate parentheses around expressions which should have higher precedence. The parser though do parses constructs with respect to precedence, e.g. if the user inputs 1+2*3, then the syntax tree represents 1+(2*3). If (1+2)*3 is wanted then naturally parentheses must be present. If transformations are used for building the constructs, then the user must remember to transform constructs into bracketed constructs.

The specification part I, has been specified in a generic way, e.i. it can be used to developing other syntax-directed editors.

The editor is implemented in C++ and using Windows Win32s.

12.2 Future improvements and extensions

This section are dealing with matters of possible improvements of the editor in the future.

12.2.1 Semantic checking

In CSG generated editors one can use the notion of attribute grammars, to check for semantics. This allows checks for unknown identifiers, illegal types, function application errors and so on.

This section examines how the editor could be extended with such a semantic checking through use of attributes.

Attributes are essential just variables which contains semantic information. The information could be type information e.i. integer type, boolean type etc. or it could be operator precedence information. Since attributes are variables, a logical way to implement them, is to extend each node in the syntax tree with semantic variables. For example could a new class SyntaxTreeAttr be declared, which inherits from the SyntaxTree class and adds more private attributes:

```
enum { integer, real, boolean , ... } TYPE;

class SyntaxTreeAttr: public SyntaxTree {
  TYPE type;
  int precedence_level;
public:

  /* function to handle attributes */
};
```

New member functions must be added to handle the attributes, e.g. sets the attributes or compares two attributes. The most difficult functions would be the one handling incremental attribute evaluation. Several literature exists already, but the most thorough must be Thomas W. Reps ACM Doctoral Dissertation Award 1983 about Generating Language Based Environment [Reps 83]. It contains nearly all necessary information - including algorithms - for using incremental attribute evaluation. A more easy approach is to make the semantic checking static, e.i. it is only executed by a command from the user.

Already defined functions, such as unparsing, must be overwritten to implement unparsing schemes which could extract information from the semantic attributes. This can e.g. be used to implement fencing.

Furthermore a table of identifiers could be created, which would help to check if identifiers already were defined. That means that classes representing lexical terminal nodes (the LexTerminal macro) must be redeclared to point to indexes in the identifier table.

12.2.2 Unparsing

The unparsing in RSLED should be improved. A more efficient algorithm should be developed which only unparses the necessary part of the tree, e.i. the part which could be seen by the user.

Furthermore the system could be more flexible to user requirement, by allowing different unparsing schemes. A solution would be to implement the ability to load different unparsing schemes from a file. The unparsing schemes could then be defined as the user wants them.

Another feature could be a pretty printer to LaTeX. This again would just be definition of another unparsing scheme for each construct. This extension would allow users to insert specifications directly into their own LaTeX documents.

12.2.3 Transformations

Could allow transformations for hole trees which extract information from the old one, e.g. from a while-statement construct to a repeat-until construct without loosing any information. This ability could improve program development.

Another more user friendly extension is transformation for optional elements[1]. If for example the selection currently is a component_kind, then the user normally only see the <:*type_expr*:> transformations, but it would be more user friendly also allowing two extra buttons for opt-destructor and opt-reconstructor. When activated the syntax tree should be extended with the appropriate optional node.

12.2.4 User interface

There is always several features regarding the user interface, which could be implemented in programs. In RSLED a few can be named:

- Use of colors,
- Context sensitive help,
- Multiple clip-buffers,
- Drag and drop from/to multiple (clip) buffers or edit windows.
- Unparsing directly to printer.

From inside Windows all the more advanced features could be used, such as OLE (Object Linking Embedding), MDI (Multiple Document Interfaces) allowing RSLED to edit multiple specifications at the "same" time.

12.2.5 Text editing

The text editing facility should be much more user friendly and more compatible with the *eden* editor. The limit of 255 characters is far too small, and an optimal solution would be to make the text buffer dynamic. It should furthermore be possible to edit multiple lines, which could span beyond the window.

[1] this is implemented in the Unix version of eden, the RAISE Specification Language Editor (GrammaTech v. 4.0)

12.2.6 Make RSLED a generic syntax-directed editor generator

RSLED can "easily" be extended to be an generic structure editor generator.

The main thing is to make a compiler, which as input takes a language grammar and as output produces the necessary files e.i. scanner, parser, node identifiers, unparsing schemes and allowable transformations (see figure 12.1 below).

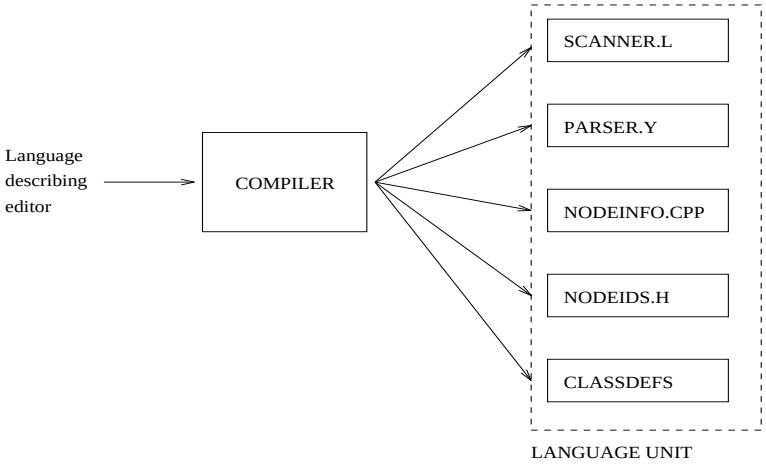

Figure 12.1: Generic language system.

12.3 Final remarks

With this project I have learned a great deal about programming C++ and programming applications to Windows.

At present time I would like to re-specify and reprogram the hole system. I now have better understanding of the problem domain which would make it easier to make a better system. The specifications could be much more precise, capture more details like the connections between window screen and syntax tree. The implementation of the user interface could be adopted to an object oriented approach, with use of a class library e.g. MFC.

Last I hope that this project would be the start of several new projects, which extend RSLED into a fully useful syntax-directed editor environment. I certainly see a few possible projects:

1. Optimization of code, especially the unparsing function.
2. User interface optimization in general. Maybe with a fully detailed examination of the user interface design.
3. Attribute grammar evaluation.

Lyngby, Copenhagen, February 1994

Michael Suodenjoki

Part III

User manual

Chapter 13

User Manual

Welcome to RSLED Version 1.0. RSLED is a syntax-directed editor for the RAISE [1] Specification Language (RSL). To get more information how to understand and using RSL a book is available: The RAISE Specification Language, The RAISE Language Group, The BCS Practioners Series, Prentice Hall, 1992. The syntax of the language can also be seen in appendix A.

A syntax-directed editor is an editor which ensures that the written text is syntactically right, by only allowing to extend the text by doing transformations. Normally syntax-directed editors also allows textual editing facilities. RSLED has also a limited textual facility.

The results from RSLED can be saved as an ordinary ASCII text file, which is compatible with *eden*, the CSG generated editor for RSL to the Unix system. Likewise any output[2] from *eden* can be loaded into RSLED.

RSLED is running under Microsoft Windows 3.1 with Win32s installed. The Win32s API is distributed along with RSLED and must be installed before RSLED can be used. Likewise several font files are needed to be installed before use.

It is necessary that you know the basics of Microsoft Windows, e.i. how to use windows, scroll-bars, mouse, keyboard, menus, buttons and dialogs.

13.1 Installation

To use RSLED the minimum requirements to your system are:

- a 386, 384 or Pentium Intel based processor in your computer,
- you are using VGA or SuperVGA screen,
- have MSDOS 5.0 or higher installed,
- have Microsoft Windows 3.1 installed,
- have at least 3Mb free on your hard disk,

[1] RAISE - Rigorous Approach to Industrial Software Engineering
[2] the output must respect the grammar specified in appendix A

- have a mouse installed under Windows,
- have a 3.5" disc drive,
- have at least 4Mb (preferable 8Mb) memory installed

Two 3.5" discs should companion you: one containing the executable file of RSLED named **RSLWIN32.EXE** and the font files it uses and another disc with the Win32s redistributable files.

To install RSLED follow these steps:

Install Win32s by copying all files from the Win32s disc to your hard disc - preferable create a new directory e.g. C:\WIN32S. From inside Windows you should run the **SETUP.EXE** from your new directory. This should automatically install and setup Win32s for you. Just follow the steps on the screen. If you want to free space from your hard disc you can remove the files from the directory you created e.g. C:\WIN32S.

Install font by choosing the *control panel* and *fonts* group from inside Windows (see figure 13.1 below). This will allow you to add more fonts to your system. Select that you want to add new fonts from drive A: and insert the RSLED disc into drive A. The necessary fonts resides in the A:\FONTS subdirectory.

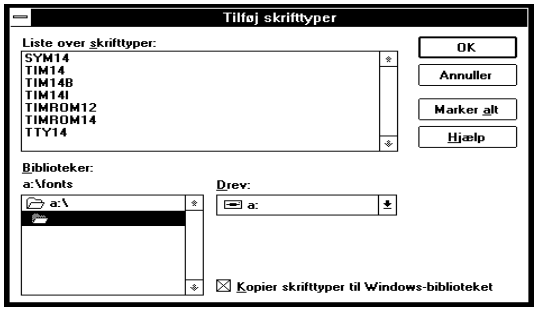

Figure 13.1: Add font dialog.

Install RSLED by copying RSLWIN32.EXE to a directory of your own choice - you can create a new one e.g. C:\RSLED. From inside windows you should open the *program* group and from the File menu item you should select that you want to add a new program element to the program group. You will be asked to enter the path and filename of the program to add. Just enter the path and filename for RSLED e.g. C:\RSLED\RSLWIN32.EXE. If proper installed the RSLED icon should appear in the program group.

You should now be ready to run RSLED, just double click on to the RSLED icon in the program group.

13.2 Abstract Syntax-Trees

In a syntax-directed editor you manipulate abstract-syntax trees. An abstract-syntax tree is a structure which represent the grammatical constructs in RSL.

Even though the program manipulates the abstract-syntax tree you do not see any form of trees on the screen. What is shown on the screen is a tree-traversal of the tree while printing each node in the tree. This process is called *unparsing*.

Take for example the following piece of unparsed RSL:

$$a := a + 1$$

which is a assignment_expr. The assignment will have the abstract syntax tree :

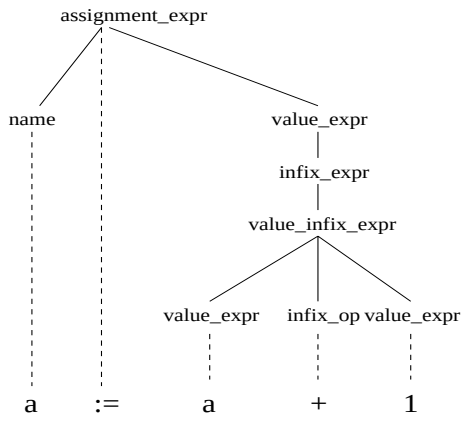

Figure 13.2: Abstract-syntax tree for a := a + 1.

The syntactic construct for the assignment expression is:

$$\mathsf{assigment ::= name := value\_expr}$$

e.i. an assignment consist of a name, a assignment token := and a value_expr.

13.2.1 Selection

The current node which you can manipulate or move from to another node is called the current *selection*. The selection is indicated by being in reverse video. The selection can be changed by either using your mouse or by movement commands. With the mouse you simply move to the wanted syntactic construct and click on the left mouse button. Movement can also be done by commands which traverse through the tree (see sec. 13.2.3).

Several nodes in the syntax tree can be selected by holding the left mouse button down while moving the mouse. In this way a hole list of nodes can be deleted or cutted/pasted to/from a clipbuffer (see menu command Edit in sec. 13.4).

13.2.2 Placeholders

A *placeholder* is a special node in the tree, which represent possible future expansions of the tree. Every placeholder is shown with the name of the node, which can be expanded. For example the root node of the complete syntax tree is a module declaration. When starting a new specification the edit window (about windows see sec. 13.3) contains a selection, which is the placeholder

<:*module_decl*:> and the transformation window contains two buttons - a button for scheme_decl and a button for object_decl (see figure 13.5). When you activate the buttons the placeholder will expand into the desired construct, e.g. if you press the button for scheme_decl the placeholder will transform into:

scheme
  <:*comment*:>
  <:*id*:> = <:*class_expr*:>

and three new placeholders is introduced.

13.2.3 Movement in abstract-syntax trees

How does one move around in the text, when the text is perceived as an abstract-syntax tree ? Before the answer can be given we must define how nodes in the tree refers to each other. To this we use concepts know from genealogical trees. As seen in the figure 13.3 below the tree consists of a root, which is the top node. The root node is the only node which do not have a parent. A node may have any number of children, which again are (sub-)trees. The children of a node can be referred to each other as siblings, e.i. each node may have a left sibling or a right sibling.

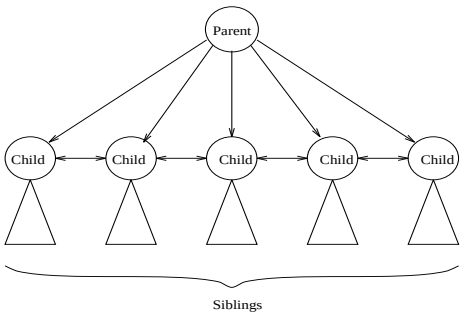

Figure 13.3: Relation between nodes in a genealogical tree.

Traversal

A forward preorder traversal of the syntax tree is a left-first, depth-first traversal, e.i. try first to move to the left child in the tree, then try to move to the right sibling and finally try to move to the parent. Backward preorder traversal of the syntax tree is right-first, depth-first e.i. try first to move to right child in the tree, then trying to move to left sibling and last try to move to the parent.

Resting place

A *resting place* is a node in the syntax tree, which stops a traversal. As soon a traversal meets a resting place the traversal stops and the node is selected. Not all nodes in a syntax tree are resting places. For example would the value_infix_expr node in figure 13.2, not be a resting_place, because it represent a grammar construct, which does not needed to be displayed to you. Since it is never displayed, you cannot select it. The exact rules determining which nodes are resting place

are somewhat complicated, but may be summarized as "only those nodes which are meaningful to the user are resting place".

Movement

Below in figure 13.4 and in table 13.1 you can see the 9 different movement commands.

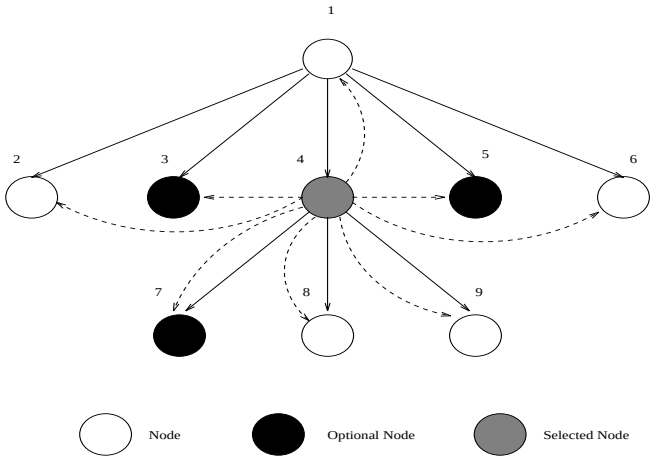

Figure 13.4: Tree movement

Command	Leads to node
`ascend-to-parent`	1
`backward-preorder`	9
`backward-sibling`	2
`backward-sibling-with-optionals`	3
`backward-with-optionals`	1
`forward-preorder`	8
`forward-sibling`	6
`forward-sibling-with-optionals`	5
`forward-with-optionals`	7

Table 13.1: Table of movements from selected node 4.

13.2.4 Optionals

In the figure 13.4 above you could see something called optional nodes. Those nodes reflects optional constructs in the RSL syntax e.g.

scheme_instantiation ::= name opt-actual_scheme_parameter

The meaning of an optional construct is that it *may* appear in the abstract-syntax tree, but it is up to you to decide if you want to use it. Placeholders for optional constructs are not normally shown but after a transformation or by using a movement command that allows optionals, the placeholders for optional elements appear in the abstract-syntax tree (and on the screen). If you do not transform/extend optional constructs they will disappear again when the selection changes.

13.2.5 Lists and Strings

Lists and strings plays a special role in the abstract-syntax tree. They reflect RSL constructs such as

scheme_decl ::= **scheme** scheme_def-list

which means that one or more scheme definitions could appear after each other separated by commas. If several elements appears in a list then they are siblings in the syntax tree. That means that you can move between them by using the movement commands:

- `forward-sibling`
- `forward-sibling-with-optionals`
- `backward-sibling`
- `backward-sibling-with-optionals`

The two movement commands which take optionals under consideration allow you to insert optional placeholders between two list elements. That means that list elements can be inserted in between any two list elements.

13.3 Windows

The main screen is divided in tree windows (see figure 13.5):

Edit window is the window where the edited text are shown. The current selection is shown in reversed video. If the text are too large to fit in the window vertical and/or horizontal scroll-bars appears at either right side or bottom of the edit window. By clicking at the right mouse button in the edit window you enter a forward-preorder with optionals movement.

Position window displays the name of the currently selected syntactic construct.

Transformation window displays buttons which activates transformation of the current selection.

13.4 Menu

The application also has a menu bar from which you can issue different commands such as file loading and saving, cut and paste of abstract-syntax trees, movement commands or change options.

File The file menu item controls loading (see figure 13.7) and saving of your trees (see figure 13.8). The *New* command starts editing of a new specification.

If you are trying to load a file which does not correspond to the RSL syntax, e.i. have syntax errors, it will not be loaded. Instead an error message will be displayed that explains where in the file an error resides.

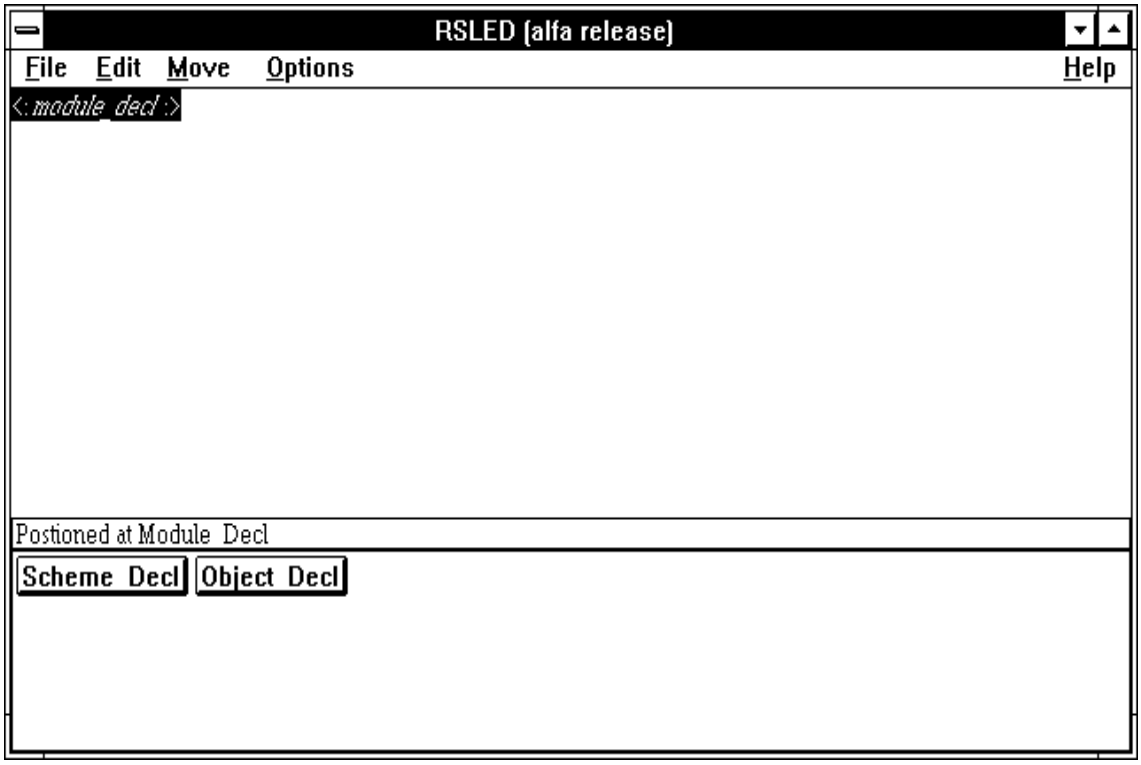

Figure 13.5: Layout of RSLED windows.

Figure 13.6: File menu item.

Edit From within the edit menu item you can cut, copy and paste the abstract-syntax trees to/from a *clipbuffer*.

Overview of commands:

Undo *This function has not yet been implemented !*

Cut Cuts the current selection into the clipbuffer. The current selection changes to the placeholder of the cutted tree.

Copy Copy the current selection into the clipbuffer. Previous content of the clipbuffer are removed. The current selection does not change.

Paste Copy the clipbuffer to the current selection. If the selection is not a placeholder, but a tree, then the tree will be deleted and the selection changes to be the contents from the clipbuffer. The contents of the clipbuffer are unchanged.

Clear Clears (deletes) the current selection. This is equal to issuing a Ctrl-K command. The selection changes to the appropriate placeholder.

Move The commands in this menu item issues movement commands. These changes the selection accordingly to different traversal strategies in the abstract-syntax tree.

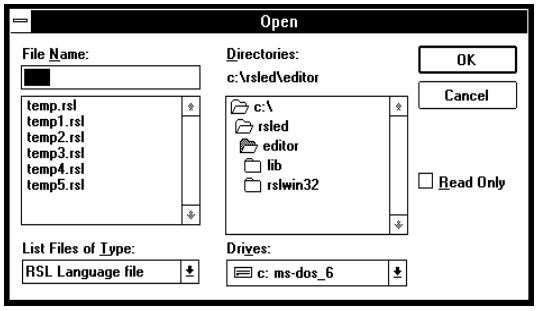

Figure 13.7: Openfile Dialog.

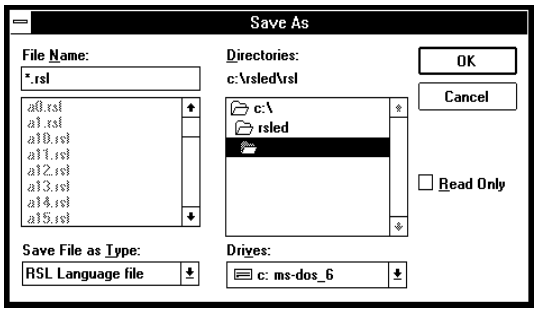

Figure 13.8: SaveAs Dialog.

ascend-to-parent Change the selection to the closest enclosing resting place.

forward-preorder Change the selection to the next resting place in a forward preorder traversal of the abstract-syntax tree. Do not stop at nodes for optional constituents.

forward-with-optionals Change the selection to the next resting place in a forward preorder traversal of the abstract-syntax tree. Stop at nodes for optional constituents.

forward-sibling Bypass all resting places contained within the current selection and advance to the next sibling in a forward preorder traversal of the abstract-syntax tree. If there is no next sibling, ascend to the enclosing resting place and advance to its sibling, etc. Do not stop at nodes for optional constituents.

forward-sibling-with-optionals Same as **forward-sibling**, but stopping, in addition, at nodes for optional constituents.

backward-preorder Change the selection to the previous resting place in a forward preorder traversal of the abstract-syntax tree. Do not stop at nodes for optional constituents.

backward-with-optionals Change the selection to the previous resting place in a forward preorder traversal of the abstract-syntax tree. Stop at nodes for optional constituents.

backward-sibling Bypass all resting places contained within the current selection and advance to the previous sibling in forward preorder traversal of the abstract-syntax tree.

Figure 13.9: Edit menu item.

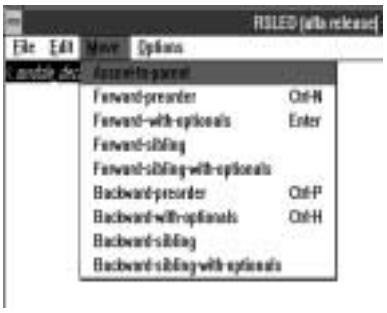

Figure 13.10: Move menu item.

If there is no next sibling, ascend to the enclosing resting place and advance to its previous sibling, *etc.* Do not stop at place-holders for optional constituents.

backward-sibling-with-optionals Same as **backward-sibling**, but stopping, in addition, at nodes for optional constituents.

Options The commands in this menu item allow you to change different options.

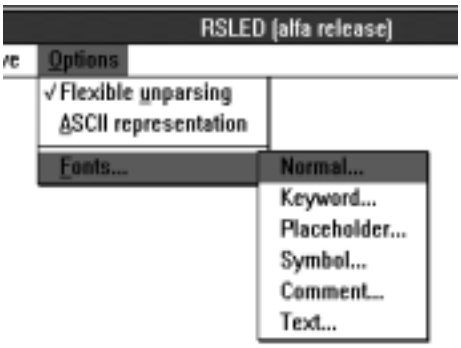

Figure 13.11: Options menu item.

Flexible unparsing *This function is not implemented !* Currently RSLED always do flexible unparsing.

ASCII representation allows you to change have special symbols are displayed e.g. the equivalence symbol is displayed either by ≡or by ASCII `is`.

Fonts... allows you to change which fonts are used to display the different syntactic constructs e.g. keywords, placeholders etc.

Help This menu item has only two sub items, which shows respectively a about message and a version message. In future versions of RSLED it will be expanded with sub items like help index, introduction etc.

13.5 Textual editing

Textual editing as in a normal textual editor can be done by start typing the wanted text. The characters will appear in the edit window at the current selection. When you have entered the

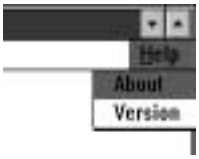

Figure 13.12: Help menu item.

complete text for that syntactic construct you should press ENTER, any movement command or using mouse to select another part, and the text will be parsed into an abstract-syntax tree which is inserted at the current selection. If an parsing error occurs it will be displayed on the screen.

Placeholders can be typed, just type <:*placeholder_name*:>.

Beyond that normal characters can be typed into the text editor, some other keys are available:

Key	Command
Left Arrow	Moves the caret on character position left
Right Arrow	Moves the caret on character position right
Backspace	Deletes the character on left side of caret
Delete	Deletes the character on right side of caret
Enter	Stops editing and starts parser for edited text
Ctrl-K	Stops text editing and restores the selection to its previous state

Table 13.2: Table of special keys in text editor

Inside the text editor special RSL symbols can be referred to by using special ASCII characters. See table in appendix C.

13.6 Elision

Elision is a feature, which better can help you overview your specifications. Any node in the syntax tree, which can be selected, can also be *elided*. When you elides a selection the complete selection changes to only displaying a special elision mark (...). This means that if the selection was very big, it now only uses a small part of the window. Several elisions can be active at one time.

To elide/unelide a selection use Ctrl-E key.

13.7 Other keybindings

The following table shows an overview of the available keys and their function in RSLED:

Key	Function
F1	Issue a `ascend-to-parent` movement command
F2	Issue a `forward-preorder` movement command
F3	Issue a `forward-preorder-with-optionals` movement command
F4	Issue a `forward-sibling` movement command
F5	Issue a `forward-sibling-with-optionals` movement command
F6	Issue a `backward-preorder` movement command
F7	Issue a `backward-preorder-with-optionals` movement command
F8	Issue a `backward-sibling` movement command
F9	Issue a `backward-sibling-with-optionals` movement command
Enter	Issue a `forward-preorder-with-optionals` movement command
Ctrl-K	Deletes the current selection and restore it to a placeholder
Ctrl-N	Issue a `forward-preorder` movement command
Ctrl-P	Issue a `backward-preorder` movement command
Ctrl-E	Elides/Unelides the current selection

Table 13.3: Table of other keybindings.

Appendix A

RSL Syntax

A.1 Conventions

The convention below are used for defining different syntactic optional and lists constructs. For any production name 'p' the following production rules are assumed:

opt-p ::=
  | p

p-string ::=
  p | p p-string

p-list ::=
  p | p , p-list

p-list2 ::=
  p , p |
  p , p-list2

p-choice ::=
  p |
  p | p-choice

p-choice2 ::=
  p | p |
  p | p-choice2

p-product ::=
  p |
  p × p-product

p-product2 ::=
  p × p |
  p × p-product2

A.2 Syntax

Specifications

specification ::=
  module_decl-string

module_decl ::=
  scheme_decl |
  object_decl

Declarations

decl ::=
  scheme_decl |
  object_decl |
  type_decl |
  value_decl |
  variable_decl |
  channel_decl |
  axiom_decl

Scheme Declarations

scheme_decl ::=
  scheme scheme_del-list

scheme_def ::=
  opt-comment-string
  id opt-formal_scheme_parameter = class_expr

formal_scheme_parameter ::=
  (formal_scheme_argument-list)

formal_scheme_argument ::=
  object_def

Object Declarations

object_decl ::=
  object object_def-list

object_def ::=
  opt-comment-string
  id opt-formal_array_parameter : class_expr

formal_array_parameter ::=
  [typing-list]

Type Declarations

type_decl ::=
  type type_def-list

type_def ::=
  sort_def |
  variant_def |
  union_def |
  short_record_def |
  abbreviation_def

Sort Definitions

sort_def ::=
  opt-comment-string id

Variant Definitions

variant_def ::=
  opt-comment-string id == variant-choice

variant ::=
  constructor |
  record_variant

record_variant ::=
  constructor (component_kind-list)

component_kind ::=
  opt-dectructor type_expr opt-reconstructor

constructor ::=
  id_or_op |
  —

desctructor ::=
  id_or_op :

reconstructor ::=
  ↔ id_or_op

Union Definitions

union_def ::=
  opt-comment-string
  id = name_or_wildcard-choice2

name_or_wildcard ::=
  name |
  —

Short Record Definitions

short_record_def ::=
  opt-comment-string
  id :: component_kind-string

Abbreviation Definitions

abbreviation_def ::=
  opt-comment-string id = type_expr

Value Declarations

value_decl ::=
  value value_def-list

value_def ::=
  commented_typing |
  explicit_value_def |
  implicit_value_def |
  explicit_function_def |
  implicit_function_def

Explicit Value Definitions

explicit_value_def ::=
  opt-comment-string
  single_typing = pure-value_expr

Implicit Value Definitions

implicit_value_def ::=
  opt-comment-string
  single_typing = restrictrion

Explicit Function Definitions

explicit_function_def ::=
  opt-comment-string single_typing
  formal_function_application ≡ value_expr
  opt-pre_condition

formal_function_application ::=
  id_application |
  prefix_application |
  infix_application

id_application ::=
  id formal_function_parameter-string

formal_function_parameter ::=
  (opt-binding-list)

prefix_application ::=
  prefix_op id

infix_application ::=
  id infix_op id

Implicit Function Definitions

implicit_function_def ::=
  opt-comment-string
  single_typing formal_function_application
  post_condition opt-pre_condition

Variable Declarations

variable_decl ::=
  variable variable_def-list

variable_def ::=
  single_variable_def |
  multiple_variable_def

single_variable_def ::=
  opt-comment-string
  id : type_expr opt-initialisation

initialisation ::=
  := value_expr

multiple_variable_def ::=
  opt-comment-string id-list2 : type_expr

Channel Declarations

channel_decl ::=
  channel channel_def-list

channel_def ::=
  single_channel_def |
  multiple_channel_def

single_channel_def ::=
  opt-comment-string id : type_expr

multiple_channel_def ::=
  opt-comment-string id-list2 : type_expr

Axiom Declarations

axiom_decl ::=
  axiom opt-axiom_quantification
  axiom_def-list

axiom_quantification ::=
  forall typing_list •

axiom_def ::=
  opt-comment-string opt-axiom_naming
  value_expr

axiom_naming ::=
  [id]

Class Expressions

class_expr ::=
  basic_class_expr |
  extending_class_expr |
  hiding_class_expr |
  renaming_class_expr |
  scheme_instantiation

Basic Class Expressions

basic_class_expr ::=
  class opt-decl-string **end**

Extending Class Expressions

extending_class_expr ::=
  extend class_expr **with** class_expr

Hiding Class Expressions

hiding_class_expr ::=
  hide defined_item-list **in** class_expr

Renaming Class Expressions

renaming_class_expr ::=
  use rename_pair-list **in** class_expr

Scheme Instantiations

scheme_instantiation ::=
  name opt-actual_scheme_parameter

actual_scheme_parameter ::=
  (object_expr-list)

Rename Pairs

rename_pair ::=
  defined_item **for** defined_item

Defined Items

defined_item ::=
  id_or_op |
  disambiguated_item

disambiguated_item ::=
  id_or_op : type_expr

Object Expressions

object_expr ::=
  name |
  element_object_expr |
  array_object_expr |
  fitting_object_expr

Element Object Expressions

element_object_expr ::=
  object_expr actual_array_parameter

actual_array_parameter ::=
  [pure-value_expr-list]

Array Object Expressions

array_object_expr ::=
  [| typing_list • object_expr |]

Fitting Object Expressions

fitting_object_expr ::=
  object_expr { rename_pair-list }

Type Expressions

type_expr ::=
  type_literal |
  name |
  product_type_expr |
  set_type_expr |
  list_type_expr |
  map_type_expr |
  function_type_expr |
  subtype_expr |
  bracketed_type_expr

Type Literals

type_literal ::=
  Unit |
  Bool |
  Int |
  Nat |
  Real |
  Text |
  Char|

Product Type Expressions

product_type_expr ::=
  type_expr-product2

Set Type Expressions

set_type_expr ::=
  finite_set_type_expr |
  infinite_set_type_expr

finite_set_type_expr ::=
  type_expr **-set**

infinite_set_type_expr ::=
  type_expr **-infset**

List Type Expressions

list_type_expr ::=
  finite_list_type_expr |
  infinite_list_type_expr

finite_list_type_expr ::=
  type_expr *

infinite_list_type_expr ::=
  type_expr $^\omega$

Map Type Expressions

map_type_expr ::=
  type_expr $\overrightarrow{m}$ type_expr

Function Type Expressions

function_type_expr ::=
  type_expr function_arrow result_desc

function_arrow ::=
  $\xrightarrow{\sim}$ |
  $\rightarrow$

result_desc ::=
  opt-access_desc-string type_expr

Subtype Expressions

subtype_expr ::=
  {| single_typing restriction |}

Bracketed Type Expressions

bracketed_type_expr ::=
  (type_expr)

Access Descriptions

access_desc ::=
  access_mode access-list

access_mode ::=
  read |
  write |
  in |
  out

access ::=
  name |
  enumerated_access |
  completed_access |
  comprehended_access

enumerated_access ::=
  { opt-access-list }

completed_access ::=
  opt-qualification **any**

comprehended_access ::=
  { access | set_limitation }

Value Expressions

value_expr ::=
  value_literal |
  name |
  pre_name |
  basic_expr |
  product_expr |
  set_expr |
  list_expr |
  map_expr |
  function_expr |
  application_expr |
  quantified_expr |
  equivalence_expr |
  post_expr |
  disambiguation_expr |
  bracketed_expr |
  infix_expr |
  prefix_expr |
  comprehended_expr |
  initialise_expr |
  assignment_expr |
  input_expr |
  output_expr |
  structured_expr

Value Literals

value_literal ::=
  unit_literal |
  bool_literal |
  int_literal |
  real_literal |
  text_literal |
  char_literal

unit_literal ::=
  ()

bool_literal ::=
  true |
  false

Pre Names

pre_name ::=
  name `

Basic Expressions

basic_expr ::=
  chaos |
  skip |
  stop |
  swap

Product Expressions

product_expr ::=
  (value_expr-list2)

Set Expressions

set_expr ::=
  ranged_set_expr |
  enumerated_set_expr |
  comprehended_set_expr

Ranged Set Expressions

ranged_set_expr ::=
  { value_expr .. value_expr }

Enumerated Set Expressions

enumerated_set_expr ::=
  { opt-value_expr-list }

Comprehended Set Expressions

comprehended_set_expr ::=
  { value_expr | set_limitation }

set_limitation ::=
  typing-list opt-restriction

restriction ::=
  • value_expr

List Expressions

list_expr ::=
  ranged_list_expr |
  enumerated_list_expr |
  comprehended_list_expr

Ranged List Expressions

ranged_list_expr ::=
  ⟨ value_expr .. value_epxr ⟩

Enumerated List Expressions

enumerated_list_expr ::=
  ⟨ opt-value_expr-list ⟩

Comprehended List Expressions

comprehended_list_expr ::=
  ⟨ value_expr | list_limitation ⟩

list_limitation ::=
  binding **in** value_expr
  opt-restriction

Map Expressions

map_expr ::=
  enumerated_map_expr |
  comprehended_map_expr

Enumerated Map Expressions

enumerated_map_expr ::=
  [opt-value_expr_pair_list]

value_expr_pair_list ::=
  value_expr ↦ value_expr

Comprehended Map Expressions

comprehended_map_expr ::=
  [value_expr_pair | set_limitation]

Function Expressions

function_expr ::=
  λ lambda_parameter • value_expr

lambda_parameter ::=
  lambda_typing |
  single_typing

lambda_typing ::=
  (opt-typing-list)

Application Expressions

application_expr ::=
  value_expr
  actual_function_parameter-string

actual_function_parameter ::=
  (opt-value_expr-list)

Quantified Expressions

quantified_expr ::=
  quantifier typing-list restriction

quantifier ::=
  ∀ |
  ∃ |
  ∃!

Equivalence Expressions

equivalence_expr ::=
  value_expr ≡ value_expr opt-pre_condition

pre_condition ::=
  pre value_expr

Post Expressions

post_expr ::=
  value_expr
  post_condition opt-pre_condition

post_condition ::=
  opt-result_naming
  post value_expr

result_naming ::=
  as binding

Disambiguation Expressions

disambiguation_expr ::=
  value_expr : type_expr

Bracketed Expressions

bracketed_expr ::=
  (value_expr)

Infix Expressions

infix_expr ::=
  stmt_infix_expr |
  axiom_infix_expr |
  value_infix_expr

Statement Infix Expressions

stmt_infix_expr ::=
  value_expr infix_combinator value_expr

Axiom Infix Expressions

axiom_infix_expr ::=
  value_expr infix_connective value_expr

Value Infix Expressions

value_infix_expr ::=
  value_expr infix_op value_expr

Prefix Expressions

prefix_expr ::=
  axiom_prefix_expr |
  universal_prefix_expr |
  value_prefix_expr

Axiom Prefix Expressions

axiom_prefix_expr ::=
  prefix_connective value_expr

Universal Prefix Expressions

universal_prefix_expr ::=
  □ value_expr

Value Prefix Expressions

value_prefix_expr ::=
  prefix_op value_expr

Comprehended Expressions

comprehended_expr ::=
  infix_combinator
  { value_expr | set_limitation }

Initialise Expressions

initialise_expr ::=
  opt-qualification **initialise**

Assignment Expressions

assignment_expr ::=
  name :=value_expr

Input Expressions

input_expr ::=
  name ?

Output Expressions

output_expr ::=
  name ! value_expr

Structured Expressions

structured_expr ::=
  local_expr |
  let_expr |
  if_expr |
  case_expr |
  while_expr |
  until_expr |
  for_expr

Local Expressions

local_expr ::=
  local opt-decl-string **in** value_expr **end**

Let Expressions

let_expr ::=
  let let_def-list **in** value_expr **end**

let_def ::=
  typing |
  explicit_let |
  implicit_let

explicit_let ::=
  let_binding = value_expr

implicit_let ::=
  single_typing restriction

let_binding ::=
  binding |
  record_pattern |
  list_pattern

If Expressions

if_expr ::=
  if value_expr **then** value_expr
  opt-elsif_branch-string
  opt-else_branch
  end

elsif_branch ::=
  elsif value_expr **then** value_expr

else_branch ::=
  else value_expr

Case Expressions

case_expr ::=
  case value_expr **of** case_branch-list **end**

case_branch ::=
  pattern → value_expr

While Expression

while_expr ::=
  while value_expr
  do value_expr **end**

Until Expressions

until_expr ::=
  do value_expr
  until value_expr **end**

For Expressions

for_expr ::=
  for list_limitation **do** value_expr **end**

Bindings

binding ::=
  id_or_op |
  product_binding

product_binding ::=
  (binding-list2)

Typings

typing ::=
  single_typing |
  multiple_typing

single_typing ::=
  binding : type_expr

multiple_typing ::=
  binding-list2 : type_expr

commented_typing ::=
  opt-comment-string typing

Patterns

pattern ::=
  value_literal |
  pure_value-name |
  wildcard_pattern |
  product_pattern |
  record_pattern |
  list_pattern

Wildcard Patterns

wildcard_pattern ::=
  _

Product Patterns

product_pattern ::=
  (inner_pattern-list2)

Record Patterns

record_pattern ::=
  name (inner_pattern-list)

List Patterns

list_pattern ::=
  enumerated_list_pattern |
  concatenated_list_pattern |
  right_list_pattern

Enumerated List Patterns

enumerated_list_pattern ::=
  ⟨ opt-inner_pattern-list ⟩

Concatenated List Patterns

concatenated_list_pattern ::=
  enumerated_list_pattern ⌢ inner_pattern

Right List Patterns

right_list_pattern ::=
  id ⌢ enumerated_list_pattern

Inner Patterns

inner_pattern ::=
  value_literal |
  id_or_op |
  wildcard_pattern |
  product_pattern |
  record_pattern |
  list_pattern |
  equality_pattern

Equality Patterns

equality_pattern ::=
  = name

Names

name ::=
  qualified_id |
  qualified_op

Qualified Identifiers

qualified_id ::=
  opt-qualification id

qualification ::=
  object_expr

Qualified Operators

qualified_op ::=
  opt-qualification (op)

Identifiers and Operators

id_or_op ::=
  id |
  op

op ::=
  infix_op |
  prefix_op

Infix Operators

infix_op ::=
  = |
  ≠ |
  > |
  < |
  ≥ |
  ≤ |
  ∪ |
  ⊂ |
  ⊃ |
  ⊆ |
  ∈ |
  ∉ |
  + |
  − |
  \ |
  ∖ |
  ∪ |
  † |
  * |
  / |
  ∘ |
  ∩ |
  ↑

Prefix Operators

prefix_op ::=
  abs |
  int |
  real |
  card |
  len |
  inds |
  elems |
  hd |
  tl |
  dom |
  rng

Connectives

connective ::=
  infix_connective |
  prefix_connective

Infix Connectives

infix_connective ::=
  ⇒ |
  ∨ |
  ∧

Prefix Connectives

prefix_connective ::=
  ∼

Infix Combinators

infix_combinator ::=
  [] |
  ∏ |
  || |
  # |
  ;

A.3 Lexical productions

This section deals with productions which explains the most basic constructs in RSL. The productions reflects the lexemes which are recognized by the scanner.

Identifiers

id ::=
  letter opt-letter_or_digit_or_prime-string

letter_or_digit_or_prime ::=
  letter | digit | underline | prime

letter ::=
  ascii_letter

Comments

comment ::=
  '/*' comment_item-string '*/'

comment_item ::=
  comment_char

comment_char ::=
  ascii_letter | digit | prime | quote | backslash

Integers

int_literal ::=
  digit-string

Reals

real_literal ::=
  digit-string '.' digit-string

Texts

text_literal ::=
  '"' opt-text_character-string '"'

text_character ::=
  character | prime

Chars

char_literal ::=
  "' char_character "'

char_character ::=
  character | quote

character ::=
  ascii_letter | digit | graphic | escape

Digits

digit ::=
  '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

ASCII letters

ascii_letter ::=
  'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' |
  'j' | 'k' | 'l' | 'm' | 'n' | 'o | 'p' | 'q' | 'r' |
  's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' |
  'A | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' |
  'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' |
  'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

Special characters

underline ::=
  '_'

prime ::=
  "'

quote ::=
  '"'

backslash ::=
  '\'

graphic ::=
  ' ' | '!' | '#' | '$' | '%' | '&' | '(' | ')' | '*' |
  '+' | ',' | '-' | '.' | '/' | ':' | ';' | '<' | '=' |
  '>' | '?' | '@' | '[' | ']' | '^' | '_' | '`' | '{' |
  '|' | '}' | '~'

Escape characters

escape ::=
  '\r' | '\n' | '\t' | '\a' | '\b' | '\f' | '\v' | '\?' | '\" |
  '\"' | '\x' hex_constant

hex_constant ::=
  hex_digit-string

hex_digit ::=
  digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'A' | 'B' |
  'C' | 'D' | 'E' | 'F'

Appendix B

Precedence and Associativity of Operators

Value operator precedence - increasing		
Prec	Operator(s)	Associativity
14	$\square\ \lambda\ \forall\ \exists\ \exists!$	Right
13	$\equiv$ **post**	
12	$[]\ [\!]\ \|\|\ \|\!\|$	Right
11	;	Right
10	:=	
9	$\Rightarrow$	Right
8	$\vee$	Right
7	$\wedge$	Right
6	$= \neq > < \geq \leq \subset \subseteq \supset \supseteq \in \notin$	
5	$+\ -\ \setminus\ \frown\ \cup\ \dagger$	Left
4	$*\ /\ \circ\ \cap$	Left
3	$\uparrow$	
2	:	
1	$\sim$ prefix_op	

Type operator precedence - increasing		
Prec	Operator(s)	Associativity
3	$\overrightarrow{m}\ \xrightarrow{\sim}\ \rightarrow$	Right
2	$\times$	
1	**-set -infset** $^*\ ^\omega$	

Table B.1: Precedence and associativity of RSL operators

Appendix C

ASCII representation of RSL symbols

ASCII	Symbol	ASCII	Symbol	ASCII	Symbol
><	$\times$	isin	$\in$	~isin	$\notin$
||	$\parallel$	++	$\#$	-\	λ
|=|	$\square$	|^|	$\prod$	-list	*
**	$\uparrow$	-inflist	$^\omega$	~=	$\neq$
/\	$\wedge$	\/	$\vee$	+>	$\mapsto$
>=	$\geq$	exists	$\exists$	all	$\forall$
<=	$\leq$	union	$\cup$	!!	$\dagger$
inter	$\cap$	<<	$\subset$	always	$\Box$
-m->	$\overrightarrow{m}$	<<=	$\subseteq$	=>	$\Rightarrow$
-~->	$\overrightarrow{\sim}$	>>	$\supset$	is	$\equiv$
->	$\rightarrow$	>>=	$\supseteq$	<->	$\leftrightarrow$
#	$\circ$	<.	$\langle$	.>	$\rangle$
:-	$\bullet$				

Table C.1: Table of ASCII representation of RSL symbols

Appendix D

Symantec C++ 6.0/6.1 details

The compiler used for this project was the Symantec C++ Professional 6.0 from Symantec Corporation. The package contained 17 3.5" disc and four reference manuals. This appendix contains some general information about using Symantec C++.

The use of the compiler has not been an overwhelming succes, since it quickly was realized that several errors was found mainly in the user environment of the editor and in the debugger. For a duration of 6 weeks I reinstalled Symantec 3 times[1] due to errors reported on the harddisc e.g. by isuing the **C:\CHKDSK C: /F** command. This is often caused by the debugger since if the program has failed, then sometimes some .DLL files gets corrupted. I have found no errors in the compiler itself.

D.1 Patch for upgrading from 6.0 to 6.1

A patch for upgrading from version 6.0 to version 6.1 exists on the FTP archive site **ftp.cica.indiana.edu**. The patch should correct a lot of the errors from version 6.0 and the ability to change the editor colors has been added.

D.2 Mailing list

I have been joined to the Symantec mailing list **sym-list symcpp.com**, which should discuss problems using Symantec. This mailing list has not given the feeling that Symantec was a popular software house, since several people has been (very) upset by the responds they get (or they not get) from Symantec technical support. Daily mail arived telling that people wants to get off the mailing list or they want to deliver Symantec back and convert to other products.

I myself has only written to the technical support once **support symcpp.com**, because I had a serious error when I was developing RSLED. The answer was first given to me in mid of february, and at that time I already have found other ways to solve my problem.

[1] each installation take about 1 hour

Appendix E

Flex++ & Bison++ details

This appendix contains some detailed information of how to obtain and how to use Flex++ and Bison++.

E.1 How to obtain

The Flex++ and Bison++ source code is available at most "popular" ftp sites, e.i. ftp.nic.funet, wuarchive.wustl.edu. However there is one ftp site, which must be considered the main site of Flex++ and Bison++ source:

> University of Nebraska - Lincoln
> Department of Computer Science and Engineering
>
> FTP site : cse.unl.edu
> Directory: pub/nandy/c++/tools/LATEST

The directory is moderated by Prashant Nandavan, Email address: nandy@tamana.unl.edu

Currently (28th of February), the following files should be available:

bison++-1.21-8.DOCps.tar.Z
bison++-1.21-8.DOSexe.tar.Z
bison++-1.21-8.tar.Z
flex++-2.3.8-7.DOCps.tar.Z
flex++-2.3.8-7.DOSexe.tar.Z
flex++-2.3.8-7.tar.Z
flex++bison++.README
flex++bison++misc-5.tar.Z

The following commands will download the source:

```
> ftp cse.unl.edu
Connected to cse.unl.edu.
220-
```

```
 University of Nebraska - Lincoln
 Department of Computer Science and Engineering
 Unauthorized Access to This System or Network is Prohibited

220 cse.unl.edu FTP server ready.
Name (cse.unl.edu:ms): ftp
331 Guest login ok, type your name as password.
Password: ms@id.dth.dk
230 Guest login ok, access restrictions apply.
ftp> cd pub/nandy/c++/tools/LATEST
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> prompt
Interactive mode off.
ftp> mget *
...
ftp> quit
```

The files are in compressed format and must first be uncompressed and restored with the tape archiver tar. To uncompress a file use:

gunzip filename.Z

To restore tape archiver:

tar xf filename.tar

E.2 How to use

When both Flex++ and Bison++ has been downloaded you can either use the DOS executables (the files in *xx*.DOSexe) on a PC or try to compile the source code under Unix (this may not be easy).

First of all you should print the documentation on a printer. It is a lot easier to understand how things are used, when one have read the documentation.

With the source code some sample input files for Flex++ and Bison++ are distributed. They are resided in the file flex++bison++misc-5.tar.Z.

In RSLED it have been necessary to use Flex++ on the PC and due to some memory problems on the PC, to use Bison++ on Unix[1]. On the PC the executable file of Flex++ is called flex_pp.

To use Flex++ simply write:

C:\ > flex_pp -8 -hscanner.h -oscanner.cpp -Sflexskel.cpp scanner.l

[1] It was quite easy to compile Bison++ on Unix, but the directory where Bison++ resides, demands that a subdirectory called lib exists with the bison.cc and bison.h files.

Here the input file to Flex++ is scanner.l (see appendix H.1). The outfiles are names scanner.h and scanner.cpp. The -8 parameter tells Flex++ to use 8 bit constants for tokens to return to the parser.

To use Bison++ simply write:

```
/home/??: bison++ -oparser.cpp parser.y
```

Here the input file to Bison++ is parser.y (see appendix H.2). The output file name are parser.cpp.

I hope some of this information will be useful.

Appendix F

Specification .H files

F.1 DEF.H

```
/*************************************************************************
*
* DEF.H
*
* Type Definitions & Constants
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef DEF_H
#define DEF_H

/************** Type Definitions **********/

#ifndef BOOL
typedef int BOOL;
#endif
typedef struct {
  int x,y;
  unsigned int width,height;
} VIEW;
typedef unsigned int NODEID;
typedef enum { none, to_screen, to_file, to_buffer } OUTPUT;
typedef unsigned char SYMBOL;

/************** Constants *****************/

#ifndef TRUE
const BOOL TRUE = 1;
#endif

#ifndef FALSE
const BOOL FALSE = 0;
#endif

#ifndef NULL
void * const NULL = 0L;
#endif

#endif
```

F.2 IO.H

```
/*************************************************************************
*
* IO.H
*
* Specification of Input / Output Rutines
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef IO_H
#define IO_H

#include "DEF.H"
#include <WINDOWS.H>

/** Type definitions **/

typedef COLORREF COLOR;
typedef HFONT FONT;
typedef HANDLE FILEHANDLE;

/** Constants **/

const COLOR WHITE = RGB(255,255,255);
const COLOR BLACK = RGB(0,0,0);

#define NORMAL_FONT         hfontNormal
#define KEYWORD_FONT        hfontKeyword
#define PLACEHOLDER_FONT    hfontPlaceholder
#define SYMBOL_FONT         hfontSymbol
#define COMMENT_FONT        hfontComment
#define TEXT_FONT           hfontText

extern FONT NORMAL_FONT;
extern FONT KEYWORD_FONT;
extern FONT PLACEHOLDER_FONT;
extern FONT SYMBOL_FONT;
extern FONT COMMENT_FONT;
extern FONT TEXT_FONT;

/** Screen Rutines **/

void set_outputHDC(HDC);

FONT select_font(FONT);
FONT get_font(void);

void text_out(const int,const int,const char *);
void draw_rect(const VIEW rect);
unsigned int sizeof_text(const char *);

void error(const char *,const char *);

// Color rutunes
void set_bk_color(COLOR);
void set_text_color(COLOR);
COLOR get_text_color(void);

unsigned int get_char_width(void);
unsigned int get_char_height(void);
```

```
unsigned int get_window_width(void);
unsigned int get_window_height(void);

void show_mouse_busy(void);
void show_mouse_normal(void);

/** File Rutines **/

BOOL writefile(FILEHANDLE,const void *,unsigned long);

BOOL readfile(FILEHANDLE,void *,unsigned long,unsigned long &);

#endif
```

F.3 LANGUAGE.H

```
/*************************************************************************
*
* LANGUAGE.H
*
* General Langyage Types and Specific Constants
*
* Created 14 December, 1993, Michael Suodenjoki
* Recreated 20 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef LANGUAGE_H
#define LANGUAGE_H

// Maximal length of identifier
const unsigned int IDLENGTH = 40;
// Number of node-identifiers
const unsigned int NUMNODEIDS = 364;

typedef struct {
  const char *name;
  const char *unp_scheme;
  const char *unp_sep_scheme;
} NODEINFO;

extern NODEINFO NodeInfoTable[NUMNODEIDS];

/*********************************************************
*
* Transformation Schemes for Sample Language
*
*********************************************************/


typedef struct {
  int nodeid;     // id for node to transform
  const char *text;     // name of transformation (will appear on button for transformation)
} TRANSFORMATION;

// Number of transformation schemes
const unsigned int NUMTRANS = 210;

extern TRANSFORMATION TransformationTable[NUMTRANS];

/*********************************************************
* Prototype.
```

```
* This function returns a pointer to a tree, which
* is supposed to be inserted into the root tree
* at a place where a placeholder is selected.
* It is a part of doing transformations.
* The actual function is defined in nodeinfo.cpp
* Input:
* int - a number which identifier the transformation
* confer with TransformationTable.
* Output:
* PSYNTAXTREE - a pointer to a tree
* Side Effects:
*
*********************************************************/
extern PSYNTAXTREE MakeIt(const unsigned int);


#endif
```

F.4 MYSTRING.H

```
/*************************************************************************
*
* MYSTRING.H
*
* Specification of my string routines
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef MYSTRING_H
#define MYSTRING_H

char *my_strcpy(char *,const char *);
char *my_strncpy(char *,const char *,int);

int my_strcmp(const char *,const char *);
int my_strcmpi(const char *,const char *);

unsigned int my_strlen(const char *);

#endif
```

F.5 NARYTREE.H

```
/*************************************************************************
*
* NARYTREE.H
*
* Specification of NaryTree Class
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef NARYTREE_H
#define NARYTREE_H

#include "DEF.H"
```

```
#include "SYNNODE.H"

// Class prototype
class NaryTree;

typedef NaryTree* PNARYTREE;

// Class Specification
class NaryTree: public SyntaxTreeNode {
  PNARYTREE parent;
  PNARYTREE firstchild;
  PNARYTREE lastchild;
  PNARYTREE prevsibling;
  PNARYTREE nextsibling;
public:
  NaryTree();

  void null(void) {
    parent=firstchild=lastchild=prevsibling=nextsibling=NULL;
  }

  void insert_tree_first(PNARYTREE);
  void insert_tree_last(PNARYTREE);
  void insert_tree_before(PNARYTREE);
  void insert_tree_after(PNARYTREE);

  void delete_subtree(PNARYTREE);
  void delete_treenode(PNARYTREE);

  void cut_subtree(void);
  void paste_subtree(const PNARYTREE);

  BOOL compare_trees(PNARYTREE,PNARYTREE) const;
  virtual BOOL compare_nodes(PNARYTREE,PNARYTREE) const;

  PNARYTREE get_parent(void) const { return parent; }
  PNARYTREE get_firstchild(void) const { return firstchild; }
  PNARYTREE get_lastchild(void) const { return lastchild; }
  PNARYTREE get_prevsibling(void) const { return prevsibling; }
  PNARYTREE get_nextsibling(void) const { return nextsibling; }

  unsigned int count_childs(PNARYTREE) const;

};

#endif
```

F.6 NODEIDS.H

```
/*************************************************************************
*
* NODEIDS.H
*
* Node Identifiers Definitions
*
* Created 11 January, 1994, Michael Suodenjoki
*
*************************************************************************/

#ifndef NODEIDS_H
#define NODEIDS_H

// Node Identifiers in Abstract Syntax Tree
```

```
#define NID_IDNULL                    0
#define NID_ID                        1
#define NID_IDLIST2                   2
#define NID_IDENTIFIER                3
#define NID_INTEGERNULL               4
#define NID_OPTCOMMENTSTRING          5

#define NID_SPECIFICATION             6
#define NID_MODULEDECLSTRING          7
#define NID_MODULEDECL                8
#define NID_MODULEDECLNULL            9
#define NID_DECL                      10
#define NID_OPTDECLSTRING             11
#define NID_DECLNULL                  12
#define NID_SCHEMEDECL                13
#define NID_SCHEMEDEF                 14
#define NID_SCHEMEDEFLIST             15
#define NID_FORMALSCHEMEPARAMETER     16
#define NID_OPTFORMALSCHEMEPARAMETER  17
#define NID_OBJECTDECL                18
#define NID_OBJECTDEF                 19
#define NID_OBJECTDEFLIST             20
#define NID_FORMALARRAYPARAMETER      21
#define NID_OPTFORMALARRAYPARAMETER   22
#define NID_TYPEDECL                  23
#define NID_TYPEDEF                   24
#define NID_TYPEDEFNULL               25
#define NID_TYPEDEFLIST               26
#define NID_SORTDEF                   27
#define NID_VARIANTDEF                28
#define NID_VARIANT                   29
#define NID_VARIANTNULL               30
#define NID_VARIANTCHOICE             31
#define NID_RECORDVARIANT             32
#define NID_COMPONENTKIND             33
#define NID_COMPONENTKINDLIST         34
#define NID_COMPONENTKINDSTRING       35
#define NID_CONSTRUCTOR               36
#define NID_CONSTRUCTORNULL           37
#define NID_DESTRUCTOR                38
#define NID_OPTDESTRUCTOR             39
#define NID_RECONSTRUCTOR             40
#define NID_OPTRECONSTRUCTOR          41
#define NID_UNIONDEF                  42
#define NID_NAMEORWILDCARD            43
#define NID_NAMEORWILDCARDNULL        44
#define NID_NAMEORWILDCARDCHOICE2     45
#define NID_SHORTRECORDDEF            46
#define NID_ABBREVIATIONDEF           47
#define NID_VALUEDECL                 48
#define NID_VALUEDEF                  49
#define NID_VALUEDEFNULL              50
#define NID_VALUEDEFLIST              51
#define NID_EXPLICITVALUEDEF          52
#define NID_IMPLICITVALUEDEF          53
#define NID_EXPLICITFUNCTIONDEF       54
#define NID_FORMALFUNCTIONAPPLICATION 55
#define NID_FORMALFUNCTIONAPPLICATIONNULL 56
#define NID_IDAPPLICATION             57
#define NID_FORMALFUNCTIONPARAMETER   58
#define NID_FORMALFUNCTIONPARAMETERSTRING 59
#define NID_PREFIXAPPLICATION         60
#define NID_INFIXAPPLICATION          61
#define NID_IMPLICITFUNCTIONDEF       62
#define NID_VARIABLEDECL              63
#define NID_VARIABLEDEF               64
#define NID_VARIABLEDEFNULL           65
```

```
#define NID_VARIABLEDEFLIST 66
#define NID_SINGLEVARIABLEDEF 67
#define NID_INITIALISATION 68
#define NID_OPTINITIALISATION 69
#define NID_MULTIPLEVARIABLEDEF 70
#define NID_CHANNELDECL 71
#define NID_CHANNELDEF 72
#define NID_CHANNELDEFNULL 73
#define NID_CHANNELDEFLIST 74
#define NID_SINGLECHANNELDEF 75
#define NID_MULTIPLECHANNELDEF 76
#define NID_AXIOMDECL 77
#define NID_AXIOMQUANTIFICATION 78
#define NID_OPTAXIOMQUANTIFICATION 79
#define NID_AXIOMDEF 80
#define NID_AXIOMDEFLIST 81
#define NID_AXIOMNAMING 82
#define NID_OPTAXIOMNAMING 83
#define NID_CLASSEXPR 84
#define NID_CLASSEXPRNULL 85
#define NID_BASICCLASSEXPR 86
#define NID_EXTENDINGCLASSEXPR 87
#define NID_HIDINGCLASSEXPR 88
#define NID_RENAMINGCLASSEXPR 89
#define NID_SCHEMEINSTANTIATION 90
#define NID_ACTUALSCHEMEPARAMETER 91
#define NID_OPTACTUALSCHEMEPARAMETER 92
#define NID_RENAMEPAIR 93
#define NID_RENAMEPAIRLIST 94
#define NID_DEFINEDITEM 95
#define NID_DEFINEDITEMNULL 96
#define NID_DEFINEDITEMLIST 97
#define NID_DISAMBIGUATEDITEM 98
#define NID_OBJECTEXPR 99
#define NID_OBJECTEXPRNULL 100
#define NID_OBJECTEXPRLIST 101
#define NID_ELEMENTOBJECTEXPR 102
#define NID_ACTUALARRAYPARAMETER 103
#define NID_ARRAYOBJECTEXPR 104
#define NID_FITTINGOBJECTEXPR 105
#define NID_TYPEEXPR 106
#define NID_TYPEEXPRNULL 107
#define NID_TYPEEXPRPRODUCT2 108
#define NID_TYPELITERAL 109
#define NID_TYPELITERALNULL 110
#define NID_PRODUCTTYPEEXPR 111
#define NID_SETTYPEEXPR 112
#define NID_SETTYPEEXPRNULL 113
#define NID_FINITESETTYPEEXPR 114
#define NID_INFINITESETTYPEEXPR 115
#define NID_LISTTYPEEXPR 116
#define NID_LISTTYPEEXPRNULL 117
#define NID_FINITELISTTYPEEXPR 118
#define NID_INFINITELISTTYPEEXPR 119
#define NID_MAPTYPEEXPR 120
#define NID_FUNCTIONTYPEEXPR 121
#define NID_FUNCTIONARROW 122
#define NID_FUNCTIONARROWNULL 123
#define NID_RESULTDESC 124
#define NID_SUBTYPEEXPR 125
#define NID_BRACKETEDTYPEEXPR 126
#define NID_ACCESSDESC 127
#define NID_OPTACCESSDESCSTRING 128
#define NID_ACCESSMODE 129
#define NID_ACCESSMODENULL 130
#define NID_ACCESS 131
#define NID_ACCESSNULL 132
```

```
#define NID_ACCESSLIST 133
#define NID_OPTACCESSLIST 134
#define NID_ENUMERATEDACCESS 135
#define NID_COMPLETEDACCESS 136
#define NID_COMPREHENDEDACCESS 137
#define NID_VALUEEXPR 138
#define NID_VALUEEXPRNULL 139
#define NID_VALUEEXPRLIST 140
#define NID_OPTVALUEEXPRLIST 141
#define NID_VALUEEXPRLIST2 142
#define NID_VALUELITERAL 143
#define NID_VALUELITERALNULL 144
#define NID_UNITLITERAL 145
#define NID_BOOLLITERAL 146
#define NID_BOOLLITERALNULL 147
#define NID_PRENAME 148
#define NID_BASICEXPR 149
#define NID_BASICEXPRNULL 150
#define NID_PRODUCTEXPR 151
#define NID_SETEXPR 152
#define NID_SETEXPRNULL 153
#define NID_RANGEDSETEXPR 154
#define NID_ENUMERATEDSETEXPR 155
#define NID_COMPREHENDEDSETEXPR 156
#define NID_SETLIMITATION 157
#define NID_RESTRICTION 158
#define NID_OPTRESTRICTION 159
#define NID_LISTEXPR 160
#define NID_LISTEXPRNULL 161
#define NID_RANGEDLISTEXPR 162
#define NID_ENUMERATEDLISTEXPR 163
#define NID_COMPREHENDEDLISTEXPR 164
#define NID_LISTLIMITATION 165
#define NID_MAPEXPR 166
#define NID_MAPEXPRNULL 167
#define NID_ENUMERATEDMAPEXPR 168
#define NID_VALUEEXPRPAIR 169
#define NID_OPTVALUEEXPRPAIRLIST 170
#define NID_COMPREHENDEDMAPEXPR 171
#define NID_FUNCTIONEXPR 172
#define NID_LAMBDAPARAMETER 173
#define NID_LAMBDAPARAMETERNULL 174
#define NID_LAMBDATYPING 175
#define NID_APPLICATIONEXPR 176
#define NID_ACTUALFUNCTIONPARAMETER 177
#define NID_ACTUALFUNCTIONPARAMETERSTRING 178
#define NID_QUANTIFIEDEXPR 179
#define NID_QUANTIFIER 180
#define NID_QUANTIFIERNULL 181
#define NID_EQUIVALENCEEXPR 182
#define NID_PRECONDITION 183
#define NID_OPTPRECONDITION 184
#define NID_POSTEXPR 185
#define NID_POSTCONDITION 186
#define NID_RESULTNAMING 187
#define NID_OPTRESULTNAMING 188
#define NID_DISAMBIGUATEDEXPR 189
#define NID_BRACKETEDEXPR 190
#define NID_INFIXEXPR 191
#define NID_INFIXEXPRNULL 192
#define NID_STMTINFIXEXPR 193
#define NID_AXIOMINFIXEXPR 194
#define NID_VALUEINFIXEXPR 195
#define NID_PREFIXEXPR 196
#define NID_PREFIXEXPRNULL 197
#define NID_AXIOMPREFIXEXPR 198
#define NID_UNIVERSALPREFIXEXPR 199
```

```
#define NID_VALUEPREFIXEXPR 200
#define NID_COMPREHENDEDEXPR 201
#define NID_INITIALISEEXPR 202
#define NID_ASSIGNMENTEXPR 203
#define NID_INPUTEXPR 204
#define NID_OUTPUTEXPR 205
#define NID_STRUCTUREDEXPR 206
#define NID_STRUCTUREDEXPRNULL 207
#define NID_LOCALEXPR 208
#define NID_LETEXPR 209
#define NID_LETDEF 210
#define NID_LETDEFNULL 211
#define NID_LETDEFLIST 212
#define NID_EXPLICITLET 213
#define NID_IMPLICITLET 214
#define NID_LETBINDING 215
#define NID_IFEXPR 216
#define NID_ELSIFBRANCH 217
#define NID_OPTELSIFBRANCHSTRING 218
#define NID_ELSEBRANCH 219
#define NID_OPTELSEBRANCH 220
#define NID_CASEEXPR 221
#define NID_CASEBRANCH 222
#define NID_CASEBRANCHLIST 223
#define NID_WHILEEXPR 224
#define NID_UNTILEXPR 225
#define NID_FOREXPR 226
#define NID_BINDING 227
#define NID_BINDINGNULL 228
#define NID_BINDINGLIST2 229
#define NID_OPTBINDINGLIST 230
#define NID_PRODUCTBINDING 231
#define NID_TYPING 232
#define NID_TYPINGNULL 233
#define NID_TYPINGLIST 234
#define NID_OPTTYPINGLIST 235
#define NID_SINGLETYPING 236
#define NID_MULTIPLETYPING 237
#define NID_COMMENTEDTYPING 238
#define NID_PATTERN 239
#define NID_PATTERNNULL 240
#define NID_WILDCARDPATTERN 241
#define NID_PRODUCTPATTERN 242
#define NID_RECORDPATTERN 243
#define NID_LISTPATTERN 244
#define NID_LISTPATTERNNULL 245
#define NID_ENUMERATEDLISTPATTERN 246
#define NID_INNERPATTERN 247
#define NID_INNERPATTERNNULL 248
#define NID_INNERPATTERNLIST 249
#define NID_OPTINNERPATTERNLIST 250
#define NID_INNERPATTERNLIST2 251
#define NID_EQUALITYPATTERN 252
#define NID_NAME 253
#define NID_NAMENULL 254
#define NID_QUALIFIEDID 255
#define NID_QUALIFICATION 256
#define NID_OPTQUALIFICATION 257
#define NID_QUALIFIEDOP 258
#define NID_IDOROP 259
#define NID_IDOROPNULL 260
#define NID_OP 261
#define NID_OPNULL 262
#define NID_INFIXOP 263
#define NID_INFIXOPNULL 264
#define NID_PREFIXOP 265
#define NID_PREFIXOPNULL 266
```

```
#define NID_CONNECTIVE 267
#define NID_CONNECTIVENULL 268
#define NID_INFIXCONNECTIVE 269
#define NID_INFIXCONNECTIVENULL 270
#define NID_PREFIXCONNECTIVE 271
#define NID_INFIXCOMBINATOR 272
#define NID_INFIXCOMBINATORNULL 273

// Terminals
#define NID_WILDCARD 274
#define NID_TUNIT 275
#define NID_TBOOL 276
#define NID_TINT 277
#define NID_TNAT 278
#define NID_TREAL 279
#define NID_TTEXT 280
#define NID_TCHAR 281
#define NID_PARRIGHTARROW 282
#define NID_RIGHTARROW 283
#define NID_READ 284
#define NID_WRITE 285
#define NID_IN 286
#define NID_OUT 287
#define NID_TRUE 288
#define NID_FALSE 289
#define NID_CHAOS 290
#define NID_SKIP 291
#define NID_STOP 292
#define NID_SWAP 293
#define NID_FORALL 294
#define NID_EXISTS 295
#define NID_EXISTSUNIQUE 296
#define NID_EQUAL 297
#define NID_NOTEQUAL 298
#define NID_GT 299
#define NID_LT 300
#define NID_GE 301
#define NID_LE 302
#define NID_SUPERSET 303
#define NID_SUBSET 304
#define NID_PROPERSUPERSET 305
#define NID_PROPERSUBSET 306
#define NID_ISIN 307
#define NID_NOTISIN 308
#define NID_ADDITION 309
#define NID_SUBTRACTION 310
#define NID_BACKSLASH 311
#define NID_CONCAT 312
#define NID_UNION 313
#define NID_OVERRIDE 314
#define NID_MULTIPLICATION 315
#define NID_DIVISION 316
#define NID_COMPOSITION 317
#define NID_INTERSECTION 318
#define NID_EXPONENTIATION 319
#define NID_ABS 320
#define NID_INT 321
#define NID_REAL 322
#define NID_CARD 323
#define NID_LEN 324
#define NID_INDS 325
#define NID_ELEMS 326
#define NID_HD 327
#define NID_TL 328
#define NID_DOM 329
#define NID_RNG 330
#define NID_IMPLICATION 331
```

```
#define NID_OR                          332
#define NID_AND                         333
#define NID_NOT                         334
#define NID_EXTERNALCHOICE              335
#define NID_INTERNALCHOICE              336
#define NID_CONCURRENTCOMPOSITION 337
#define NID_INTERLOCKEDCOMPOSITION 338
#define NID_SEQUENTIALCOMPOSITION 339

// Forgets !
#define NID_CONCATENATEDLISTPATTERN 340
#define NID_LETBINDINGNULL              341
#define NID_COMMENT                     342
#define NID_INTLITERAL                  343
#define NID_REALLITERAL                 344
#define NID_TEXTLITERAL                 345
#define NID_CHARLITERAL                 346

#define NID_COMMENTNULL                 347
#define NID_COMMENTSTRING               348
#define NID_ELSIFBRANCHNULL             349
#define NID_ELSIFBRANCHSTRING           350
#define NID_VALUEEXPRPAIRNULL           351
#define NID_VALUEEXPRPAIRLIST           352
#define NID_ACCESSDESCNULL              353
#define NID_ACCESSDESCSTRING            354
#define NID_DECLSTRING                  355

#define NID_BINDINGLIST                 356
#define NID_RIGHTLISTPATTERN            357
#define NID_COMMENTTOKENNULL            358
#define NID_COMMENTTOKEN                359
#define NID_COMMENTTOKENSTRING          360
#define NID_COMMENTSPACE                361
#define NID_COMMENTNEWLINE              362
#define NID_COMMENTTEXT                 363

#endif /* NODEIDS_H */
```

F.7 RSL.H

```
/*****************************************************************************
*
* RSL.H
* Specification, Definition of the RAISE Specification Language
*
* Created 15 November, 1993, Michael Suodenjoki
* Recreated 22 December, 1993, Michael Suodenjoki
*
*****************************************************************************/

// Changed:
// 10/12-93 - Further classes added plus new nodeid definitions
//

#ifndef RSL_H
#define RSL_H

#include "SYNTREE.H"
#include "LANGUAGE.H"

/***************** Class Definitions ********************/
```

```
/* Class defintion for a placeholder e.g. IdNull */
#define NullCl(x,y) \
class x ## Null: public SyntaxTree { \
public: \
  x ## Null (); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for variant nodes */
/* a ::= b1 | b2 | b3 | ... */
#define VarCl(x,y) \
NullCl(x,y) \
\
class x : public SyntaxTree { \
public: \
  x (); \
  void make_subtree(void); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for list nodes */
/* a ::= b | a "," b */
#define ListCl(x,y) \
class x ## List: public SyntaxTree { \
public: \
  x ## List (); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for list nodes */
/* a ::= b | a "," b */
#define ListClRest(x,y) \
class x ## List: public SyntaxTree { \
public: \
  x ## List (); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};


/* Class definition for choice nodes (also a list) */
/* a ::= b | a "|" b */
#define ChoiceCl(x,y) \
class x ## Choice: public SyntaxTree { \
public: \
  x ## Choice();\
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for list2 nodes */
/* a ::= b "," b | a "," b */
#define List2Cl(x,y) \
class x ## List2: public SyntaxTree { \
public: \
  x ## List2(); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for string (lists) nodes */
/* a ::= b | a " " b */
```

```
#define StringCl(x,y) \
class x ## String: public SyntaxTree { \
public: \
  x ## String (); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for optional nodes */
/* a ::= | b */
#define Optional(x,y) \
class Opt ## x : public SyntaxTree { \
public: \
  Opt ## x (); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for terminal nodes */
/* a ::= b */
#define Terminal(x,y) \
class x : public SyntaxTree { \
public: \
  x (); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for non-terminal nodes */
/* a ::= b1 b2 b3 ... */
#define Class(x,y) \
class x : public SyntaxTree { \
public: \
  x (); \
  void make_subtree(void); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for non-terminal nodes */
/* Normally these are non-resting-places, */
/* but can be maked resting places by */
/* using this class. */
#define ClassRest(x,y) \
class x : public SyntaxTree { \
public: \
  x (); \
  void make_subtree(void); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for optional strings */
/* a1 ::= | a2 */
/* a2 ::= b | a2 " " b */
#define OptString(x,y) \
StringCl(x,y) \
\
class Opt ## x ## String: public SyntaxTree { \
public: \
  Opt ## x ## String(); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for optional lists */
/* a1 ::= | a2 */
/* a2 ::= b | a2 "," b */
#define OptList(x,y) \
```

```
ListCl(x,y) \
\
class Opt ## x ## List: public SyntaxTree { \
public: \                                                              160
  Opt ## x ## List(); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for choice2 lists */
/* a ::= b "|" b | a "|" b */
#define Choice2Cl(x,y) \
class x ## Choice2: public SyntaxTree { \
public: \                                                              170
  x ## Choice2(); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for product2 lists */
/* a ::= b "x" b | a "x" b */
#define Product2Cl(x,y) \
class x ## Product2: public SyntaxTree { \                             180
public: \
  x ## Product2(); \
  void make_subtree(void); \
  PSYNTAXTREE make_child(); \
  PSYNTAXTREE clone(void) const; \
};

/* Class definition for lexical terminals */
/* e.g. identifier, integers and reals */
#define LexTerminal(x,y) \                                             190
class x : public SyntaxTree { \
private: \
  char str[IDLENGTH]; \
public: \
  x () {} \
  x (char *); \
  PSYNTAXTREE clone(void) const; \
  virtual char *unparse_special(void) const; \
  virtual BOOL compare(const PSYNNODE,const PSYNNODE) const; \
};                                                                     200


/* Definition of class for the language */

LexTerminal(Identifier,IDENTIFIER)
LexTerminal(CommentText,COMMENTTEXT)

VarCl(Id,ID)
List2Cl(Id,ID)
                                                                       210
Terminal(CommentNewline,COMMENTNEWLINE)
Terminal(CommentSpace,COMMENTSPACE)
VarCl(CommentToken,COMMENTTOKEN)
StringCl(CommentToken,COMMENTTOKEN)

VarCl(Comment,COMMENT)

OptString(Comment,COMMENT)

Terminal(Wildcard,WILDCARD)                                            220
Terminal(TUnit,TUNIT)
Terminal(TBool,TBOOL)
Terminal(TInt,TINT)
```

```
Terminal(TNat,TNAT)
Terminal(TReal,TREAL)
Terminal(TText,TTEXT)
Terminal(TChar,TCHAR)
Terminal(ParRightArrow,PARRIGHTARROW)
Terminal(RightArrow,RIGHTARROW)
Terminal(Read,READ)                                                    230
Terminal(Write,WRITE)
Terminal(In,IN)
Terminal(Out,OUT)

LexTerminal(IntLiteral,INTLITERAL)
LexTerminal(RealLiteral,REALLITERAL)
LexTerminal(TextLiteral,TEXTLITERAL)
LexTerminal(CharLiteral,CHARLITERAL)

Terminal(True,TRUE)                                                    240
Terminal(False,FALSE)
Terminal(Chaos,CHAOS)
Terminal(Skip,SKIP)
Terminal(Stop,STOP)
Terminal(Swap,SWAP)
Terminal(Forall,FORALL)
Terminal(Exists,EXISTS)
Terminal(ExistsUnique,EXISTSUNIQUE)
// InfixOp terminals
Terminal(Equal,EQUAL)                                                  250
Terminal(NotEqual,NOTEQUAL)
Terminal(Gt,GT)
Terminal(Lt,LT)
Terminal(GE,GE)
Terminal(LE,LE)
Terminal(SuperSet,SUPERSET)
Terminal(Subset,SUBSET)
Terminal(ProperSuperset,PROPERSUPERSET)
Terminal(ProperSubset,PROPERSUBSET)
Terminal(IsIn,ISIN)                                                    260
Terminal(NotIsIn,NOTISIN)
Terminal(Plus,ADDITION)
Terminal(Minus,SUBTRACTION)
Terminal(Backslash,BACKSLASH)
Terminal(Concat,CONCAT)
Terminal(Union,UNION)
Terminal(Override,OVERRIDE)
Terminal(Mult,MULTIPLICATION)
Terminal(Divide,DIVISION)
Terminal(Composition,COMPOSITION)                                      270
Terminal(Inter,INTERSECTION)
Terminal(Exp,EXPONENTIATION)
Terminal(Abs,ABS)
Terminal(Int,INT)
Terminal(Real,REAL)
Terminal(Card,CARD)
Terminal(Len,LEN)
Terminal(Inds,INDS)
Terminal(Elems,ELEMS)
Terminal(Hd,HD)                                                        280
Terminal(Tl,TL)
Terminal(Dom,DOM)
Terminal(Rng,RNG)

Terminal(Not,NOT)
Terminal(Implication,IMPLICATION)
Terminal(Or,OR)
Terminal(And,AND)
Terminal(ExtChoice,EXTERNALCHOICE)
Terminal(IntChoice,INTERNALCHOICE)                                     290
```

Terminal(*ConComposition*, *CONCURRENTCOMPOSITION*)
Terminal(*IntComposition*, *INTERLOCKEDCOMPOSITION*)
Terminal(*SeqComposition*, *SEQUENTIALCOMPOSITION*)

VarCl(*InfixCombinator*, *INFIXCOMBINATOR*)
Terminal(*PrefixConnective*, *PREFIXCONNECTIVE*)
VarCl(*InfixConnective*, *INFIXCONNECTIVE*)
VarCl(*Connective*, *CONNECTIVE*)
VarCl(*PrefixOp*, *PREFIXOP*)
VarCl(*InfixOp*, *INFIXOP*)
VarCl(*Op*, *OP*)
VarCl(*IdOrOp*, *IDOROP*)
Class(*QualifiedOp*, *QUALIFIEDOP*)
Class(*Qualification*, *QUALIFICATION*)
Optional(*Qualification*, *QUALIFICATION*)
Class(*QualifiedId*, *QUALIFIEDID*)
VarCl(*Name*, *NAME*)
Class(*EqualityPattern*, *EQUALITYPATTERN*)
VarCl(*InnerPattern*, *INNERPATTERN*)
ListCl(*InnerPattern*, *INNERPATTERN*)
Optional(*InnerPatternList*, *INNERPATTERNLIST*)
List2Cl(*InnerPattern*, *INNERPATTERN*)
Class(*ConcatenatedListPattern*, *CONCATENATEDLISTPATTERN*)
Class(*EnumeratedListPattern*, *ENUMERATEDLISTPATTERN*)
Class(*RightListPattern*, *RIGHTLISTPATTERN*)
VarCl(*ListPattern*, *LISTPATTERN*)
Class(*RecordPattern*, *RECORDPATTERN*)
Class(*ProductPattern*, *PRODUCTPATTERN*)
Terminal(*WildcardPattern*, *WILDCARDPATTERN*)
VarCl(*Pattern*, *PATTERN*)
Class(*CommentedTyping*, *COMMENTEDTYPING*)
Class(*MultipleTyping*, *MULTIPLETYPING*)
Class(*SingleTyping*, *SINGLETYPING*)
VarCl(*Typing*, *TYPING*)
ListCl(*Typing*, *TYPING*)
Optional(*TypingList*, *TYPINGLIST*)
Class(*ProductBinding*, *PRODUCTBINDING*)
VarCl(*Binding*, *BINDING*)
List2Cl(*Binding*, *BINDING*)
OptList(*Binding*, *BINDING*)
Class(*ForExpr*, *FOREXPR*)
Class(*UntilExpr*, *UNTILEXPR*)
Class(*WhileExpr*, *WHILEEXPR*)
Class(*CaseBranch*, *CASEBRANCH*)
ListCl(*CaseBranch*, *CASEBRANCH*)
Class(*CaseExpr*, *CASEEXPR*)
Class(*ElseBranch*, *ELSEBRANCH*)
Optional(*ElseBranch*, *ELSEBRANCH*)
Class(*ElsifBranch*, *ELSIFBRANCH*)
NullCl(*ElsifBranch*, *ELSIFBRANCH*)
OptString(*ElsifBranch*, *ELSIFBRANCH*)
Class(*IfExpr*, *IFEXPR*)
VarCl(*LetBinding*, *LETBINDING*)
Class(*ImplicitLet*, *IMPLICITLET*)
Class(*ExplicitLet*, *EXPLICITLET*)
VarCl(*LetDef*, *LETDEF*)
ListCl(*LetDef*, *LETDEF*)
Class(*LetExpr*, *LETEXPR*)
Class(*LocalExpr*, *LOCALEXPR*)
VarCl(*StructuredExpr*, *STRUCTUREDEXPR*)
Class(*OutputExpr*, *OUTPUTEXPR*)
Class(*InputExpr*, *INPUTEXPR*)
Class(*AssignmentExpr*, *ASSIGNMENTEXPR*)
Class(*InitialiseExpr*, *INITIALISEEXPR*)
Class(*ComprehendedExpr*, *COMPREHENDEDEXPR*)
Class(*ValuePrefixExpr*, *VALUEPREFIXEXPR*)
Class(*UniversalPrefixExpr*, *UNIVERSALPREFIXEXPR*)

Class(*AxiomPrefixExpr*,*AXIOMPREFIXEXPR*)
VarCl(*PrefixExpr*,*PREFIXEXPR*)
Class(*ValueInfixExpr*,*VALUEINFIXEXPR*)
Class(*AxiomInfixExpr*,*AXIOMINFIXEXPR*)
Class(*StmtInfixExpr*,*STMTINFIXEXPR*)
VarCl(*InfixExpr*,*INFIXEXPR*)
Class(*BracketedExpr*,*BRACKETEDEXPR*)
Class(*DisambiguatedExpr*,*DISAMBIGUATEDEXPR*)
Class(*ResultNaming*,*RESULTNAMING*)
Optional(*ResultNaming*,*RESULTNAMING*)
Class(*PostCondition*,*POSTCONDITION*)
Class(*PostExpr*,*POSTEXPR*)
Class(*PreCondition*,*PRECONDITION*)
Optional(*PreCondition*,*PRECONDITION*)
Class(*EquivalenceExpr*,*EQUIVALENCEEXPR*)
VarCl(*Quantifier*,*QUANTIFIER*)
Class(*QuantifiedExpr*,*QUANTIFIEDEXPR*)
Class(*ActualFunctionParameter*,*ACTUALFUNCTIONPARAMETER*)
StringCl(*ActualFunctionParameter*,*ACTUALFUNCTIONPARAMETER*)
Class(*ApplicationExpr*,*APPLICATIONEXPR*)
Class(*LambdaTyping*,*LAMBDATYPING*)
VarCl(*LambdaParameter*,*LAMBDAPARAMETER*)
Class(*FunctionExpr*,*FUNCTIONEXPR*)
Class(*ComprehendedMapExpr*,*COMPREHENDEDMAPEXPR*)
Class(*ValueExprPair*,*VALUEEXPRPAIR*)
NullCl(*ValueExprPair*,*VALUEEXPRPAIR*)
OptList(*ValueExprPair*,*VALUEEXPRPAIR*)
Class(*EnumeratedMapExpr*,*ENUMERATEDMAPEXPR*)
VarCl(*MapExpr*,*MAPEXPR*)
Class(*ListLimitation*,*LISTLIMITATION*)
Class(*ComprehendedListExpr*,*COMPREHENDEDLISTEXPR*)
Class(*EnumeratedListExpr*,*ENUMERATEDLISTEXPR*)
Class(*RangedListExpr*,*RANGEDLISTEXPR*)
VarCl(*ListExpr*,*LISTEXPR*)
Class(*Restriction*,*RESTRICTION*)
Optional(*Restriction*,*RESTRICTION*)
Class(*SetLimitation*,*SETLIMITATION*)
Class(*ComprehendedSetExpr*,*COMPREHENDEDSETEXPR*)
Class(*EnumeratedSetExpr*,*ENUMERATEDSETEXPR*)
Class(*RangedSetExpr*,*RANGEDSETEXPR*)
VarCl(*SetExpr*,*SETEXPR*)
Class(*ProductExpr*,*PRODUCTEXPR*)
VarCl(*BasicExpr*,*BASICEXPR*)
Class(*PreName*,*PRENAME*)
VarCl(*BoolLiteral*,*BOOLLITERAL*)
Terminal(*UnitLiteral*,*UNITLITERAL*)
VarCl(*ValueLiteral*,*VALUELITERAL*)
VarCl(*ValueExpr*,*VALUEEXPR*)
ListCl(*ValueExpr*,*VALUEEXPR*)
List2Cl(*ValueExpr*,*VALUEEXPR*)
Optional(*ValueExprList*,*VALUEEXPRLIST*)
Class(*ComprehendedAccess*,*COMPREHENDEDACCESS*)
Class(*CompletedAccess*,*COMPLETEDACCESS*)
Class(*EnumeratedAccess*,*ENUMERATEDACCESS*)
VarCl(*Access*,*ACCESS*)
ListCl(*Access*,*ACCESS*)
Optional(*AccessList*,*ACCESSLIST*)
VarCl(*AccessMode*,*ACCESSMODE*)
Class(*AccessDesc*,*ACCESSDESC*)
NullCl(*AccessDesc*,*ACCESSDESC*)
OptString(*AccessDesc*,*ACCESSDESC*)
Class(*BracketedTypeExpr*,*BRACKETEDTYPEEXPR*)
Class(*SubTypeExpr*,*SUBTYPEEXPR*)
Class(*ResultDesc*,*RESULTDESC*)
VarCl(*FunctionArrow*,*FUNCTIONARROW*)
Class(*FunctionTypeExpr*,*FUNCTIONTYPEEXPR*)
Class(*MapTypeExpr*,*MAPTYPEEXPR*)

Class(*InfiniteListTypeExpr*,*INFINITELISTTYPEEXPR*)
Class(*FiniteListTypeExpr*,*FINITELISTTYPEEXPR*)
VarCl(*ListTypeExpr*,*LISTTYPEEXPR*)
Class(*InfiniteSetTypeExpr*,*INFINITESETTYPEEXPR*)
Class(*FiniteSetTypeExpr*,*FINITESETTYPEEXPR*)
VarCl(*SetTypeExpr*,*SETTYPEEXPR*)
Class(*ProductTypeExpr*,*PRODUCTTYPEEXPR*)
VarCl(*TypeLiteral*,*TYPELITERAL*)
VarCl(*TypeExpr*,*TYPEEXPR*)
Product2Cl(*TypeExpr*,*TYPEEXPR*)
Class(*FittingObjectExpr*,*FITTINGOBJECTEXPR*)
Class(*ArrayObjectExpr*,*ARRAYOBJECTEXPR*)
Class(*ActualArrayParameter*,*ACTUALARRAYPARAMETER*)
Class(*ElementObjectExpr*,*ELEMENTOBJECTEXPR*)
VarCl(*ObjectExpr*,*OBJECTEXPR*)
ListCl(*ObjectExpr*,*OBJECTEXPR*)
Class(*DisambiguatedItem*,*DISAMBIGUATEDITEM*)
VarCl(*DefinedItem*,*DEFINEDITEM*)
ListCl(*DefinedItem*,*DEFINEDITEM*)
Class(*RenamePair*,*RENAMEPAIR*)
ListCl(*RenamePair*,*RENAMEPAIR*)
Class(*ActualSchemeParameter*,*ACTUALSCHEMEPARAMETER*)
Optional(*ActualSchemeParameter*,*ACTUALSCHEMEPARAMETER*)
Class(*SchemeInstantiation*,*SCHEMEINSTANTIATION*)
Class(*RenamingClassExpr*,*RENAMINGCLASSEXPR*)
Class(*HidingClassExpr*,*HIDINGCLASSEXPR*)
Class(*ExtendingClassExpr*,*EXTENDINGCLASSEXPR*)
Class(*BasicClassExpr*,*BASICCLASSEXPR*)
VarCl(*ClassExpr*,*CLASSEXPR*)
Class(*AxiomNaming*,*AXIOMNAMING*)
Optional(*AxiomNaming*,*AXIOMNAMING*)
ClassRest(*AxiomDef*,*AXIOMDEF*)
ListCl(*AxiomDef*,*AXIOMDEF*)
Class(*AxiomQuantification*,*AXIOMQUANTIFICATION*)
Optional(*AxiomQuantification*,*AXIOMQUANTIFICATION*)
Class(*AxiomDecl*,*AXIOMDECL*)
Class(*MultipleChannelDef*,*MULTIPLECHANNELDEF*)
Class(*SingleChannelDef*,*SINGLECHANNELDEF*)
VarCl(*ChannelDef*,*CHANNELDEF*)
ListCl(*ChannelDef*,*CHANNELDEF*)
Class(*ChannelDecl*,*CHANNELDECL*)
Class(*MultipleVariableDef*,*MULTIPLEVARIABLEDEF*)
Class(*Initialisation*,*INITIALISATION*)
Optional(*Initialisation*,*INITIALISATION*)
Class(*SingleVariableDef*,*SINGLEVARIABLEDEF*)
VarCl(*VariableDef*,*VARIABLEDEF*)
ListCl(*VariableDef*,*VARIABLEDEF*)
Class(*VariableDecl*,*VARIABLEDECL*)
Class(*ImplicitFunctionDef*,*IMPLICITFUNCTIONDEF*)
Class(*InfixApplication*,*INFIXAPPLICATION*)
Class(*PrefixApplication*,*PREFIXAPPLICATION*)
Class(*FormalFunctionParameter*,*FORMALFUNCTIONPARAMETER*)
StringCl(*FormalFunctionParameter*,*FORMALFUNCTIONPARAMETER*)
Class(*IdApplication*,*IDAPPLICATION*)
VarCl(*FormalFunctionApplication*,*FORMALFUNCTIONAPPLICATION*)
Class(*ExplicitFunctionDef*,*EXPLICITFUNCTIONDEF*)
Class(*ImplicitValueDef*,*IMPLICITVALUEDEF*)
Class(*ExplicitValueDef*,*EXPLICITVALUEDEF*)
VarCl(*ValueDef*,*VALUEDEF*)
ListCl(*ValueDef*,*VALUEDEF*)
Class(*ValueDecl*,*VALUEDECL*)
Class(*AbbreviationDef*,*ABBREVIATIONDEF*)
Class(*ShortRecordDef*,*SHORTRECORDDEF*)
VarCl(*NameOrWildcard*,*NAMEORWILDCARD*)
Choice2Cl(*NameOrWildcard*,*NAMEORWILDCARD*)
Class(*UnionDef*,*UNIONDEF*)
Class(*Reconstructor*,*RECONSTRUCTOR*)

```
Optional(Reconstructor,RECONSTRUCTOR)
Class(Destructor,DESTRUCTOR)
Optional(Destructor,DESTRUCTOR)
VarCl(Constructor,CONSTRUCTOR)
Class(ComponentKind,COMPONENTKIND)
ListCl(ComponentKind,COMPONENTKIND)
StringCl(ComponentKind,COMPONENTKIND)
Class(RecordVariant,RECORDVARIANT)
VarCl(Variant,VARIANT)                                                    500
ChoiceCl(Variant,VARIANT)
Class(VariantDef,VARIANTDEF)
Class(SortDef,SORTDEF)
VarCl(TypeDef,TYPEDEF)
ListCl(TypeDef,TYPEDEF)
Class(TypeDecl,TYPEDECL)
Class(FormalArrayParameter,FORMALARRAYPARAMETER)
Optional(FormalArrayParameter,FORMALARRAYPARAMETER)
Class(ObjectDef,OBJECTDEF)
ListCl(ObjectDef,OBJECTDEF)                                               510
Class(ObjectDecl,OBJECTDECL)
ClassRest(FormalSchemeParameter,FORMALSCHEMEPARAMETER)
Optional(FormalSchemeParameter,FORMALSCHEMEPARAMETER)
Class(SchemeDef,SCHEMEDEF)
ListCl(SchemeDef,SCHEMEDEF)
Class(SchemeDecl,SCHEMEDECL)
VarCl(Decl,DECL)
OptString(Decl,DECL)
VarCl(ModuleDecl,MODULEDECL)
StringCl(ModuleDecl,MODULEDECL)                                           520
Class(Specification,SPECIFICATION)

#endif /* RSL_H */
```

F.8 RSLEDWIN.H

```
/************************************************************************
*
* RSLEDWIN.H
*
* Main Program Specification, RAISE Specification Language Editor
*
* Created 20 December, 1993, Michael Suodenjoki
*
************************************************************************/
                                                                          10
#ifndef RSLEDWIN_H
#define RSLEDWIN_H

#define DEBUG 1

#include <WINDOWS.H>
#include <MEMORY.H>
#include <STDLIB.H>

// Incluce resource .h file for ID* identifiers                           20
#include "RSLMENU.H"

// Include Syntax Tree
#include "SYNTREE.H"

// Include Language dependent features
#include "LANGUAGE.H"
#include "RSL.H"
```

```
// Root is the main Abstract Syntax-Tree
// SelectedNode points to the selected node in Abstract Syntax-Tree
#ifndef GLOBAL
extern PSYNTAXTREE Root;
extern PSYNTAXTREE SelectedNode;
extern PSYNTAXTREE EditTree;
#else
PSYNTAXTREE Root = new ModuleDecl;
PSYNTAXTREE SelectedNode = Root;
PSYNTAXTREE EditTree = NULL;
#endif

/*********************************************************************
*
* Window Specific Variables
*
*********************************************************************/

// Name of Main Application Window Class
#ifndef GLOBAL
extern char szAppName[];
#else
char szAppName[]    = "RSLED";
#endif


#ifndef GLOBAL
#define GLOBAL extern
#endif

// Window Handles
GLOBAL HWND hwndMain;
GLOBAL HWND hwndPosChild;
GLOBAL HWND hwndTransChild;
GLOBAL HWND hwndEditChild;
GLOBAL HWND hwndTextEdit;

// Handle for (old) Window in focus
GLOBAL HWND wndOldFocus;

// Button Handles
GLOBAL HWND  hwndButton[NUMTRANS];

// Prototypes of Window Procedures
long FAR PASCAL PosWndProc(HWND,unsigned,WPARAM,LPARAM);
long FAR PASCAL TransWndProc(HWND,unsigned,WPARAM,LPARAM);
long FAR PASCAL EditWndProc(HWND,unsigned,WPARAM,LPARAM);
long FAR PASCAL WndProc(HWND,unsigned,WPARAM,LPARAM);

// Defintion of special Win32/Win function
#if __NT__
inline HINSTANCE GetWindowNumber(HWND hwnd) {
  return (HINSTANCE) GetWindowLong(hwnd,GWL_HINSTANCE);
}
#else
inline WORD GetWindowNumber(HWND hwnd) {
  return GetWindowWord(hwnd,GWW_HINSTANCE);
}
#endif

// Prototypes of other functions
void updateScrollBars(HWND);
void updateWindowOrigin(VIEW v);

// Font Handles
GLOBAL HFONT hfontNormal;
```

```
GLOBAL HFONT hfontKeyword;
GLOBAL HFONT hfontPlaceholder;
GLOBAL HFONT hfontSymbol;
GLOBAL HFONT hfontComment;
GLOBAL HFONT hfontText;                                                    100

// Window specific variables
GLOBAL int cxChar;              // Average with of a character
GLOBAL int cyChar;              // Height of a character
GLOBAL int widthClient;         // Width of client area of edit-window
GLOBAL int heightClient;        // Height of client area of edit-window

#endif /* RSLEDWIN_H */
```

F.9 SYNNODE.H

```
/************************************************************************
*
* SYNNODE.H
*
* Specification of Syntax Tree Nodes
*
* Created 19 December, 1993, Michael Suodenjoki
*
************************************************************************/
                                                                           10
#ifndef SYNNODE_H
#define SYNNODE_H

#include "DEF.H"


// Prototype for class
class SyntaxTreeNode;

typedef SyntaxTreeNode* PSYNNODE;                                          20

// Class Specification
class SyntaxTreeNode {
  NODEID        nodeid;
  BOOL          resting_place;
  BOOL          listnode;
  BOOL          optional;
  BOOL          selected;
  unsigned char minfixed;
  BOOL          elided;                                                  30
public:
  VIEW view;

  SyntaxTreeNode();

  void set_nodeid(NODEID nid) { nodeid=nid; }
  void set_resting_place(BOOL rp) { resting_place=rp; }
  void set_listnode(BOOL ln) { listnode=ln; }
  void set_optional(BOOL op) { optional=op; }
  void set_minfixed(const unsigned char mf) { minfixed=mf; }               40
  void set_elided(BOOL el) { elided=el; }

  BOOL is_selected(void) const { return selected; }
  BOOL is_resting_place(void) const { return resting_place; }
  BOOL is_listnode(void) const { return listnode; }
  BOOL is_optional(void) const { return optional; }
  BOOL is_elided(void) const { return elided; }
```

```
  unsigned char get_minfixed(void) const { return minfixed; }
                                                                              50
  void select(void) { selected=TRUE; }
  void de_select(void) { selected=FALSE; }

  void set_view(const int,const int,const unsigned int,const unsigned int);
  VIEW get_view(void) const;
  BOOL isinview(const int,const int);
  BOOL view_overlaps_view(VIEW);
  BOOL view_isin_view(VIEW);

  NODEID get_nodeid(void) const { return nodeid; }                            60

  virtual BOOL compare(const PSYNNODE,const PSYNNODE) const;

};

#endif /* SYNNODE_H */
```

F.10 SYNTREE.H

```
/************************************************************************
*
* SYNTREE.H
*
* Specification of Abstract Syntax Tree
*
* Created 19 December, 1993, Michael Suodenjoki
*
************************************************************************/
                                                                              10
#ifndef SYNTREE_H
#define SYNTREE_H

#include "DEF.H"
#include "IO.H"
#include "NARYTREE.H"

// Class prototype
class SyntaxTree;
                                                                              20
typedef SyntaxTree* PSYNTAXTREE;

// Class specification
class SyntaxTree: public NaryTree {
  static PSYNTAXTREE clip_buffer;
  static BOOL        mult_clipped; // is multiple listnodes in clipbuffer
public:
  SyntaxTree();

  void insert(PSYNTAXTREE,PSYNTAXTREE = NULL,PSYNTAXTREE = NULL,              30
              PSYNTAXTREE = NULL,PSYNTAXTREE = NULL);

  PSYNTAXTREE ascend_parent(BOOL&);
  PSYNTAXTREE forward_preorder(BOOL&);
  PSYNTAXTREE forward_preorder_with_optionals(BOOL&);
  PSYNTAXTREE forward_sibling(BOOL&);
  PSYNTAXTREE forward_sibling_with_optionals(BOOL&);
  PSYNTAXTREE backward_preorder(BOOL&);
  PSYNTAXTREE backward_preorder_with_optionals(BOOL&);
  PSYNTAXTREE backward_sibling(BOOL&);                                        40
  PSYNTAXTREE backward_sibling_with_optionals(BOOL&);
```

```
  void de_select(const int,const int,BOOL&);
  void de_select_mult(BOOL&);

  PSYNTAXTREE belongs_to(const int,const int);

  void unparse(OUTPUT,FILEHANDLE);
  BOOL parse_file(FILEHANDLE);

  virtual void make_subtree(void);
  virtual char *unparse_special(void) const;
  virtual PSYNTAXTREE make_child(void);

  void cut_to_clipped(BOOL,BOOL);
  BOOL paste_from_clipped(BOOL&);

private:
  PSYNTAXTREE forward_preorder(const BOOL,const BOOL,BOOL&);
  PSYNTAXTREE backward_preorder(const BOOL,const BOOL,BOOL&);

  void insert_optional(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_first(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_before(const PSYNTAXTREE,const BOOL,BOOL&);
  void insert_after(const PSYNTAXTREE,const BOOL,BOOL&);
  void delete_optional(const PSYNTAXTREE,BOOL&);
  void delete_listnode(const PSYNTAXTREE,BOOL&);

  SYMBOL nextsymbol(const char *,unsigned int &);
  void unparse_symbol_to_screen(const char *,PSYNTAXTREE,int&,int&,BOOL,
                                unsigned int,unsigned int);
  BOOL unparse_symbol_to_file(FILEHANDLE,const char *,PSYNTAXTREE,int&,int&,BOOL,
                              unsigned int,unsigned int);

  PSYNTAXTREE copy_tree(void);
  virtual PSYNTAXTREE clone(void) const;
};

#endif /* SYNTREE_H */
```

F.11 TEXTBUF.H

```
/*************************************************************************
*
* TEXTBUF.H
*
* Specification of Text Buffer Class
*
* Created 6 February, 1994, Michael Suodenjoki
*
*************************************************************************/

#ifndef TEXTBUF_H
#define TEXTBUF_H

#include <WINDOWS.H>
#include "DEF.H"

#define MaxPos 255

class TextBuffer {
  char *buffer;
  int charPos,charMaxPos;
  VIEW bufview;
```

```
    HWND hwnd;
public:
    TextBuffer();
    TextBuffer(HWND, VIEW,char *);
    ~TextBuffer();
    char *get_buffer(void);
    int get_buflen(void) { return charMaxPos; }
                                                                        30
    void insert_char(char);
    void delete_char(void);
    void delete_prev_char(void);

    void left(void);
    void right(void);

    void paint(void);
    void update_caret(void);
    void erase(int);                                                    40
};

#endif /* TEXTBUF_H */
```

Appendix G

Implementation .CPP files

G.1 EDITWND.CPP

```
/*****************************************************************************
*
* EDITWND.CPP
*
* Edit Window Procedure Implementation
*
* Created 20 December, 1993, Michael Suodenjoki
*
*****************************************************************************/
                                                                              10
#ifndef EDITWND_CPP
#define EDITWND_CPP

#include "RSLEDWIN.H"
#include "IO.H"
#include "TEXTBUF.H"

// Global (private) variables for this module
static int      XPos,XPosMax; // Horizontal scrollbar variables
static int      YPos,YPosMax; // Vertical scrollbar variables                 20
static int      ViewOrgX;     // X-coord of current view-origin
       int      ViewOrgY;     // Y-coord of current view-origin
static BOOL     DoingTextEdit;

// Prototypes for process functions
void EditWndProcessCREATE(HWND);
void EditWndProcessUNDO(HWND);
void EditWndProcessCUT(HWND);
void EditWndProcessPASTE(HWND);
void EditWndProcessCOPY(HWND);                                                30
void EditWndProcessKEYDOWN(HWND,WPARAM);
void EditWndProcessCHAR(HWND,WPARAM);
void EditWndProcessVSCROLL(HWND,WPARAM);
void EditWndProcessHSCROLL(HWND,WPARAM);
void EditWndProcessLBUTTONDOWN(HWND,LPARAM);
void EditWndProcessMOUSEMOVE(HWND,WPARAM,LPARAM);
void EditWndProcessPAINT(HWND);

// Prototypes for utility function
BOOL StopTextEdit(HWND);                                                      40
void InvalidateSelected(HWND,PSYNTAXTREE Selected,BOOL);
```

```
// Mouse variables
static BOOL MouseMove = FALSE; // Is TRUE if mouse moves while left button pressed
static BOOL MultListnodesSelected = FALSE; // Is TRUE if multiple nodes are selected

// Variable to communicate from transform to edit window
BOOL justTransformed = FALSE;

// Previous selection                                                          50
PSYNTAXTREE Sel = NULL;

// A very limited text buffer
TextBuffer* tbuf;

/*****************************************************************************
*
* Edit Window Procedure
*
*****************************************************************************/                                          60

long FAR PASCAL EditWndProc(HWND hwnd,unsigned message,WPARAM wParam,LPARAM lParam)
{
  switch(message) {
    case WM_CREATE:
      EditWndProcessCREATE(hwnd);
      return 0;
    case WM_COMMAND:
      if(wParam==1&&HIWORD(lParam)==EN_ERRSPACE)
        MessageBox(hwnd,"Text edit out of space.",szAppName,MB_OK | MB_ICONEXCLAMATION);   70
      return 0;
    case WM_UNDO:
      EditWndProcessUNDO(hwnd);
      return 0;
    case WM_CUT:
      EditWndProcessCUT(hwnd);
      return 0;
    case WM_PASTE:
      EditWndProcessPASTE(hwnd);
      return 0;                                                                 80
    case WM_COPY:
      EditWndProcessCOPY(hwnd);
      return 0;
    case WM_VSCROLL:
      EditWndProcessVSCROLL(hwnd,wParam);
      return 0;
    case WM_HSCROLL:
      EditWndProcessHSCROLL(hwnd,wParam);
      return 0;
    case WM_CHAR:                                                               90
      EditWndProcessCHAR(hwnd,wParam);
      return 0;
    case WM_KEYDOWN:
      EditWndProcessKEYDOWN(hwnd,wParam);
      return 0;
    case WM_LBUTTONDOWN:
      SelectedNode->de_select_mult(MultListnodesSelected);
      EditWndProcessLBUTTONDOWN(hwnd,lParam);
      return 0;
    case WM_LBUTTONUP:                                                          100
      if(MouseMove) {
        InvalidateRect(hwndPosChild,NULL,TRUE);
        InvalidateRect(hwndTransChild,NULL,FALSE);
        MouseMove=FALSE;
      }
      return 0;
    case WM_MOUSEMOVE:
      EditWndProcessMOUSEMOVE(hwnd,wParam,lParam);
      return 0;
```

```
    case WM_RBUTTONDOWN:
      SendMessage(hwnd,WM_KEYDOWN,VK_F3,0L);
      return 0;
    case WM_SIZE:
      widthClient = LOWORD(lParam);
      heightClient = HIWORD(lParam);
      if(Root) Root->unparse(none,NULL); // update views
      updateScrollBars(hwnd);
      return 0;
    case WM_PAINT:
      EditWndProcessPAINT(hwnd);
      return 0;
    case WM_DESTROY :
      PostQuitMessage(0);
      return 0;
  }
  return DefWindowProc(hwnd,message,wParam,lParam);
}

/*********************************************************
* Process WM_CREATE messages. Creates four different
* fonts and presets scrollbar variables.
* Input:
* hwnd - Handle of window
* Output:
* none
* Side Effects:
*
*********************************************************/
void EditWndProcessCREATE(HWND hwnd) {

  XPos = XPosMax = 0;
  YPos = YPosMax = 0;

  SetScrollRange(hwnd,SB_VERT,0,YPosMax,FALSE);
  SetScrollPos(hwnd,SB_VERT,YPos,TRUE);

  SetScrollRange(hwnd,SB_HORZ,0,XPosMax,FALSE);
  SetScrollPos(hwnd,SB_HORZ,XPos,TRUE);

  ViewOrgX=ViewOrgY=0;

  DoingTextEdit=FALSE;
}

/*********************************************************
* Process IDM_UNDO messages. Should undo the last
* edition
* Input:
* hwnd - Handle of window
* Output:
* none
* Side Effects:
* Changes the clip buffer and the abstract syntax-tree
*********************************************************/
void EditWndProcessUNDO(HWND hwnd) {
  error("Undo not yet implemented !","Undo Error");
}


/*********************************************************
* Process IDM_CUT messages. Should move the current
* selection to the clipbuffer.
* Input:
* hwnd - Handle of window
* Output:
* none
```

```
* Side Effects:
* Changes the clip buffer and the abstract syntax-tree
*******************************************************/
void EditWndProcessCUT(HWND hwnd) {                                    180
  // Cut selected to clipped buffer
  SelectedNode->cut_to_clipped(MultListnodesSelected,FALSE);

  updateScrollBars(hwnd);

  // Update screen
  InvalidateRect(hwnd,NULL,TRUE);
  InvalidateRect(hwndPosChild,NULL,TRUE);
  InvalidateRect(hwndTransChild,NULL,TRUE);
}                                                                      190

/*******************************************************
* Process IDM_PASTE messages. Should move the current
* contents of the clipbuffer to the current selection
* Input:
* hwnd - Handle of window
* Output:
* none
* Side Effects:
* Changes the clip buffer and the abstract syntax-tree                 200
*******************************************************/
void EditWndProcessPASTE(HWND hwnd) {
  if(SelectedNode->paste_from_clipped(MultListnodesSelected)) {
    // Update views
    Root->unparse(none,NULL);

    updateScrollBars(hwnd);

    // Update screen
    InvalidateRect(hwnd,NULL,TRUE);                                    210
    InvalidateRect(hwndPosChild,NULL,TRUE);
    InvalidateRect(hwndTransChild,NULL,TRUE);
  }
}

/*******************************************************
* Process IDM_COPY messages. Should move the current
* selection to the clipbuffer.
* Input:
* hwnd - Handle of window                                              220
* Output:
* none
* Side Effects:
* Changes the clip buffer and the abstract syntax-tree
*******************************************************/
void EditWndProcessCOPY(HWND hwnd) {
  // Copy selected to clipped buffer
  SelectedNode->cut_to_clipped(MultListnodesSelected,TRUE);
}
                                                                       230
/*******************************************************
* Process WM_VSCROLL messages. Reacts on scroll bars
* Input:
* hwnd - Handle of window
* wParam - Which scrollbar command has been sent
* Output:
* none
* Side Effects:
*
*******************************************************/                240
void EditWndProcessVSCROLL(HWND hwnd,WPARAM wParam) {
  switch(wParam) {
    case SB_LINEUP: YPos=max(0,YPos-1); break;
```

```
    case SB_LINEDOWN: YPos=min(YPosMax,YPos+1); break;
    case SB_PAGEUP: YPos=max(0,YPos-heightClient/cyChar); break;
    case SB_PAGEDOWN: YPos=YPos+heightClient/cyChar; break;
    case SB_THUMBPOSITION: break;
    default: break;
  }
  if(YPos != GetScrollPos(hwnd,SB_VERT)) {                                    250
    ViewOrgY=YPos*cyChar;
    if(ERROR==ScrollWindowEx(hwnd,0,-(YPos-GetScrollPos(hwnd,SB_VERT))*cyChar,
      NULL,NULL,NULL,NULL,SW_ERASE | SW_INVALIDATE))
    {
      InvalidateRect(hwnd,NULL,TRUE);
      UpdateWindow(hwnd);
    }
    SetScrollPos(hwnd,SB_VERT,YPos,TRUE);
  }
}                                                                             260

/*********************************************************
* Process WM_HSCROLL messages. Reacts on scroll bars
* Input:
* hwnd - Handle of window
* wParam - Which scrollbar command has been sent
* Output:
* none
* Side Effects:
*                                                                             270
*********************************************************/
void EditWndProcessHSCROLL(HWND hwnd,WPARAM wParam) {
  switch(wParam) {
    case SB_LINELEFT: XPos=max(0,XPos-1); break;
    case SB_LINERIGHT: XPos=min(XPosMax,XPos+1); break;
    case SB_PAGELEFT: XPos=max(0,XPos-widthClient/cxChar); break;
    case SB_PAGERIGHT: XPos=XPos+widthClient/cxChar; break;
    case SB_THUMBPOSITION: break;
    default: break;
  }                                                                           280
  if(XPos != GetScrollPos(hwnd,SB_HORZ)) {
    ViewOrgX=XPos*cxChar;
    if(ERROR==ScrollWindowEx(hwnd,-(XPos-GetScrollPos(hwnd,SB_HORZ))*cxChar,0,
       NULL,NULL,NULL,NULL,SW_ERASE | SW_INVALIDATE))
    {
      InvalidateRect(hwnd,NULL,TRUE);
      UpdateWindow(hwnd);
    }
    SetScrollPos(hwnd,SB_HORZ,XPos,TRUE);
  }                                                                           290
}

/*********************************************************
* Process WM_CHAR messages
* Input:
* hwnd - Handle of window
* wParam - Window parameters (contains pressed key)
* Output:
* none
* Side Effects:                                                               300
* Could invalidate window, so that it must be redrawn.
*********************************************************/
void EditWndProcessCHAR(HWND hwnd,WPARAM wParam) {
  HDC           hdc;

  switch(wParam) {
    case '\r':                          // Enter
      if(DoingTextEdit) {
        if(justTransformed=StopTextEdit(hwnd)) {
          SendMessage(hwnd,WM_KEYDOWN,VK_F5,(LPARAM) 0);                      310
```

```
        }
        else InvalidateSelected(hwnd,SelectedNode,TRUE);
      }
      else
      SendMessage(hwnd,WM_KEYDOWN,VK_F3,(LPARAM) 0);
      break;
    case '\x0b' :                          // Ctrl-K
      if(DoingTextEdit) {          // Stop text editing
        delete tbuf;
        DoingTextEdit=FALSE;
      }
      else {
        SelectedNode->delete_subtree(SelectedNode);
        SelectedNode->make_subtree();
      }
      updateScrollBars(hwnd);
      InvalidateRect(hwnd,NULL,TRUE);
      InvalidateRect(hwndPosChild,NULL,TRUE);
      InvalidateRect(hwndTransChild,NULL,TRUE);
      break;
    case '\x0e' :                          // Ctrl-N
      SendMessage(hwnd,WM_KEYDOWN,VK_F2,0L);
      break;
    case '\x10' :                          // Ctrl-P
      SendMessage(hwnd,WM_KEYDOWN,VK_F6,0L);
      break;
    case '\x05' :                          // Ctrl-E
      SelectedNode->set_elided(!SelectedNode->is_elided());
      updateScrollBars(hwnd);
      InvalidateSelected(hwnd,SelectedNode,TRUE);
      InvalidateRect(hwndPosChild,NULL,TRUE);
      InvalidateRect(hwndTransChild,NULL,TRUE);
      break;
    case '\b':                             // Backspace
      if(DoingTextEdit)
        tbuf->delete_prev_char();
      else SendMessage(hwnd,WM_KEYDOWN,VK_F7,0L);
      break;
    default:
      if(wParam>31&&wParam<=255) { // If normal characters
        if(!DoingTextEdit) {
          DoingTextEdit=TRUE;
          tbuf = new TextBuffer(hwnd,SelectedNode->get_view(),NULL);
        }
        tbuf->insert_char((char) wParam);
      }
      break;
  }
}
/*********************************************************
* Process WM_KEYDOWN messages
* Input:
* hwnd - Handle of window
* wParam - Window parameters (contains pressed key)
* lParam - if TRUE then this function has been
* called from Transformation window
* Output:
* none
* Side Effects:
* Could invalidate window, so that it must be redrawn.
*********************************************************/
void EditWndProcessKEYDOWN(HWND hwnd,WPARAM wParam) {
  VIEW v;
  RECT rect;
  BOOL flag;

  switch(wParam) {
```

```
    case VK_LEFT:
      if(DoingTextEdit) tbuf->left();
      break;
    case VK_RIGHT:
      if(DoingTextEdit) tbuf->right();
      break;
    case VK_DELETE:
      if(DoingTextEdit) tbuf->delete_char();
      break;
    default:
      if(wParam>=VK_F1&&wParam<=VK_F9&&SelectedNode) {

        if(MultListnodesSelected) InvalidateRect(hwnd,NULL,FALSE);
        SelectedNode->de_select_mult(MultListnodesSelected);

        // Stop text edition first
        if(DoingTextEdit) {
          justTransformed=StopTextEdit(hwnd);
          if(!justTransformed) return;
        }

        // Find rectangle to repaint
        v=SelectedNode->get_view();
        SetRect(&rect,0,v.y-ViewOrgY,widthClient,heightClient);

        // Do movement
        switch(wParam) {
          case VK_F1: SelectedNode=SelectedNode->ascend_parent(flag); break;
          case VK_F2: SelectedNode=SelectedNode->forward_preorder(flag); break;
          case VK_F3: SelectedNode=SelectedNode->forward_preorder_with_optionals(flag); break;
          case VK_F4: SelectedNode=SelectedNode->forward_sibling(flag); break;
          case VK_F5: SelectedNode=SelectedNode->forward_sibling_with_optionals(flag); break;
          case VK_F6: SelectedNode=SelectedNode->backward_preorder(flag); break;
          case VK_F7: SelectedNode=SelectedNode->backward_preorder_with_optionals(flag); break;
          case VK_F8: SelectedNode=SelectedNode->backward_sibling(flag); break;
          case VK_F9: SelectedNode=SelectedNode->backward_sibling_with_optionals(flag); break;
        };

        // Update rectangle to repaint
        if(flag||justTransformed) Root->unparse(none,NULL);
          // if a node is inserted or deleted then force recalculation of views
        v=SelectedNode->get_view();
        if(v.y<rect.top) rect.top=v.y;

        if(v.y+cyChar>ViewOrgY+heightClient||v.y<ViewOrgY||
           v.x+v.width>ViewOrgX+widthClient||v.x<ViewOrgX)
        {
          SetRect(&rect,0,0,widthClient,heightClient);
          if(!justTransformed) flag=1;
        }

  // if(lParam) { YPos=v.y/cyChar; XPos=v.x/cxChar; }
        updateWindowOrigin(v);
        updateScrollBars(hwnd);

        // Repaint
        InvalidateRect(hwnd,&rect,justTransformed || flag);
        InvalidateRect(hwndPosChild,NULL,TRUE);
        InvalidateRect(hwndTransChild,NULL,FALSE);

        justTransformed=FALSE;
    }
  }
}

/*******************************************************
* Process WM_LBUTTONDOWN messages
```

```
* Input:
* hwnd - Handle of window
* lParam - Window parameters (contains mouse coords)
* Output:
* none
* Side Effects:                                                          450
* Could invalidate window, so that it must be redrawn.
*********************************************************/
void EditWndProcessLBUTTONDOWN(HWND hwnd,LPARAM lParam) {
  VIEW v;
  RECT rect;
  BOOL flag;

  if(!Root||!SelectedNode) return;

  // Stop text edition first                                              460
  if(DoingTextEdit) {
    justTransformed=StopTextEdit(hwnd);
    if(!justTransformed) return;
  }

  // Calculate mouse coordinates in virtual window
  const WORD xMouse=LOWORD(lParam)+ViewOrgX;
  const WORD yMouse=HIWORD(lParam)+ViewOrgY;

  // Find new selection                                                   470
  PSYNTAXTREE temp = Root->belongs_to(xMouse,yMouse);

  // Deselect and maybe delete optionals
  SelectedNode->de_select(xMouse,yMouse,flag);

  // Undraw previous selection. Only necessary if
  // the previous selection is delete e.i. was optional
  if(flag) {
    v=SelectedNode->get_view();
    SetRect(&rect,0,v.y-ViewOrgY,widthClient,heightClient);              480
    InvalidateRect(hwnd,&rect,flag);
    UpdateWindow(hwnd);
    flag=0;
  }
  // The new selection
  SelectedNode=temp;
  Sel=temp;

  v=SelectedNode->get_view();
  if(v.y+cyChar>ViewOrgY+heightClient||v.y<ViewOrgY||                     490
     v.x+v.width>ViewOrgX+widthClient||v.x<ViewOrgX)
    flag=1;

  // Select new selection
  SelectedNode->select();

  updateWindowOrigin(v);
  updateScrollBars(hwnd);

  // Repaint                                                              500
  InvalidateRect(hwnd,NULL,flag);
  InvalidateRect(hwndPosChild,NULL,TRUE);
  InvalidateRect(hwndTransChild,NULL,FALSE);
}

/*********************************************************
* Process WM_MOUSEMOVE messages
* Input:
* hwnd - Handle of window
* Output:                                                                510
* none
```

```
* Side Effects:
* Could invalidate window, so that it must be redrawn.
*********************************************************/
void EditWndProcessMOUSEMOVE(HWND hwnd,WPARAM wParam,LPARAM lParam) {
  MouseMove=wParam & MK_LBUTTON;
  if(MouseMove) {
    VIEW v;
    RECT rect;
    BOOL flag = FALSE;                                                    520

    if(!Root||!SelectedNode) return;

    // Calculate mouse coordinates in virtual window
    const WORD xMouse=LOWORD(lParam)+ViewOrgX;
    const WORD yMouse=HIWORD(lParam)+ViewOrgY;

    // Find new selection
    PSYNTAXTREE temp = Root->belongs_to(xMouse,yMouse);
                                                                          530
    if(temp==Sel&&!MultListnodesSelected||
       MultListnodesSelected&&!SelectedNode->isinview(xMouse,yMouse))
    {
      if(MultListnodesSelected&&(temp->get_parent()!=SelectedNode->get_parent()))
      {
        SelectedNode->de_select_mult(MultListnodesSelected);
      }
      MultListnodesSelected=(SelectedNode->get_parent()&&
        temp->get_parent()&&
        SelectedNode->get_parent()->is_listnode()&&                       540
        temp->get_parent()==SelectedNode->get_parent());
      if(!MultListnodesSelected)
        SelectedNode->de_select(xMouse,yMouse,flag);

      // Undraw previous selection. Only necessary if
      // the previous selection is delete e.i. was optional
      if(flag) {
        InvalidateSelected(hwnd,SelectedNode,flag);
        UpdateWindow(hwnd);
        flag=0;                                                           550
      }
      // The new selection
      SelectedNode=temp;

      v=SelectedNode->get_view();
      if(v.y+cyChar>ViewOrgY+heightClient||v.y<ViewOrgY||
         v.x+v.width>ViewOrgX+widthClient||v.x<ViewOrgX)
        flag=1;

      // Select new selection                                             560
      SelectedNode->select();

      // Repaint
      InvalidateRect(hwnd,NULL,flag);
    }

  }
}

/*********************************************************              570
* Process WM_PAINT messages
* Input:
* hwnd - Handle of window
* Output:
* none
* Side Effects:
* Could invalidate window, so that it must be redrawn.
*********************************************************/
```

```
void EditWndProcessPAINT(HWND hwnd) {
  HDC         hdc;                                                    580
  PAINTSTRUCT ps;

  hdc = BeginPaint(hwnd,&ps);
  // Show busy mouse
  show_mouse_busy();

  // NOTE: You can use the paint structure information to make
  // the unparsing quicker. Let the unparse function get
  // the ps.rect parameter and then test ps.rect when
  // outputting.                                                      590

  SelectObject(hdc,hfontNormal);

  // Set HDC for screen I/O
  set_outputHDC(hdc);

  // Set the window origin
#ifdef __NT__
  SetWindowOrgEx(hdc, ViewOrgX, ViewOrgY, NULL);
#else                                                                600
  SetWindowOrg(hdc, ViewOrgX, ViewOrgY);
#endif

  // Do the actual outputting
  if(Root) Root->unparse(to_screen,NULL);

  // Write edit textbuffer
  if(DoingTextEdit&&tbuf) {
    tbuf->paint();
    tbuf->erase(tbuf->get_buflen());                                 610
  }

  // Show mouse normal e.i. arrow shape
  show_mouse_normal();

  EndPaint(hwnd,&ps);
}

/*****************************************************************************
*                                                                           620
* Utility rutines for Edit Window Procedure
*
*****************************************************************************/

/*********************************************************
* Given a view which should represent the current
* selection, update the Virtual Window Origin
* so the right screen(page) are shown under WM_PAINT
* Input:
* v - View                                                               630
* Output:
* none
* SideEffects:
* Sets ViewOrgX, XPos, ViewOrgY and YPos
*********************************************************/
void updateWindowOrigin(VIEW v) {
  // Check whether the selection is outside window
  // if that is the case then change window origins
  if(v.y+cyChar>ViewOrgY+heightClient)
    ViewOrgY=ViewOrgY+(heightClient/cyChar)/2*cyChar;//heightClientheightClient/2);  640
  if(v.y<ViewOrgY)
    ViewOrgY=max(0,v.y-(heightClient/cyChar)/2*cyChar);
  YPos=ViewOrgY/cyChar;

  if(v.x+v.width>ViewOrgX+widthClient&&v.width<widthClient)
```

```
    ViewOrgX = ViewOrgX + widthClient/2;
  if(v.x < ViewOrgX)
    ViewOrgX = max(0,v.x - widthClient/2);
  XPos = ViewOrgX / cxChar;
}

/*********************************************************
* Update the scroll bar range and position
* Input:
* hwnd - Handle of window owning scroll bars
* Output:
* none
* Side effects:
* The YPos, YPosMax, XPos and XPosMax global variables
* are changed appropiately
*********************************************************/
void updateScrollBars(HWND hwnd) {
  if(!Root) return;
  VIEW sv=Root->get_view();

  YPosMax=sv.height/cyChar;
  if(YPosMax>heightClient/cyChar)
    SetScrollRange(hwnd,SB_VERT,0,YPosMax,FALSE);
  else {
    SetScrollRange(hwnd,SB_VERT,0,0,TRUE);
    ViewOrgY=0;
  }
  SetScrollPos(hwnd,SB_VERT,YPos,TRUE);

  XPosMax=sv.width/cxChar;
  if(XPosMax>widthClient/cxChar)
    SetScrollRange(hwnd,SB_HORZ,0,XPosMax,FALSE);
  else {
    SetScrollRange(hwnd,SB_HORZ,0,0,TRUE);
    ViewOrgX=0;
  }
  SetScrollPos(hwnd,SB_HORZ,XPos,FALSE);
}

/*********************************************************
* Stops the text editing, starts parsing and
* return if all went ok
* Input:
* hwnd - handle of window
* Output:
* BOOL - is TRUE if parsing went ok
*********************************************************/
BOOL StopTextEdit(HWND hwnd)
{
  BOOL flag=FALSE;

  // At this point the text edit is finished
  // and the parser can be called
  // But first save it in a temporary file
  HANDLE tempfile;
  tempfile = CreateFile("rsled000.tmp", GENERIC_WRITE, 0,
               (LPSECURITY_ATTRIBUTES) NULL,
               CREATE_ALWAYS,
               FILE_ATTRIBUTE_NORMAL,
               (HANDLE) 0);


  char nbbuf[6];
  int nbbuflen;
  nbbuflen=wsprintf(nbbuf,"@%i ",SelectedNode->get_nodeid());
  if(!writefile(tempfile,nbbuf,nbbuflen)||
     !writefile(tempfile,tbuf->get_buffer(),tbuf->get_buflen())) {
```

```
    error("Couldn't create parse buffer file !","Parse Error");
    flag=FALSE;
  }
  else {
    // And now call parser width file
    CloseHandle(tempfile);
    tempfile = CreateFile("rsled000.tmp", GENERIC_READ, 0,
                  (LPSECURITY_ATTRIBUTES) NULL,                                  720
                  OPEN_EXISTING,
                  FILE_ATTRIBUTE_NORMAL,
                  (HANDLE) 0);

    if(!SelectedNode->parse_file(tempfile)) {
      SelectedNode->delete_subtree(SelectedNode);
      SelectedNode->paste_subtree(EditTree);
      flag=TRUE;
// SelectedNode->select();
      // Should the root of EditTree be deleted ? YES                            730
      // delete EditTree
    }
    CloseHandle(tempfile);
  }
  if(flag) { delete tbuf; DoingTextEdit=FALSE; }
  return flag;
}

/*********************************************************
* The function invalidates the are from the selected node                       740
* and downwards.
* Input:
* hwnd - window to invalidate area in
* SelectedNode - the Selected Node
* flag - should the invalidated area be erased before
* reprinted ?
* Output:
* none
* Side Effects:
* changes the screen at invalidates area                                        750
*********************************************************/
void InvalidateSelected(HWND hwnd,PSYNTAXTREE SelectedNode,BOOL flag) {
  RECT rect;
  VIEW v;
  v=SelectedNode->get_view();
  SetRect(&rect,0,v.y-ViewOrgY,widthClient,heightClient);
  InvalidateRect(hwnd,&rect,flag);
}

#endif /* EDITWND_CPP */                                                        760
```

G.2 IO.CPP

```
/*************************************************************************
*
* IO.CPP
*
* Implementation of Input / Output Rutines
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/
                                                                               10
#ifndef IO_CPP
#define IO_CPP
```

```
#include "IO.H"
// PS. Do not need to include MYSTRING.H since this unit
// knows that it is IO dependent of Windows 3.1

#ifndef GLOBAL
#define GLOBAL extern
#endif

GLOBAL int cxChar;
GLOBAL int cyChar;
GLOBAL int widthClient;
GLOBAL int heightClient;

// Handle of Device Context for output
static HDC outputHDC;
// Handle for font currently in use
static FONT hFontInUse;
// Handle for currently used mouse shape
static HCURSOR hCursor;

// Window handle for main window
extern HWND hwndMain;
// Application name
extern char szAppName[];


/** Screen Rutines **/

/************************************************************
* Sets the Handle for the Device Context of window to
* where output should occur. The function MUST have been
* called before any of the other screen I/O rutines is
* used.
* Input:
* hdc - Handle of DC
* Output:
* none
* Side Effects:
* Sets the global variable outputHDC, which is used
* in all Screen I/O rutines.
************************************************************/
void set_outputHDC(HDC hdc) {
  outputHDC=hdc;
}

/************************************************************
* Select the current font
* Input:
* font - the font to select
* Output:
* returns the old font
* Side Effects:
* See side effects for SelectObject function in
* windows documentation
************************************************************/
FONT select_font(FONT font) {
  hFontInUse=font;
  return SelectObject(outputHDC,font);
}

/************************************************************
* Gets the current font
* Input:
* none
* Output:
* returns the current font
```

```
* Side Effects:
* See side effects for SelectObject function in
* windows documentation
*********************************************************/
FONT get_font(void) {
  return hFontInUse;
}

/*********************************************************
* Output some text to screen, with the current font,
* current background color and current color
* Input:
* x,y - output to (x,y) position
* lpszString - null terminated string to write
* Output:
* none
* Side Effects:
* See side effects for ExtTextOut function in
* windows documentation
*********************************************************/
void text_out(const int x,const int y,const char *lpszString) {
  RECT r;
  SetRect(&r,x,y,x+sizeof_text(lpszString),y+get_char_height());
  ExtTextOut(outputHDC,x,y,ETO_OPAQUE,&r,lpszString,lstrlen(lpszString),NULL);
}

/*********************************************************
* The function draws a rectangle at the position and
* width/height specified in parameter. The function are
* mainly used for debugging purposes.
* Input:
* rect - (rect.x,rect.y) specify starting coord and
* rect.width and rect.height specify the width
* and height (in pixels) of the rectangle
* Output:
* none
* Side Effects:
* Could affect the screen
*********************************************************/
void draw_rect(const VIEW rect) {
#ifdef __NT__
  MoveToEx(outputHDC,rect.x,rect.y,NULL);
#else
  MoveTo(outputHDC,rect.x,rect.y);
#endif
  LineTo(outputHDC,rect.x+rect.width,rect.y);
  LineTo(outputHDC,rect.x+rect.width,rect.y+rect.height);
  LineTo(outputHDC,rect.x,rect.y+rect.height);
  LineTo(outputHDC,rect.x,rect.y);
}

/*********************************************************
* Returns the current width of the string in pixels,
* Input:
* lpszString - null terminated string to examine
* Output:
* unsigned int - the width in pixels
* Side Effects:
* See side effects for GetTextExtentPoint function in
* windows documentation
*********************************************************/
unsigned int sizeof_text(const char *lpszString) {
  SIZE size;
  GetTextExtentPoint(outputHDC,lpszString,lstrlen(lpszString),&size);
  return size.cx;
}
```

```
/*********************************************************
* Returns the current width of the string in pixels,
* Input:
* lpszString - null terminated string to examine
* Output:
* unsigned int - the width in pixels
* Side Effects:
* See side effects for GetTextExtentPoint function in
* windows documentation
*********************************************************/
void error(const char *string,const char *head) {
  MessageBeep(MB_ICONEXCLAMATION);
  MessageBox(hwndMain,string,head,MB_ICONEXCLAMATION | MB_OK);
}


// Color rutines
void set_bk_color(COLOR color) { SetBkColor(outputHDC,color); }
void set_text_color(COLOR color) { SetTextColor(outputHDC,color); }
COLOR get_text_color(void) { return GetTextColor(outputHDC); }

unsigned int get_char_width(void) { return cxChar; }
unsigned int get_char_height(void) { return cyChar; }

unsigned int get_window_width(void) { return widthClient; }
unsigned int get_window_height(void) { return heightClient; }

/*********************************************************
* Show the mouse shape as it was at last change
* Input:
* none
* Output:
* none
* Side Effects:
* changes the mouse shape
*********************************************************/
void show_mouse_normal(void) {
  ShowCursor(FALSE);
  SetCursor(hCursor);
}

/*********************************************************
* Show the mouse shape busy e.i. hourglass
* Input:
* none
* Output:
* none
* Side Effects:
* changes the mouse shape and remember the old one
*********************************************************/
void show_mouse_busy(void) {
  hCursor = SetCursor(LoadCursor(NULL,IDC_WAIT));
  ShowCursor(TRUE);
}

/** File Rutines **/

/*********************************************************
* Write data to a file.
* Input:
* file - file handler
* buf - buffer to write
* n - number of bytes to write from buffer
* Output:
* BOOL - is TRUE if all went ok
* Side Effects:
* none
```

```
*********************************************************/
BOOL writefile(FILEHANDLE file,const void *buf,unsigned long n) {
#ifdef __NT__
  DWORD nwritten;
  BOOL ret;
  ret=WriteFile(file,buf,n,&nwritten,NULL);
  return ret && (n==nwritten);                                          220
#else
  #pragma SC message "writefile function not yet implemented in IO.cpp file"
  return FALSE;
#endif
}

/*********************************************************
* Read data from a file.
* Input:
* file - file handler                                                   230
* buf - buffer to read into
* toread - number of bytes to read into buffer
* hasread - (output) actual number of bytes read into the
* buffer
* Output:
* BOOL - is TRUE if all went ok
* Side Effects:
* none
*********************************************************/
BOOL readfile(FILEHANDLE file,void *buf,unsigned long toread,           240
              unsigned long &hasread)
{
#ifdef __NT__
  BOOL ret;
  ret=ReadFile(file,buf,toread,&hasread,NULL);
  return ret;
#else
  #pragma SC message "readfile function noy yet implemented in IO.cpp file
  return FALSE;
#endif                                                                  250
}


#endif /* IO_CPP */
```

G.3 MYSTRING.CPP

```
/*************************************************************************
*
* MYSTRING.CPP
*
* Implementation of my string routines
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/
                                                                        10
#ifndef MYSTRING_CPP
#define MYSTRING_CPP

#include "MYSTRING.H"
#ifndef _INC_WINDOWS
#include "STRING.H"
#endif

/*********************************************************
```

```
* Copy string2 to string1
* Input:
* string1, string2 - the two strings
* Output:
* A char pointer to string1
* Side Effects:
* none
*********************************************************/
char *my_strcpy(char *string1,const char *string2) {
#ifdef _INC_WINDOWS
  return lstrcpy(string1,string2);
#else
  return strcpy(string1,string2);
#endif
}

/*********************************************************
* Copy n characters from string2 to string1
* Input:
* string1, string2 - the two strings
* n - number of characters to copy
* Output:
* A char pointer to string1
* Side Effects:
* none
*********************************************************/
char *my_strncpy(char *string1,const char *string2,int n) {
  return strncpy(string1,string2,n);
}


/*********************************************************
* Casesensitive compares two strings
* Input:
* string1, string2 - the two strings to compare
* Output:
* int < 0 - if string1 is less than string2
* int = 0 - if string1 is equal to string2
* int > 0 - if string1 is greater than string2
* Side Effects:
* none
*********************************************************/
int my_strcmp(const char *string1,const char *string2) {
#ifdef _INC_WINDOWS
  return lstrcmp(string1,string2);
#else
  return strcmp(string1,string2);
#endif
}

/*********************************************************
* NonCasesensitive compare two strings
* Input:
* string1, string2 - the two strings to compare
* Output:
* int < 0 - if string1 is less than string2
* int = 0 - if string1 is equal to string2
* int > 0 - if string1 is greater than string2
* Side Effects:
* none
*********************************************************/
int my_strcmpi(const char *string1,const char *string2) {
#ifdef _INC_WINDOWS
  return lstrcmpi(string1,string2);
#else
  return strcmpi(string1,string2);
#endif
```

```
}


/*********************************************************
* Find the length of the string, excluding the teminating
* '\0'. The string MUST be null terminated.
* Input:
* string - the string to find length of
* Output:
* unsigned int - the length of the string
* Side Effects:
* none
*********************************************************/
unsigned int my_strlen(const char *string) {
#ifdef _INC_WINDOWS
  return lstrlen(string);
#else
  return strlen(string);
#endif
}

#endif
```

G.4 NARYTREE.CPP

```
/*************************************************************************
*
* NARYTREE.CPP
*
* Implementation of Nary Trees
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef NARYTREE_CPP
#define NARYTREE_CPP

#include "NARYTREE.H"

/*********************************************************
* Constructor for Nary Tree
* Input:
* none
* Output:
* node
* Side Effects:
* The pointer connections are properly initialized
*********************************************************/
NaryTree::NaryTree() {
  parent=firstchild=lastchild=prevsibling=nextsibling=NULL;
}

/*********************************************************
* Insert a subtree as firstchild of tree.
* Input:
* tree - the subtree to insert
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::insert_tree_first(PNARYTREE tree) {
```

```
   tree->parent=this;
   if(!firstchild) lastchild=tree;
   else {
      tree->nextsibling=firstchild;
      firstchild->prevsibling=tree;
   }
   firstchild=tree;
}

/*********************************************************
* Insert/appends a subtree as lastchild of tree.
* Input:
* tree - the subtree to insert
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::insert_tree_last(PNARYTREE tree) {
   tree->parent=this;
   if(!firstchild) firstchild=tree;
   else {
      lastchild->nextsibling=tree;
      tree->prevsibling=lastchild;
   }
   lastchild=tree;
}

/*********************************************************
* Insert a subtree as prevsibling of tree.
* Input:
* tree - the subtree to insert
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::insert_tree_before(PNARYTREE tree) {
   tree->parent=parent;
   tree->nextsibling=this;
   if(prevsibling) prevsibling->nextsibling=tree;
   tree->prevsibling=prevsibling;
   prevsibling=tree;
   if(this==parent->firstchild) parent->firstchild=tree;
}

/*********************************************************
* Insert a subtree as nextsibling of tree.
* Input:
* tree - the subtree to insert
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::insert_tree_after(PNARYTREE tree) {
   tree->parent=parent;
   tree->prevsibling=this;
   tree->nextsibling=nextsibling;
   if(nextsibling) tree->nextsibling->prevsibling=tree;
   nextsibling=tree;
   if(this==parent->lastchild) parent->lastchild=tree;
}


/*********************************************************
* Deletes a subtree. It MUST be called with a syntax of
```

```
* atree->delete_subtree(t) where atree is a pointer to
* a tree, which is NOT in the subtree of t.
* Input:
* tree - the subtree to delete
* Output:                                                          110
* node
* Side Effects:
* The pointer connections are properly changed
*************************************************************/
void NaryTree::delete_subtree(PNARYTREE tree) {
  PNARYTREE temp = tree;
  if(temp&&temp->firstchild) {
    temp=temp->firstchild;
    if(temp->nextsibling) {
      while(temp->nextsibling) {                                   120
        temp=temp->nextsibling;
        delete_subtree(temp->prevsibling);
        delete_treenode(temp->prevsibling);
      }
    }
    delete_subtree(temp);
    delete_treenode(temp);
  }
}
                                                                   130
/*************************************************************
* Deletes a specific tree node. Does NOT delete eventually
* childs ! Use function delete_tree instead.
* Input:
* node - the node to delete
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*************************************************************/   140
void NaryTree::delete_treenode(PNARYTREE node) {
  // There is 4 cases of how to delete the tree (node)

  // 1. the tree to delete is the only child of the trees parent
  // e.i. has no siblings
  if(node->parent&&node==node->parent->firstchild&&
     node==node->parent->lastchild)
  {
    node->parent->firstchild=NULL;
    node->parent->lastchild=NULL;                                  150
  }
  else
  // 2. the tree to delete is the lastchild of the trees parent
  // e.i. has only a previous sibling
  if(node->parent&&node==node->parent->lastchild) {
    if(node->prevsibling) node->prevsibling->nextsibling=NULL;
    node->parent->lastchild=node->prevsibling;
  }
  else
  // 3. the tree to delete is the firstchild of the trees parent    160
  // e.i. has only a next sibling
  if(node->parent&&node==node->parent->firstchild) {
    if(node->nextsibling) node->nextsibling->prevsibling=NULL;
    node->parent->firstchild=node->nextsibling;
  }
  else {
  // 4. the tree to delete has both prev- and next-sibling
    if(node->nextsibling) node->nextsibling->prevsibling=node->prevsibling;
    if(node->prevsibling) node->prevsibling->nextsibling=node->nextsibling;
  }                                                                170
  // End with physically deallocating the node
  delete node;
```

```
}

/*********************************************************
* Cuts the nodes childs e.g. the connection to them.
* It is espected that other pointers points to the
* subtrees which is cut off !
* Input:
* none                                                    180
* Output:
* node
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::cut_subtree(void) {
  firstchild=NULL;
  lastchild=NULL;
}
                                                          190
/*********************************************************
* Paste nodes from a tree to this
* Input:
* tree - tree to paste from
* Output:
* none
* Side Effects:
* The pointer connections are properly changed
*********************************************************/
void NaryTree::paste_subtree(const PNARYTREE tree) {      200
  firstchild=tree->firstchild;
  lastchild=tree->lastchild;
  PNARYTREE temp=tree->firstchild;
  while(temp) {
    temp->parent=this;
    temp=temp->nextsibling;
  }
}

/*********************************************************  210
* Compares two trees.
* Input:
* t1,t2 - the two trees to compare
* Output:
* BOOL - returns TRUE if the trees are identical
*********************************************************/
BOOL NaryTree::compare_trees(PNARYTREE t1,PNARYTREE t2) const {
  if(t1==NULL&&t2==NULL) return TRUE;
  if(t1==NULL||t2==NULL) return FALSE;
  if(compare(t1,t2)) {                                    220
    t1=t1->firstchild;
    t2=t2->firstchild;
    BOOL flag = TRUE;
    while(t1&&t2&&flag) {
      flag = compare_trees(t1,t2);
      t1=t1->nextsibling;
      t2=t2->nextsibling;
    }
    return flag && (t1==NULL) && (t2==NULL);
  }                                                       230
  return FALSE;
}

/*********************************************************
* Compares two nodes. For Nary trees this is always false
* because there is no node information !
* Input:
* t1,t2 - the trees (nodes) to compare
* Output:
```

```
* BOOL - returns TRUE if the trees (nodes) are identical                    240
*************************************************************/
BOOL NaryTree::compare_nodes(PNARYTREE t1,PNARYTREE t2) const {
  return t1->get_nodeid()==t2->get_nodeid();
}

/*************************************************************
* Counts the number of childnodes for a tree.
* The tree must not be NULL !
* Input:
* tree - the tree to count child for                                        250
* Output:
* unsigned int - number of childs
*************************************************************/
unsigned int NaryTree::count_childs(PNARYTREE tree) const {
  unsigned int n = 0;
  if(tree->firstchild) {
    n++;
    tree=tree->firstchild;
    while(tree->nextsibling) {
      tree=tree->nextsibling;                                               260
      n++;
    }
  }
  return n;
}

#endif /* NARYTREE_CPP */
```

G.5 NODEINFO.CPP

```
/*************************************************************************
*
* NODEINFO.CPP
*
* Implementation of RSL language features
*
* Created 6 January 1994, Michael Suodenjoki
*
*************************************************************************/
                                                                            10
#ifndef NODEINFO_CPP
#define NODEINFO_CPP

#include "RSL.H"
#include "LANGUAGE.H"
#include "NODEIDS.H"

const char *sc = "@";
const char *spc = "%@"; // Used to refer for lexical element
                                                                            20
// This nodename table connects nodeid's to nodename's and unparsing schemes
NODEINFO NodeInfoTable[NUMNODEIDS] = {
/* NID_IDNULL */               { "IdentifierNull", "%(placeholder%:id%)", NULL },
/* NID_ID */                   { "Identifier", sc, NULL },
/* NID_IDLIST2 */              { NULL, sc, "," },
/* NID_IDENTIFIER */           { "Identifier", spc, NULL },
/* NID_INTEGERNULL */          { "Integer", "%(placeholder%:integer%)", NULL },
/* NID_OPTCOMMENTSTRING */ { "Opt_Comment_String", sc, NULL },

/* NID_SPECIFICATION */        { "Specification", sc, NULL },                30
/* NID_MODULEDECLSTRING */ { "Module_Decl_String", sc, " " },
```

```
/* NID_MODULEDECL */          { "Module_Decl", sc, NULL },
/* NID_MODULEDECLNULL */      { NULL, "%(placeholder%:module_decl%)", NULL },
/* NID_DECL */                { "Decl", sc, NULL },
/* NID_OPTDECLSTRING */       { "Opt_Decl_String", sc, NULL },
/* NID_DECLNULL */            { NULL, "%(placeholder%:decl%)", NULL },

/* NID_SCHEMEDECL */          { "Scheme_Decl", "%(keyword%:scheme%)%t%n%b", NULL },
/* NID_SCHEMEDEF */           { "Scheme_Def", "%c@ = ", NULL },                                  40
/* NID_SCHEMEDEFLIST */       { "Scheme_Def_List", sc, ",%n" },
/* NID_FORMALSCHEMEPARAMETER */ { "Formal_Scheme_Parameter", "(  )", NULL },
/* NID_OPTFORMALSCHEMEPARAMETER */{ "Opt_Formal_Scheme_Parameter", sc, NULL },

/* NID_OBJECTDECL */          { "Object_Decl", "%(keyword%:object%) %t%n%b", NULL },
/* NID_OBJECTDEF */           { "Object_Def", "%c@ : ", NULL },
/* NID_OBJECTDEFLIST */       { NULL, sc, ",%n" },
/* NID_FORMALARRAYPARAMETER */ { "Formal_Array_Parameter", "[ ]", NULL },
/* NID_OPTFORMALARRAYPARAMETER */{ "Opt_Formal_Array_Parameter", sc, NULL  },
                                                                                                  50
/* NID_TYPEDECL */            { "Type_Decl", "%(keyword%:type%) %t%n%b", NULL },
/* NID_TYPEDEF */             { "Type_Def", sc, NULL },
/* NID_TYPEDEFNULL */         { NULL, "%(placeholder%:type_def%)", NULL },
/* NID_TYPEDEFLIST */         { NULL, sc, ",%n" },
/* NID_SORTDEF */             { "Sort_Def", "", NULL },
/* NID_VARIANTDEF */          { "Variant_Def", " == %[%]", NULL },
/* NID_VARIANT */             { "Variant", "%c", NULL },
/* NID_VARIANTNULL */         { NULL, "%(placeholder%:variant%)", NULL },
/* NID_VARIANTCHOICE */       { "Variant_Choice", sc, " | " },
/* NID_RECORDVARIANT */       { "Record_Variant", " (  )" },                                     60
/* NID_COMPONENTKIND */       { "Component_Kind", "  " },
/* NID_COMPONENTKINDLIST */   { "Component_Kind", sc, "," },
/* NID_COMPONENTKINDSTRING */ { NULL, sc, " " },
/* NID_CONSTRUCTOR */         { "Constructor", sc, NULL },
/* NID_CONSTRUCTORNULL */     { NULL, "%(placeholder%:constructor%)", NULL },
/* NID_DESTRUCTOR */          { "Destructor", " :", NULL },
/* NID_OPTDESTRUCTOR */       { "Opt_Destructor", sc, NULL },
/* NID_RECONSTRUCTOR */       { "Reconstructor","%(symbol%:<->%)  ", NULL },
/* NID_OPTRECONSTRUCTOR */    { "Opt_Reconstructor", sc, NULL },
/* NID_UNIONDEF */            { "Union_Def", "@ = %t%b", NULL },                                 70
/* NID_NAMEORWILDCARD */      { "Name_Or_Wildcard", "", NULL },
/* NID_NAMEORWILDCARDNULL */  { NULL, "%(placeholder%:name_or_wildcard%)", NULL },
/* NID_NAMEORWILDCARDCHOICE2 */ { NULL, sc, " | " },
/* NID_SHORTRECORDDEF */      { "Short_Record_Def", "@ :: %t%b", NULL },
/* NID_ABBREVIATIONDEF */     { "Abbreviation_Def", " = %t%b", NULL },

/* NID_VALUEDECL */           { "Value_Decl", "%(keyword%:value%)%t%n%b", NULL },
/* NID_VALUEDEF */            { "Value_Def", sc, NULL },
/* NID_VALUEDEFNULL */        { NULL, "%(placeholder%:value_def%)", NULL },
/* NID_VALUEDEFLIST */        { NULL, sc, ",%n" },                                               80
/* NID_EXPLICITVALUEDEF */    { "Explicit_Value_Def", " = ", NULL },
/* NID_IMPLICITVALUEDEF */    { "Implicit_Value_Def", "@ ", NULL },
/* NID_EXPLICITFUNCTIONDEF */ { "Explicit_Function_Def", "%n %(symbol%:is%) %[%c %c%c%]", NULL },
/* NID_FORMALFUNCTIONAPPLICATION */{ "Formal_Function_Application", sc, NULL },
/* NID_FORMALFUNCTIONAPPLICATIONNULL */{ NULL, "%(placeholder%:formal_function_application%)", NULL },
/* NID_IDAPPLICATION */       { "Id_Application", "  ", NULL },
/* NID_FORMALFUNCTIONPARAMETER */{ "Formal_Function_Parameter", "(  )", NULL },
/* NID_FORMALFUNCTIONPARAMETERSTRING */ { NULL, sc, " " },
/* NID_PREFIXAPPLICATION */   { "Prefix_Application", " ", NULL },
/* NID_INFIXAPPLICATION */    { "Infix_Application", "  ", NULL },                               90
/* NID_IMPLICITFUNCTIONDEF */ { "Implicit_Function_Def", "@%n %t%b %t%n", NULL },

/* NID_VARIABLEDECL */        { "Variable_Decl", "%(keyword%:variable%) %t%n%b", NULL },
/* NID_VARIABLEDEF */         { "Variable_Def", sc, NULL },
/* NID_VARIABLEDEFNULL */     { NULL, "%(placeholder%:variable_def%)", NULL },
/* NID_VARIABLEDEFLIST */     { NULL, sc, ",%n" },
/* NID_SINGLEVARIABLEDEF */   { "Single_Variable_Def", "@ :  ", NULL },
/* NID_INITIALISATION */      { "Initialisation", " := ", NULL },
/* NID_OPTINITIALISATION */   { "Opt_Initialisation", sc, NULL },
```

```
/* NID_MULTIPLEVARIABLEDEF */   { "Multiple_Variable_Def", " : " },                                    100

/* NID_CHANNELDECL */          { "Channel_Decl", "%(keyword%:channel%) %t%n%b", NULL },
/* NID_CHANNELDEF */           { "Channel_Def", sc, NULL },
/* NID_CHANNELDEFNULL */       { NULL, "%(placeholder%:channel_def%)", NULL },
/* NID_CHANNELDEFLIST */       { NULL, sc, ",%n" },
/* NID_SINGLECHANNELDEF */     { "Single_Channel_Def", " : ", NULL },
/* NID_MULTIPLECHANNELDEF */   { "Multiple_Channel_Def", "@ : ", NULL },

/* NID_AXIOMDECL */            { "Axiom_Decl", "%(keyword%:axiom%) %t%n%b", NULL },
/* NID_AXIOMQUANTIFICATION */  { "Axiom_Quantification", "%(keyword%:forall%)  %(symbol%::-%)", NULL },  110
/* NID_OPTAXIOMQUANTIFICATION */{ "Opt_Axiom_Quantification", sc, NULL },
/* NID_AXIOMDEF */             { "Axiom_Def", "@", NULL },
/* NID_AXIOMDEFLIST */         { NULL, sc, ",%n" },
/* NID_AXIOMNAMING */          { "Axiom_Naming", "[ ] ", NULL },
/* NID_OPTAXIOMNAMING */       { "Opt_Axiom_Naming", sc, NULL },

/* NID_CLASSEXPR */            { "Class_Expr", sc, NULL },
/* NID_CLASSEXPRNULL */        { NULL, "%(placeholder%:class_expr%)", NULL },
/* NID_BASICCLASSEXPR */       { "Basic_Class_Expr",
  "%(keyword%:class%)%t@%b%n%(keyword%:end%)", NULL },                                                 120
/* NID_EXTENDINGCLASSEXPR */   { "Extending_Class_Expr",
  "%(keyword%:extend%) @ %(keyword%:with%) @", NULL },
/* NID_HIDINGCLASSEXPR */      { "Hiding_Class_Expr",
  "%(keyword%:hide%) @ %(keyword%:in%) @", NULL },
/* NID_RENAMINGCLASSEXPR */    { "Renaming_Class_Expr",
  "%(keyword%:use%) @ %(keyword%:in%) @", NULL },
/* NID_SCHEMEINSTANTIATION */  { "Scheme_Instantiation", " ", NULL },
/* NID_ACTUALSCHEMEPARAMETER */ { "Actual_Scheme_Parameter", "( )", NULL },
/* NID_OPTACTUALSCHEMEPARAMETER */{ "Opt_Actual_Scheme_Parameter", sc, NULL },
/* NID_RENAMEPAIR */           { "Rename_Pair", " %(keyword%:for%) ", NULL },                          130
/* NID_RENAMEPAIRLIST */       { NULL, sc, "," },
/* NID_DEFINEDITEM */          { "Defined_Item", sc, NULL },
/* NID_DEFINEDITEMNULL */      { NULL, "%(placeholder%:defined_item%)", NULL },
/* NID_DEFINEDITEMLIST */      { NULL, sc, "," },
/* NID_DISAMBIGUATEDITEM */    { "Disambiguated_Item", " : ", NULL },

/* NID_OBJECTEXPR */           { "Object_Expr", sc, NULL },
/* NID_OBJECTEXPRNULL */       { NULL, "%(placeholder%:object_expr%)", NULL },
/* NID_OBJECTEXPRLIST */       { NULL, sc, ",%n" },
/* NID_ELEMENTOBJECTEXPR */    { "Element_Object_Expr", " ", NULL },                                  140
/* NID_ACTUALARRAYPARAMETER */ { "Actual_Array_Parameter", "[ ]", NULL },
/* NID_ARRAYOBJECTEXPR */      { "Array_Object_Expr", "[| %(symbol%::-%) |]", NULL },
/* NID_FITTINGOBJECTEXPR */    { "Fitting_Object_Expr", " { }" },

/* NID_TYPEEXPR */             { "Type_Expr", "", NULL },
/* NID_TYPEEXPRNULL */         { NULL, "%(placeholder%:type_expr%)", NULL },
/* NID_TYPEEXPRPRODUCT2 */     { NULL, sc, " %(symbol%:><%) " },
/* NID_TYPELITERAL */          { "Type_Literal", sc, NULL },
/* NID_TYPELITERALNULL */      { NULL, "%(placeholder%:type_literal%)", NULL },
/* NID_PRODUCTTYPEEXPR */      { "Product_Type_Expr", "%[%c%c%]", NULL },                             150
/* NID_SETTYPEEXPR */          { "Set_Type_Expr", sc, NULL },
/* NID_SETTYPEEXPRNULL */      { NULL, "%(placeholder%:set_type_expr%)", NULL },
/* NID_FINITESETTYPEEXPR */    { "Finite_Set_Type_Expr", "%(keyword%:-set%)", NULL },
/* NID_INFINITESETTYPEEXPR */  { "Infinite_Set_Type_Expr", "%(keyword%:-infset%)", NULL },
/* NID_LISTTYPEEXPR */         { "List_Type_Expr", sc, NULL },
/* NID_LISTTYPEEXPRNULL */     { NULL, "%(placeholder%:list_type_expr%)", NULL },
/* NID_FINITELISTTYPEEXPR */   { "Finite_List_Type_Expr", "%(symbol%:-list%)", NULL },
/* NID_INFINITELISTTYPEEXPR */ { "Infinite_List_Type_Expr", "%(symbol%:-inflist%)", NULL },
/* NID_MAPTYPEEXPR */          { "Map_Type_Expr", " %(symbol%:-m->%) ", NULL },
/* NID_FUNCTIONTYPEEXPR */     { "Function_Type_Expr", " ", NULL },                                   160
/* NID_FUNCTIONARROW */        { "Function_Arrow", sc, NULL },
/* NID_FUNCTIONARROWNULL */    { NULL, "%(placeholder%:function_arrow%)", NULL },
/* NID_RESULTDESC */           { "Result_Desc", " ", NULL },
/* NID_SUBTYPEEXPR */          { "Subtype_Expr", "{| |}", NULL },
/* NID_BRACKETEDTYPEEXPR */    { "Bracketed_Type_Expr", "( )", NULL },
/* NID_ACCESSDESC */           { "Access_Desc", " ", NULL },
```

```
/* NID_OPTACCESSDESCSTRING */  { "Opt_Access_Desc_String", sc, " " },
/* NID_ACCESSMODE */           { "Access_Mode", sc, NULL },
/* NID_ACCESSMODENULL */       { NULL, "%(placeholder%:access_mode%)", NULL },
/* NID_ACCESS */               { "Access", sc, NULL },                                  170
/* NID_ACCESSNULL */           { NULL, "%(placeholder%:access%)", NULL },
/* NID_ACCESSLIST */           { NULL, sc, ",%n" },
/* NID_OPTACCESSLIST */        { NULL, sc, NULL },
/* NID_ENUMERATEDACCESS */     { "Enumerated_Access", "{  }", NULL },
/* NID_COMPLETEDACCESS */      { "Completed_Access", " %(keyword%:any%)", NULL },
/* NID_COMPREHENDEDACCESS */   { "Comprehended_Access", "{  |  }", NULL },

/* NID_VALUEEXPR */           { "Value_Expr", "", NULL },
/* NID_VALUEEXPRNULL */       { NULL, "%(placeholder%:value_expr%)", NULL },
/* NID_VALUEEXPRLIST */       { NULL, sc, ",%n" },                                     180
/* NID_OPTVALUEEXPRLIST */    { NULL, sc, NULL },
/* NID_VALUEEXPRLIST2 */      { NULL, sc, "," },
/* NID_VALUELITERAL */        { "Value_Literal", sc, NULL },
/* NID_VALUELITERALNULL */    { NULL, "%(placeholder%:value_literal%)", NULL },
/* NID_UNITLITERAL */         { "Unit_Literal", "()", NULL },
/* NID_BOOLLITERAL */         { "Bool_Literal", sc, NULL },
/* NID_BOOLLITERALNULL */     { NULL, "%(placeholder%:bool_literal%)", NULL },
/* NID_PRENAME */             { "Pre_Name", "`", NULL },
/* NID_BASICEXPR */           { "Basic_Expr", sc, NULL },
/* NID_BASICEXPRNULL */       { NULL, "%(placeholder%:basic_expr%)", NULL },           190
/* NID_PRODUCTEXPR */         { "Product_Expr", "(  )", NULL },
/* NID_SETEXPR */             { "Set_Expr", sc, NULL },
/* NID_SETEXPRNULL */         { NULL, "%(placeholder%:set_expr%)", NULL },
/* NID_RANGEDSETEXPR */       { "Ranged_Set_Expr", "{  ..  }", NULL },
/* NID_ENUMERATEDSETEXPR */   { "Enumerated_Set_Expr", "{  }", NULL },
/* NID_COMPREHENDEDSETEXPR */ { "Comprehended_Set_Expr", "{  |  }", NULL },
/* NID_SETLIMITATION */       { "Set_Limitation", "  ", NULL },
/* NID_RESTRICTION */         { "Restriction", "%(symbol%:-%) ", NULL },
/* NID_OPTRESTRICTION */      { NULL, sc, NULL },
/* NID_LISTEXPR */            { "List_Expr", sc, NULL },                               200
/* NID_LISTEXPRNULL */        { NULL, "%(placeholder%:list_expr%)", NULL },
/* NID_RANGEDLISTEXPR */      { "Ranged_List_Expr", "%(symbol%:<.%)  ..  %(symbol%:.>%)", NULL },
/* NID_ENUMERATEDLISTEXPR */  { "Enumerated_List_Expr", "%(symbol%:<.%)  %(symbol%:.>%)", NULL },
/* NID_COMPREHENDEDLISTEXPR */ { "Comprehended_List_Expr", "%(symbol%:<.%)  |  %(symbol%:.>%)", NULL },
/* NID_LISTLIMITATION */      { "List_Limitation", "  %(keyword%:in%)  ", NULL },
/* NID_MAPEXPR */             { "Map_Expr", sc, NULL },
/* NID_MAPEXPRNULL */         { NULL, "%(placeholder%:map_expr%)", NULL },
/* NID_ENUMERATEDMAPEXPR */   { "Enumerated_Map_Expr", "[  ]", NULL },
/* NID_VALUEEXPRPAIR */       { "Value_Expr_Pair", "  %(symbol%:+>%)  ", NULL },
/* NID_OPTVALUEEXPRPAIRLIST */ { NULL, sc, NULL },                                     210
/* NID_COMPREHENDEDMAPEXPR */ { "Comprehended_Expr", "[  |  ]", NULL },
/* NID_FUNCTIONEXPR */        { "Function_Expr", "%(symbol%:-\\%)  %(symbol%::-%)  ", NULL },
/* NID_LAMBDAPARAMETER */     { "Lambda_Parameter", sc, NULL },
/* NID_LAMBDAPARAMETERNULL */ { NULL, "%(placeholder%:lambda_parameter%)", NULL },
/* NID_LAMBDATYPING */        { "Lambda_Typing", "(  )", NULL },
/* NID_APPLICATIONEXPR */     { "Application_Expr", " ", NULL },
/* NID_ACTUALFUNCTIONPARAMETER */{ "Actual_Function_Parameter", "()", NULL },
/* NID_ACTUALFUNCTIONPARAMETERSTRING */{ NULL, sc, "" },
/* NID_QUANTIFIEDEXPR */      { "Quantified_Expr", "  ", NULL },
/* NID_QUANTIFIER */          { "Quantifier", sc, NULL },                              220
/* NID_QUANTIFIERNULL */      { NULL, "%(placeholder%:quantifier%)", NULL },
/* NID_EQUIVALENCEEXPR */     { "Equivalence_Expr", " %(symbol%:is%) @" },
/* NID_PRECONDITION */        { "Pre_Condition", "%(keyword%:pre%) ", NULL },
/* NID_OPTPRECONDITION */     { "Opt_Pre_Condition", sc, NULL },
/* NID_POSTEXPR */            { "Post_Expr", "%t@%b", NULL },
/* NID_POSTCONDITION */       { "Post_Condition", "  %t%b%(keyword%:post%)  ", NULL },
/* NID_RESULTNAMING */        { "Result_Naming", "%(keyword%:as%) ", NULL },
/* NID_OPTRESULTNAMING */     { NULL, sc, NULL },
/* NID_DISAMBIGUATEDEXPR */   { "Disambiguated_Expr", " : ", NULL },
/* NID_BRACKETEDEXPR */       { "Bracketed_Expr", "(  )", NULL },                      230
/* NID_INFIXEXPR */           { "Infix_Expr", sc, NULL },
/* NID_INFIXEXPRNULL */       { NULL, "%(placeholder%:infix_expr%)", NULL },
/* NID_STMTINFIXEXPR */       { "Stmt_Infix_Expr", "  ", NULL },
```

```
/* NID_AXIOMINFIXEXPR */        { "Axiom_Infix_Expr", "  ", NULL },
/* NID_VALUEINFIXEXPR */        { "Value_Infix_Expr", "  ", NULL },
/* NID_PREFIXEXPR */            { "Prefix_Expr", sc, NULL },
/* NID_PREFIXEXPRNULL */        { NULL, "%(placeholder%:prefix_expr%)", NULL },
/* NID_AXIOMPREFIXEXPR */       { "Atom_Prefix_Expr", " ", NULL },
/* NID_UNIVERSALPREFIXEXPR */   { "Universal_Prefix_Expr", "%(symbol%:always%)  ", NULL },
/* NID_VALUEPREFIXEXPR */       { "Value_Prefix_Expr", "  ", NULL },                    240
/* NID_COMPREHENDEDEXPR */      { "Comprehended_Expr", "  {  |  }", NULL },
/* NID_INITIALISEEXPR */        { "Initialise_Expr", " %(keyword%:initialise%)", NULL },
/* NID_ASSIGNMENTEXPR */        { "Assignment_Expr", "  :=  ", NULL },
/* NID_INPUTEXPR */             { "Input_Expr", "  ?", NULL },
/* NID_OUTPUTEXPR */            { "Output_Expr", "  !  ", NULL },
/* NID_STRUCTUREDEXPR */        { "Structured_Expr", sc, NULL },
/* NID_STRUCTUREDEXPRNULL */    { NULL, "%(placeholder%:structured_expr%)", NULL },
/* NID_LOCALEXPR */             { "Local_Expr",
  "%(keyword%:local%)%t%b%n%(keyword%:in%)%t%n%b%n%(keyword%:end%)", NULL },
/* NID_LETEXPR */               { "Let_Expr",                                          250
  "%(keyword%:let%)  %(keyword%:in%)  %(keyword%:end%)", NULL },
/* NID_LETDEF */                { "Let_Def", sc, NULL },
/* NID_LETDEFNULL */            { NULL, "%(placeholder%:let_def%)", NULL },
/* NID_LETDEFLIST */            { NULL, sc, "," },
/* NID_EXPLICITLET */           { "Explicit_Let", " = ", NULL },
/* NID_IMPLICITLET */           { "Implicit_Let", "  ", NULL },
/* NID_LETBINDING */            { "Let_Binding", sc, NULL },
/* NID_IFEXPR */                { "If_Expr",
  "%(keyword%:if%)  %(keyword%:then%) %{%t%c%b%c%} %{%c%}%(keyword%:end%)", NULL },
/* NID_ELSIFBRANCH */           { "Elsif_Branch",                                      260
  "%c%(keyword%:elsif%)  %(keyword%:then%)  ", NULL },
/* NID_OPTELSIFBRANCHSTRING */  { NULL, sc, NULL },
/* NID_ELSEBRANCH */            { "Else_Branch", "%c%(keyword%:else%)  ", NULL },
/* NID_OPTELSEBRANCH */         { NULL ,sc, NULL },
/* NID_CASEEXPR */              { "Case_Expr",
  "%(keyword%:case%) @ %(keyword%:of%) %{%t%c@ %b%c%}%(keyword%:end%)", NULL },
/* NID_CASEBRANCH */            { "Case_Branch", " %(symbol%:->%)  ", NULL },
/* NID_CASEBRANCHLIST */        { NULL, sc, ",%c" },
/* NID_WHILEEXPR */             { "While_Expr",
  "%(keyword%:while%) @ %(keyword%:do%) %{%t%c@ %b%c%}%(keyword%:end%)", NULL },       270
/* NID_UNTILEXPR */             { "Until_Expr",
  "%(keyword%:do%) %{%t%c@ %b%c%}%(keyword%:until%) @ %(keyword%:end%)", NULL },
/* NID_FOREXPR */               { "For_Expr",
  "%(keyword%:for%) @ %(keyword%:do%) %{%t%c@ %b%c%}%(keyword%:end%)", NULL },

/* NID_BINDING */               { "Binding", sc, NULL },
/* NID_BINDINGNULL */           { NULL, "%(placeholder%:binding%)", NULL },
/* NID_BINDINGLIST2 */          { NULL, sc, "," },
/* NID_OPTBINDINGLIST */        { NULL, sc, NULL },
/* NID_PRODUCTBINDING */        { "Product_Binding", "(  )", NULL },                   280

/* NID_TYPING */                { "Typing", sc, NULL },
/* NID_TYPINGNULL */            { NULL, "%(placeholder%:typing%)", NULL },
/* NID_TYPINGLIST */            { NULL, sc, "," },
/* NID_OPTTYPINGLIST */         { NULL, sc, NULL },
/* NID_SINGLETYPING */          { "Single_Typing", " : ", NULL },
/* NID_MULTIPLETYPING */        { "Multiple_Typing", " : ", NULL },
/* NID_COMMENTEDTYPING */       { "Commented_Typing", "", NULL },

/* NID_PATTERN */               { "Pattern", sc, NULL },                               290
/* NID_PATTERNNULL */           { NULL, "%(placeholder%:pattern%)", NULL },
/* NID_WILDCARDPATTERN */       { "Wildcard_Pattern", "_", NULL },
/* NID_PRODUCTPATTERN */        { "Product_Pattern", "(  )", NULL },
/* NID_RECORDPATTERN */         { "Record_Pattern", "  (  )", NULL },
/* NID_LISTPATTERN */           { "List_Pattern", sc, NULL },
/* NID_LISTPATTERNNULL */       { NULL, "%(placeholder%:list_pattern%)", NULL },
/* NID_ENUMERATEDLISTPATTERN */ { "Enumerated_List_Pattern",
  "%(symbol%:<.%) @ %(symbol%:.>%)", NULL },
/* NID_INNERPATTERN */          { "Inner_Pattern", sc, NULL },
/* NID_INNERPATTERNNULL */      { NULL, "%(placeholder%:inner_pattern%)", NULL },      300
```

```
/* NID_INNERPATTERNLIST */   { NULL, sc, "," },
/* NID_OPTINNERPATTERNLIST */  { NULL, sc, NULL },
/* NID_INNERPATTERNLIST2 */  { NULL, sc, "," },
/* NID_EQUALITYPATTERN */  { "Equality_Pattern", "= ", NULL },
/* NID_NAME */                { "Name", sc, NULL },
/* NID_NAMENULL */            { NULL, "%(placeholder%:name%)", NULL },
/* NID_QUALIFIEDID */         { "Qualified_Id", " ", NULL },
/* NID_QUALIFICATION */       { "Qualification", sc, NULL },
/* NID_OPTQUALIFICATION */    { NULL, sc, NULL },
/* NID_QUALIFIEDOP */         { "Qualified_Op", " (  )", NULL },                    310
/* NID_IDOROP */              { "Id_Or_Op", sc, NULL },
/* NID_IDOROPNULL */          { NULL, "%(placeholder%:id_or_op%)", NULL },
/* NID_OP */                  { "Op", sc, NULL },
/* NID_OPNULL */              { NULL, "%(placeholder%:op%)", NULL },
/* NID_INFIXOP */             { "Infix_Op", sc, NULL },
/* NID_INFIXOPNULL */         { NULL, "%(placeholder%:infix_op%)", NULL },
/* NID_PREFIXOP */            { "Prefix_Op", sc, NULL },
/* NID_PREFIXOPNULL */        { NULL, "%(placeholder%:prefix_op%)", NULL },
/* NID_CONNECTIVE */          { "Connective", sc, NULL },
/* NID_CONNECTIVENULL */      { NULL, "%(placeholder%:connective%)", NULL },       320
/* NID_INFIXCONNECTIVE */     { "Infix_Connective", sc, NULL },
/* NID_INFIXCONNECTIVENULL */  { NULL, "%(placeholder%:infix_connective%)", NULL },
/* NID_PREFIXCONNECTIVE */    { "Prefix_Connective", "~", NULL },
/* NID_INFIXCOMBINATOR */     { "Infix_Combinator", sc, NULL },
/* NID_INFIXCOMBINATORNULL */  { NULL, "%(placeholder%:infix_combinator%)", NULL },

// Terminals
/* NID_WILDCARD */      { "Wildcard", "_", NULL },
/* NID_TUNIT */         { NULL, "%(keyword%:Unit%)", NULL },
/* NID_TBOOL */         { NULL, "%(keyword%:Bool%)", NULL },                  330
/* NID_TINT */          { NULL, "%(keyword%:Int%)", NULL },
/* NID_TNAT */          { NULL, "%(keyword%:Nat%)", NULL },
/* NID_TREAL */         { NULL, "%(keyword%:Real%)", NULL },
/* NID_TTEXT */         { NULL, "%(keyword%:Text%)", NULL },
/* NID_TCHAR */         { NULL, "%(keyword%:Char%)", NULL },
/* NID_PARRIGHTARROW */ { NULL, "%(symbol%:-~->%)", NULL },
/* NID_RIGHTARROW */    { NULL, "%(symbol%:->%)", NULL },
/* NID_READ */          { NULL, "%(keyword%:read%)", NULL },
/* NID_WRITE */         { NULL, "%(keyword%:write%)", NULL },
/* NID_IN */            { NULL, "%(keyword%:in%)", NULL },                    340
/* NID_OUT */           { NULL, "%(keyword%:out%)", NULL },
/* NID_TRUE */          { NULL, "%(keyword%:true%)", NULL },
/* NID_FALSE */         { NULL, "%(keyword%:false%)", NULL },
/* NID_CHAOS */         { NULL, "%(keyword%:chaos%)", NULL },
/* NID_SKIP */          { NULL, "%(keyword%:skip%)", NULL },
/* NID_STOP */          { NULL, "%(keyword%:stop%)", NULL },
/* NID_SWAP */          { NULL, "%(keyword%:swap%)", NULL },
/* NID_FORALL */        { NULL, "%(symbol%:all%)", NULL },
/* NID_EXISTS */        { NULL, "%(symbol%:exists%)", NULL },
/* NID_EXISTSUNIQUE */  { NULL, "%(symbol%:exists%)!", NULL },                 350
/* NID_EQUAL */         { NULL, "=", NULL },
/* NID_NOTEQUAL */      { NULL, "%(symbol%:~=%)", NULL },
/* NID_GT */            { NULL, ">", NULL },
/* NID_LT */            { NULL, "<", NULL },
/* NID_GE */            { NULL, "%(symbol%:>=%)", NULL },
/* NID_LE */            { NULL, "%(symbol%:<=%)", NULL },
/* NID_SUPERSET */      { NULL, "%(symbol%:>>%)", NULL },
/* NID_SUBSET */        { NULL, "%(symbol%:<<%)", NULL },
/* NID_PROPERSUPERSET */{ NULL, "%(symbol%:>>=%)", NULL },
/* NID_PROPERSUBSET */  { NULL, "%(symbol%:<<=%)", NULL },                    360
/* NID_ISIN */          { NULL, "%(symbol%:isin%)", NULL },
/* NID_NOTISIN */       { NULL, "%(symbol%:~isin%)", NULL },
/* NID_ADDITION */      { NULL, "+", NULL },
/* NID_SUBTRACTION */   { NULL, "-", NULL },
/* NID_BACKSLASH */     { NULL, "\\", NULL },
/* NID_CONCAT */        { NULL, "^", NULL },
/* NID_UNION */         { NULL, "%(symbol%:union%)", NULL },
```

```
/* NID_OVERRIDE */      { NULL, "%(symbol%:!!%)", NULL },
/* NID_MULTIPLICATION */{ NULL, "*", NULL },
/* NID_DIVISION */      { NULL, "/", NULL },                                    370
/* NID_COMPOSITION */   { NULL, "%(symbol%:#%)", NULL },
/* NID_INTERSECTION */  { NULL, "%(symbol%:inter%)", NULL },
/* NID_EXPONENTIATION */{ NULL, "%(symbol%:**%)", NULL },
/* NID_ABS */           { NULL, "%(keyword%:abs%)", NULL },
/* NID_INT */           { NULL, "%(keyword%:int%)", NULL },
/* NID_REAL */          { NULL, "%(keyword%:real%)", NULL },
/* NID_CARD */          { NULL, "%(keyword%:card%)", NULL },
/* NID_LEN */           { NULL, "%(keyword%:len%)", NULL },
/* NID_INDS */          { NULL, "%(keyword%:inds%)", NULL },
/* NID_ELEMS */         { NULL, "%(keyword%:elems%)", NULL },                   380
/* NID_HD */            { NULL, "%(keyword%:hd%)", NULL },
/* NID_TL */            { NULL, "%(keyword%:tl%)", NULL },
/* NID_DOM */           { NULL, "%(keyword%:dom%)", NULL },
/* NID_RNG */           { NULL, "%(keyword%rng%)", NULL },
/* NID_IMPLICATION */   { NULL, "%(symbol%:=>%)", NULL },
/* NID_OR */            { NULL, "%(symbol%:\\/%)", NULL },
/* NID_AND */           { NULL, "%(symbol%:/\\%)", NULL },
/* NID_NOT */           { NULL, "~", NULL },
/* NID_EXTERNALCHOICE */{ NULL, "%(symbol%:|=|%)", NULL },
/* NID_INTERNALCHOICE */{ NULL, "%(symbol%:|^|%)", NULL },                     390
/* NID_CONCURRENTCOMPOSITION */{ NULL, "%(symbol%:||%)", NULL },
/* NID_INTERLOCKEDCOMPOSITION */{ NULL, "%(symbol%:++%)", NULL },
/* NID_SEQUENTIALCOMPOSITION */{ NULL, ";", NULL },

// Forgets !
/* NID_CONCATENATEDLISTPATTERN */{ "Concatenated_List_Pattern", " ^ ", NULL },
/* NID_LETBINDINGNULL */        { NULL, "%(placeholder%:let_binding%)", NULL },
/* NID_COMMENT */               { "Comment", sc, NULL },
/* NID_INTLITERAL */            { "Integer", spc, NULL },
/* NID_REALLITERAL */           { "Real", spc, NULL },                          400
/* NID_TEXTLITERAL */           { "Text", spc, NULL },
/* NID_CHARLITERAL */           { "Char", spc, NULL },
/* NID_COMMENTNULL */           { NULL, "%(placeholder%:comment%)", NULL },
/* NID_COMMENTSTRING */         { NULL, "%n", NULL },
/* NID_ELSIFBRANCHNULL */   { NULL, NULL, NULL },
/* NID_ELSIFBRANCHSTRING */   { NULL, sc, NULL },
/* NID_VALUEEXPRPAIRNULL */   { NULL, NULL, NULL },
/* NID_VALUEEXPRPAIRLIST */   { NULL, sc, NULL },
/* NID_ACCESSDESCNULL */    { NULL, NULL, NULL },
/* NID_ACCESSDESCSTRING */  { NULL, sc, "%n" },                               410
/* NID_DECLSTRING */          { NULL, "%n", "%n" },
/* NID_BINDINGLIST */           { NULL, sc, "," },
/* NID_RIGHTLISTPATTERN */      { "Right_List_Pattern", " ^ ", NULL },
/* NID_COMMENTTOKENNULL */      { NULL, "%(placeholder%:comment_token%)", NULL },
/* NID_COMMENTTOKEN */          { "Comment_Token", sc, NULL },
/* NID_COMMENTTOKENSTRING */    { NULL, "/* */", " " },
/* NID_COMMENTSPACE */          { NULL, "<cs>", NULL },
/* NID_COMMENTNEWLINE */        { NULL, "%n", NULL },
/* NID_COMMENTTEXT */           { "Comment_Text", "%@", NULL }};
                                                                               420
TRANSFORMATION TransformationTable[NUMTRANS] = {
  { NID_MODULEDECL,     NodeInfoTable[NID_SCHEMEDECL].name },
  { NID_MODULEDECL,     NodeInfoTable[NID_OBJECTDECL].name },
  { NID_DECL,           NodeInfoTable[NID_SCHEMEDECL].name },
  { NID_DECL,           NodeInfoTable[NID_OBJECTDECL].name },
  { NID_DECL,           NodeInfoTable[NID_TYPEDECL].name },
  { NID_DECL,           NodeInfoTable[NID_VALUEDECL].name  },
  { NID_DECL,           NodeInfoTable[NID_VARIABLEDECL].name },
  { NID_DECL,           NodeInfoTable[NID_CHANNELDECL].name  },
  { NID_DECL,           NodeInfoTable[NID_AXIOMDECL].name  },                  430
  { NID_TYPEDEF,        NodeInfoTable[NID_SORTDEF].name },
  { NID_TYPEDEF,        NodeInfoTable[NID_VARIANTDEF].name },
  { NID_TYPEDEF,        NodeInfoTable[NID_UNIONDEF].name },
  { NID_TYPEDEF,        NodeInfoTable[NID_SHORTRECORDDEF].name },
```

```
{ NID_TYPEDEF,         NodeInfoTable[NID_ABBREVIATIONDEF].name },
{ NID_VARIANT,         NodeInfoTable[NID_CONSTRUCTOR].name },
{ NID_VARIANT,         NodeInfoTable[NID_RECORDVARIANT].name },
{ NID_CONSTRUCTOR,     NodeInfoTable[NID_IDOROP].name },
{ NID_CONSTRUCTOR,     NodeInfoTable[NID_WILDCARD].name },
{ NID_NAMEORWILDCARD, NodeInfoTable[NID_NAME].name },                                    440
{ NID_NAMEORWILDCARD, NodeInfoTable[NID_WILDCARD].name },
{ NID_VALUEDEF,        NodeInfoTable[NID_COMMENTEDTYPING].name },
{ NID_VALUEDEF,        NodeInfoTable[NID_EXPLICITVALUEDEF].name },
{ NID_VALUEDEF,        NodeInfoTable[NID_IMPLICITVALUEDEF].name },
{ NID_VALUEDEF,        NodeInfoTable[NID_EXPLICITFUNCTIONDEF].name },
{ NID_VALUEDEF,        NodeInfoTable[NID_IMPLICITFUNCTIONDEF].name },
{ NID_FORMALFUNCTIONAPPLICATION,      NodeInfoTable[NID_IDAPPLICATION].name },
{ NID_FORMALFUNCTIONAPPLICATION,      NodeInfoTable[NID_PREFIXAPPLICATION].name },
{ NID_FORMALFUNCTIONAPPLICATION,      NodeInfoTable[NID_INFIXAPPLICATION].name },
{ NID_VARIABLEDEF,     NodeInfoTable[NID_SINGLEVARIABLEDEF].name },                      450
{ NID_VARIABLEDEF,     NodeInfoTable[NID_MULTIPLEVARIABLEDEF].name },
{ NID_CHANNELDEF,      NodeInfoTable[NID_SINGLECHANNELDEF].name },
{ NID_CHANNELDEF,      NodeInfoTable[NID_MULTIPLECHANNELDEF].name },
{ NID_CLASSEXPR,       NodeInfoTable[NID_BASICCLASSEXPR].name },
{ NID_CLASSEXPR,       NodeInfoTable[NID_EXTENDINGCLASSEXPR].name },
{ NID_CLASSEXPR,       NodeInfoTable[NID_HIDINGCLASSEXPR].name },
{ NID_CLASSEXPR,       NodeInfoTable[NID_RENAMINGCLASSEXPR].name },
{ NID_CLASSEXPR,       NodeInfoTable[NID_SCHEMEINSTANTIATION].name },
{ NID_DEFINEDITEM,     NodeInfoTable[NID_IDOROP].name },
{ NID_DEFINEDITEM,     NodeInfoTable[NID_DISAMBIGUATEDITEM].name },                      460
{ NID_OBJECTEXPR,      NodeInfoTable[NID_NAME].name },
{ NID_OBJECTEXPR,      NodeInfoTable[NID_ELEMENTOBJECTEXPR].name },
{ NID_OBJECTEXPR,      NodeInfoTable[NID_ARRAYOBJECTEXPR].name },
{ NID_OBJECTEXPR,      NodeInfoTable[NID_FITTINGOBJECTEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_TYPELITERAL].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_NAME].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_PRODUCTTYPEEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_SETTYPEEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_LISTTYPEEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_MAPTYPEEXPR].name },                            470
{ NID_TYPEEXPR,        NodeInfoTable[NID_FUNCTIONTYPEEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_SUBTYPEEXPR].name },
{ NID_TYPEEXPR,        NodeInfoTable[NID_BRACKETEDTYPEEXPR].name },
{ NID_TYPELITERAL,     "Unit" },
{ NID_TYPELITERAL,     "Bool" },
{ NID_TYPELITERAL,     "Int" },
{ NID_TYPELITERAL,     "Nat" },
{ NID_TYPELITERAL,     "Real" },
{ NID_TYPELITERAL,     "Text" },
{ NID_TYPELITERAL,     "Char" },                                                         480
{ NID_SETTYPEEXPR,     NodeInfoTable[NID_FINITESETTYPEEXPR].name },
{ NID_SETTYPEEXPR,     NodeInfoTable[NID_INFINITESETTYPEEXPR].name },
{ NID_LISTTYPEEXPR,    NodeInfoTable[NID_FINITELISTTYPEEXPR].name },
{ NID_LISTTYPEEXPR,    NodeInfoTable[NID_INFINITELISTTYPEEXPR].name },
{ NID_FUNCTIONARROW,   "-~->" },
{ NID_FUNCTIONARROW,   "->" },
{ NID_ACCESSMODE,      "Read" },
{ NID_ACCESSMODE,      "Write" },
{ NID_ACCESSMODE,      "In" },
{ NID_ACCESSMODE,      "Out" },                                                          490
{ NID_ACCESS,          NodeInfoTable[NID_NAME].name },
{ NID_ACCESS,          NodeInfoTable[NID_ENUMERATEDACCESS].name },
{ NID_ACCESS,          NodeInfoTable[NID_COMPLETEDACCESS].name },
{ NID_ACCESS,          NodeInfoTable[NID_COMPREHENDEDACCESS].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_VALUELITERAL].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_NAME].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_PRENAME].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_BASICEXPR].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_PRODUCTEXPR].name },
{ NID_VALUEEXPR,       NodeInfoTable[NID_SETEXPR].name },                                500
{ NID_VALUEEXPR,       NodeInfoTable[NID_LISTEXPR].name },
```

```
{ NID_VALUEEXPR,      NodeInfoTable[NID_MAPEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_FUNCTIONEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_APPLICATIONEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_QUANTIFIEDEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_EQUIVALENCEEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_POSTEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_DISAMBIGUATEDEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_BRACKETEDEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_INFIXEXPR].name },                    510
{ NID_VALUEEXPR,      NodeInfoTable[NID_PREFIXEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_COMPREHENDEDEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_INITIALISEEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_ASSIGNMENTEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_INPUTEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_OUTPUTEXPR].name },
{ NID_VALUEEXPR,      NodeInfoTable[NID_STRUCTUREDEXPR].name },
{ NID_VALUELITERAL,   NodeInfoTable[NID_UNITLITERAL].name },
{ NID_VALUELITERAL,   NodeInfoTable[NID_BOOLLITERAL].name },
{ NID_VALUELITERAL,   NodeInfoTable[NID_INTLITERAL].name },                   520
{ NID_VALUELITERAL,   NodeInfoTable[NID_REALLITERAL].name },
{ NID_VALUELITERAL,   NodeInfoTable[NID_TEXTLITERAL].name },
{ NID_VALUELITERAL,   NodeInfoTable[NID_CHARLITERAL].name },
{ NID_BOOLLITERAL,    "True" },
{ NID_BOOLLITERAL,    "False" },
{ NID_BASICEXPR,      "Chaos" },
{ NID_BASICEXPR,      "Skip" },
{ NID_BASICEXPR,      "Stop" },
{ NID_BASICEXPR,      "Swap" },
{ NID_SETEXPR,        NodeInfoTable[NID_RANGEDSETEXPR].name },                530
{ NID_SETEXPR,        NodeInfoTable[NID_ENUMERATEDSETEXPR].name },
{ NID_SETEXPR,        NodeInfoTable[NID_COMPREHENDEDSETEXPR].name },
{ NID_LISTEXPR,       NodeInfoTable[NID_RANGEDLISTEXPR].name },
{ NID_LISTEXPR,       NodeInfoTable[NID_ENUMERATEDLISTEXPR].name },
{ NID_LISTEXPR,       NodeInfoTable[NID_COMPREHENDEDLISTEXPR].name },
{ NID_MAPEXPR,        NodeInfoTable[NID_ENUMERATEDMAPEXPR].name },
{ NID_MAPEXPR,        NodeInfoTable[NID_COMPREHENDEDMAPEXPR].name },
{ NID_LAMBDAPARAMETER,NodeInfoTable[NID_LAMBDATYPING].name },
{ NID_LAMBDAPARAMETER,NodeInfoTable[NID_SINGLETYPING].name },
{ NID_QUANTIFIER,     "Forall" },                                             540
{ NID_QUANTIFIER,     "Exists" },
{ NID_QUANTIFIER,     "Unique Exists" },
{ NID_INFIXEXPR,      NodeInfoTable[NID_STMTINFIXEXPR].name },
{ NID_INFIXEXPR,      NodeInfoTable[NID_AXIOMINFIXEXPR].name },
{ NID_INFIXEXPR,      NodeInfoTable[NID_VALUEINFIXEXPR].name },
{ NID_PREFIXEXPR,     NodeInfoTable[NID_AXIOMPREFIXEXPR].name },
{ NID_PREFIXEXPR,     NodeInfoTable[NID_UNIVERSALPREFIXEXPR].name },
{ NID_PREFIXEXPR,     NodeInfoTable[NID_VALUEPREFIXEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_LOCALEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_LETEXPR].name },                      550
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_IFEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_CASEEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_WHILEEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_UNTILEXPR].name },
{ NID_STRUCTUREDEXPR, NodeInfoTable[NID_FOREXPR].name },
{ NID_LETDEF,         NodeInfoTable[NID_TYPING].name },
{ NID_LETDEF,         NodeInfoTable[NID_EXPLICITLET].name },
{ NID_LETDEF,         NodeInfoTable[NID_IMPLICITLET].name },
{ NID_LETBINDING,     NodeInfoTable[NID_BINDING].name },
{ NID_LETBINDING,     NodeInfoTable[NID_RECORDPATTERN].name },                560
{ NID_LETBINDING,     NodeInfoTable[NID_LISTPATTERN].name },
{ NID_BINDING,        NodeInfoTable[NID_IDOROP].name },
{ NID_BINDING,        NodeInfoTable[NID_PRODUCTBINDING].name },
{ NID_TYPING,         NodeInfoTable[NID_SINGLETYPING].name },
{ NID_TYPING,         NodeInfoTable[NID_MULTIPLETYPING].name },
{ NID_PATTERN,        NodeInfoTable[NID_VALUELITERAL].name },
{ NID_PATTERN,        NodeInfoTable[NID_NAME].name },
{ NID_PATTERN,        NodeInfoTable[NID_WILDCARDPATTERN].name },
```

```
  { NID_PATTERN,        NodeInfoTable[NID_INNERPATTERN].name },
  { NID_PATTERN,        NodeInfoTable[NID_RECORDPATTERN].name },                    570
  { NID_PATTERN,        NodeInfoTable[NID_LISTPATTERN].name },
  { NID_LISTPATTERN,    NodeInfoTable[NID_RIGHTLISTPATTERN].name },
  { NID_LISTPATTERN,    NodeInfoTable[NID_ENUMERATEDLISTPATTERN].name },
  { NID_LISTPATTERN,    NodeInfoTable[NID_CONCATENATEDLISTPATTERN].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_VALUELITERAL].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_IDOROP].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_WILDCARDPATTERN].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_PRODUCTPATTERN].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_RECORDPATTERN].name },
  { NID_INNERPATTERN,   NodeInfoTable[NID_LISTPATTERN].name },                      580
  { NID_INNERPATTERN,   NodeInfoTable[NID_EQUALITYPATTERN].name },
  { NID_NAME,           NodeInfoTable[NID_QUALIFIEDID].name },
  { NID_NAME,           NodeInfoTable[NID_QUALIFIEDOP].name },
  { NID_IDOROP,         NodeInfoTable[NID_ID].name },
  { NID_IDOROP,         NodeInfoTable[NID_OP].name },
  { NID_OP,             NodeInfoTable[NID_INFIXOP].name },
  { NID_OP,             NodeInfoTable[NID_PREFIXOP].name },
  { NID_INFIXOP,        "=" },
  { NID_INFIXOP,        "~=" },
  { NID_INFIXOP,        ">" },                                                      590
  { NID_INFIXOP,        "<" },
  { NID_INFIXOP,        ">=" },
  { NID_INFIXOP,        "<=" },
  { NID_INFIXOP,        ">>" },
  { NID_INFIXOP,        "<<" },
  { NID_INFIXOP,        ">>=" },
  { NID_INFIXOP,        "<<=" },
  { NID_INFIXOP,        "isin" },
  { NID_INFIXOP,        "~isin" },
  { NID_INFIXOP,        "+" },                                                      600
  { NID_INFIXOP,        "-" },
  { NID_INFIXOP,        "\\" },
  { NID_INFIXOP,        "^" },
  { NID_INFIXOP,        "union" },
  { NID_INFIXOP,        "!!" },
  { NID_INFIXOP,        "*" },
  { NID_INFIXOP,        "/" },
  { NID_INFIXOP,        "#" },
  { NID_INFIXOP,        "inter" },
  { NID_INFIXOP,        "**" },                                                     610
  { NID_PREFIXOP,       "abs" },
  { NID_PREFIXOP,       "int" },
  { NID_PREFIXOP,       "real" },
  { NID_PREFIXOP,       "card" },
  { NID_PREFIXOP,       "len" },
  { NID_PREFIXOP,       "inds" },
  { NID_PREFIXOP,       "elems" },
  { NID_PREFIXOP,       "hd" },
  { NID_PREFIXOP,       "tl" },
  { NID_PREFIXOP,       "dom" },                                                    620
  { NID_PREFIXOP,       "rng" },
  { NID_CONNECTIVE,     "Infix_Connective" },
  { NID_CONNECTIVE,     "~" },
  { NID_INFIXCONNECTIVE,"=>" },
  { NID_INFIXCONNECTIVE,"Or" },
  { NID_INFIXCONNECTIVE,"And" },
  { NID_INFIXCOMBINATOR,"|=|" },
  { NID_INFIXCOMBINATOR,"|^|" },
  { NID_INFIXCOMBINATOR,"||" },
  { NID_INFIXCOMBINATOR,"++" },                                                     630
  { NID_INFIXCOMBINATOR,";" }};

/*********************************************************
* This function returns a pointer to a tree, which
* is supposed to be inserted into the root tree
```

```
* at a place where a placeholder is selected.
* It is a part of doing transformations.
* Input:
*    int - a number which identifies the transformation
*          confer with TransformationTable.                                    640
* Output:
*    TreeNode* - a pointer to a tree
*******************************************************/
PSYNTAXTREE MakeIt(const unsigned int i) {
  switch(i) {
    case 1: return new SchemeDecl;
    case 2: return new ObjectDecl;
    case 3: return new SchemeDecl;
    case 4: return new ObjectDecl;
    case 5: return new TypeDecl;                                              650
    case 6: return new ValueDecl;
    case 7: return new VariableDecl;
    case 8: return new ChannelDecl;
    case 9: return new AxiomDecl;
    case 10: return new SortDef;
    case 11: return new VariantDef;
    case 12: return new UnionDef;
    case 13: return new ShortRecordDef;
    case 14: return new AbbreviationDef;
    case 15: return new Constructor;                                          660
    case 16: return new RecordVariant;
    case 17: return new IdOrOp;
    case 18: return new Wildcard;
    case 19: return new Name;
    case 20: return new Wildcard;
    case 21: return new CommentedTyping;
    case 22: return new ExplicitValueDef;
    case 23: return new ImplicitValueDef;
    case 24: return new ExplicitFunctionDef;
    case 25: return new ImplicitFunctionDef;                                  670
    case 26: return new IdApplication;
    case 27: return new PrefixApplication;
    case 28: return new InfixApplication;
    case 29: return new SingleVariableDef;
    case 30: return new MultipleVariableDef;
    case 31: return new SingleChannelDef;
    case 32: return new MultipleChannelDef;
    case 33: return new BasicClassExpr;
    case 34: return new ExtendingClassExpr;
    case 35: return new HidingClassExpr;                                      680
    case 36: return new RenamingClassExpr;
    case 37: return new SchemeInstantiation;
    case 38: return new IdOrOp;
    case 39: return new DisambiguatedItem;
    case 40: return new Name;
    case 41: return new ElementObjectExpr;
    case 42: return new ArrayObjectExpr;
    case 43: return new FittingObjectExpr;
    case 44: return new TypeLiteral;
    case 45: return new Name;                                                 690
    case 46: return new ProductTypeExpr;
    case 47: return new SetTypeExpr;
    case 48: return new ListTypeExpr;
    case 49: return new MapTypeExpr;
    case 50: return new FunctionTypeExpr;
    case 51: return new SubTypeExpr;
    case 52: return new BracketedTypeExpr;
    case 53: return new TUnit;
    case 54: return new TBool;
    case 55: return new TInt;                                                 700
    case 56: return new TNat;
    case 57: return new TReal;
```

```
case 58: return new TText;
case 59: return new TChar;
case 60: return new FiniteSetTypeExpr;
case 61: return new InfiniteSetTypeExpr;
case 62: return new FiniteListTypeExpr;
case 63: return new InfiniteListTypeExpr;
case 64: return new ParRightArrow; // -~>
case 65: return new RightArrow; // ->                                    710
case 66: return new Read;
case 67: return new Write;
case 68: return new In;
case 69: return new Out;
case 70: return new Name;
case 71: return new EnumeratedAccess;
case 72: return new CompletedAccess;
case 73: return new ComprehendedAccess;
case 74: return new ValueLiteral;
case 75: return new Name;                                                720
case 76: return new PreName;
case 77: return new BasicExpr;
case 78: return new ProductExpr;
case 79: return new SetExpr;
case 80: return new ListExpr;
case 81: return new MapExpr;
case 82: return new FunctionExpr;
case 83: return new ApplicationExpr;
case 84: return new QuantifiedExpr;
case 85: return new EquivalenceExpr;                                     730
case 86: return new PostExpr;
case 87: return new DisambiguatedExpr;
case 88: return new BracketedExpr;
case 89: return new InfixExpr;
case 90: return new PrefixExpr;
case 91: return new ComprehendedExpr;
case 92: return new InitialiseExpr;
case 93: return new AssignmentExpr;
case 94: return new InputExpr;
case 95: return new OutputExpr;                                          740
case 96: return new StructuredExpr;
case 97: return new UnitLiteral;
case 98: return new BoolLiteral;
case 99: return new IntLiteral;
case 100: return new RealLiteral;
case 101: return new TextLiteral;
case 102: return new CharLiteral;
case 103: return new True;
case 104: return new False;
case 105: return new Chaos;                                              750
case 106: return new Skip;
case 107: return new Stop;
case 108: return new Swap;
case 109: return new RangedSetExpr;
case 110: return new EnumeratedSetExpr;
case 111: return new ComprehendedSetExpr;
case 112: return new RangedListExpr;
case 113: return new EnumeratedListExpr;
case 114: return new ComprehendedListExpr;
case 115: return new EnumeratedMapExpr;                                  760
case 116: return new ComprehendedMapExpr;
case 117: return new LambdaTyping;
case 118: return new SingleTyping;
case 119: return new Forall;
case 120: return new Exists;
case 121: return new ExistsUnique;
case 122: return new StmtInfixExpr;
case 123: return new AxiomInfixExpr;
case 124: return new ValueInfixExpr;
```

```
case 125: return new AxiomPrefixExpr;
case 126: return new UniversalPrefixExpr;
case 127: return new ValuePrefixExpr;
case 128: return new LocalExpr;
case 129: return new LetExpr;
case 130: return new IfExpr;
case 131: return new CaseExpr;
case 132: return new WhileExpr;
case 133: return new UntilExpr;
case 134: return new ForExpr;
case 135: return new Typing;
case 136: return new ExplicitLet;
case 137: return new ImplicitLet;
case 138: return new Binding;
case 139: return new RecordPattern;
case 140: return new ListPattern;
case 141: return new IdOrOp;
case 142: return new ProductBinding;
case 143: return new SingleTyping;
case 144: return new MultipleTyping;
case 145: return new ValueLiteral;
case 146: return new Name;
case 147: return new WildcardPattern;
case 148: return new ProductPattern;
case 149: return new RecordPattern;
case 150: return new ListPattern;
case 151: return new RightListPattern;
case 152: return new EnumeratedListPattern;
case 153: return new ConcatenatedListPattern;
case 154: return new ValueLiteral;
case 155: return new IdOrOp;
case 156: return new WildcardPattern;
case 157: return new ProductPattern;
case 158: return new RecordPattern;
case 159: return new ListPattern;
case 160: return new EqualityPattern;
case 161: return new QualifiedId;
case 162: return new QualifiedOp;
case 163: return new Id;
case 164: return new Op;
case 165: return new InfixOp;
case 166: return new PrefixOp;
case 167: return new Equal;
case 168: return new NotEqual;
case 169: return new Gt;
case 170: return new Lt;
case 171: return new GE;
case 172: return new LE;
case 173: return new SuperSet;
case 174: return new Subset;
case 175: return new ProperSuperset;
case 176: return new ProperSubset;
case 177: return new IsIn;
case 178: return new NotIsIn;
case 179: return new Pluus;
case 180: return new Minuus;
case 181: return new Backslash;
case 182: return new Concat;
case 183: return new Union;
case 184: return new Override;
case 185: return new Mult;
case 186: return new Divide;
case 187: return new Composition;
case 188: return new Inter;
case 189: return new Exp;
case 190: return new Abs;
case 191: return new Int;
```

```
    case 192: return new Real;
    case 193: return new Card;
    case 194: return new Len;
    case 195: return new Inds;
    case 196: return new Elems;
    case 197: return new Hd;
    case 198: return new Tl;
    case 199: return new Dom;
    case 200: return new Rng;
    case 201: return new InfixConnective;
    case 202: return new Not;
    case 203: return new Implication;
    case 204: return new Or;
    case 205: return new And;
    case 206: return new ExtChoice;
    case 207: return new IntChoice;
    case 208: return new ConComposition;
    case 209: return new IntComposition;
    case 210: return new SeqComposition;
  }
  return NULL;
}

#endif /* NODEINFO_CPP */
```

G.6 POSWND.CPP

```
/*************************************************************************
*
* POSWND.CPP
*
* Position Window Procedure Implementation
*
* Created 20 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef POSWND_CPP
#define POSWND_CPP

#include "RSLEDWIN.H"

// Prototypes for process functions
void PosWndProcessPAINT(HWND);

/*************************************************************************
*
* Position Window Procedure
*
*************************************************************************/

long FAR PASCAL PosWndProc(HWND hwnd, unsigned message, WPARAM wParam, LPARAM lParam)
{
   switch(message)
   {
     case WM_PAINT:
       PosWndProcessPAINT(hwnd);
       return 0;
     case WM_DESTROY:
       PostQuitMessage(0);
       return 0;
   }
```

```
    return DefWindowProc(hwnd,message,wParam,lParam);
}

/*********************************************************
* Process WM_PAINT message for Position window                    40
* Input:
* hwnd - Handle of window
* Output:
* node
* Side Effects:
*
*********************************************************/
void PosWndProcessPAINT(HWND hwnd) {
  HDC hdc;
  PAINTSTRUCT ps;                                                 50

  hdc = BeginPaint(hwnd,&ps);
  static char s[] = "Postioned at ";

  SelectObject(hdc,hfontNormal);

  TextOut(hdc,1,-1,s,lstrlen(s));
  SIZE dwSize;
  GetTextExtentPoint(hdc,s,lstrlen(s),&dwSize);
                                                                  60
  if(SelectedNode&&NodeInfoTable[SelectedNode->get_nodeid()].name!=NULL)
  TextOut(hdc,1+dwSize.cx,-1,NodeInfoTable[SelectedNode->get_nodeid()].name,
                lstrlen(NodeInfoTable[SelectedNode->get_nodeid()].name));

  EndPaint(hwnd,&ps);
}

#endif /* POSWND_CPP */
                                                                  70
```

G.7 RSL.CPP

```
/*************************************************************************
*
* RSL.CPP
*
* Implementation of RSL language features
*
* Created 22 December 1993, Michael Suodenjoki
*
*************************************************************************/
                                                                  10
#ifndef RSL_CPP
#define RSL_CPP

#include "RSL.H"


#define MakeSubtree(x,y) void x ## ::make_subtree() { y };


/************** Implementation of make_childs() *****************/      20

MakeSubtree(QualifiedOp,
  insert_tree_last(new OptQualification);
  insert_tree_last(new Op);
)
```

```
MakeSubtree(Qualification,
  insert_tree_last(new ObjectExpr);
)

MakeSubtree(QualifiedId,
    insert_tree_last(new OptQualification);
    insert_tree_last(new Id);
)

MakeSubtree(EqualityPattern,
  insert_tree_last(new Name);
)

MakeSubtree(ConcatenatedListPattern,
  insert_tree_last(new EnumeratedListPattern);
  insert_tree_last(new InnerPattern);
)

MakeSubtree(EnumeratedListPattern,
  insert_tree_last(new OptInnerPatternList);
)

MakeSubtree(RightListPattern,
  insert_tree_last(new Id);
  insert_tree_last(new EnumeratedListPattern);
)

MakeSubtree(RecordPattern,
  insert_tree_last(new Name);
  insert_tree_last(new InnerPatternList);
)

MakeSubtree(ProductPattern,
  insert_tree_last(new InnerPatternList2);
)

MakeSubtree(CommentedTyping,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new Typing);
)

MakeSubtree(MultipleTyping,
  insert_tree_last(new BindingList2);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(SingleTyping,
  insert_tree_last(new Binding);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(ProductBinding,
  insert_tree_last(new BindingList2);
)

MakeSubtree(ForExpr,
  insert_tree_last(new ListLimitation);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(UntilExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(WhileExpr,
  insert_tree_last(new ValueExpr);
```

```
  insert_tree_last(new ValueExpr);
)

MakeSubtree(CaseBranch,
  insert_tree_last(new Pattern);
  insert_tree_last(new ValueExpr);
)                                                                    100

MakeSubtree(CaseExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new CaseBranchList);
)

MakeSubtree(ElseBranch,
  insert_tree_last(new ValueExpr);
)
                                                                     110
MakeSubtree(ElsifBranch,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(IfExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
  insert_tree_last(new OptElsifBranchString);
  insert_tree_last(new OptElseBranch);                               120
)

MakeSubtree(ImplicitLet,
  insert_tree_last(new SingleTyping);
  insert_tree_last(new Restriction);
)

MakeSubtree(ExplicitLet,
  insert_tree_last(new LetBinding);
  insert_tree_last(new ValueExpr);                                   130
)

MakeSubtree(LetExpr,
  insert_tree_last(new LetDefList);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(LocalExpr,
  insert_tree_last(new OptDeclString);
  insert_tree_last(new ValueExpr);                                   140
)

MakeSubtree(OutputExpr,
  insert_tree_last(new Name);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(InputExpr,
  insert_tree_last(new Name);
)                                                                    150

MakeSubtree(AssignmentExpr,
  insert_tree_last(new Name);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(InitialiseExpr,
  insert_tree_last(new OptQualification);
)
                                                                     160
```

```
MakeSubtree(ComprehendedExpr,
  insert_tree_last(new InfixCombinator);
  insert_tree_last(new ValueExpr);
  insert_tree_last(new SetLimitation);
)

MakeSubtree(ValuePrefixExpr,
  insert_tree_last(new PrefixOp);
  insert_tree_last(new ValueExpr);
)                                                                    170

MakeSubtree(UniversalPrefixExpr,
  insert_tree_last(new ValueExpr);
)

MakeSubtree(AxiomPrefixExpr,
  insert_tree_last(new PrefixConnective);
  insert_tree_last(new ValueExpr);
)
                                                                     180
MakeSubtree(ValueInfixExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new InfixOp);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(AxiomInfixExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new InfixConnective);
  insert_tree_last(new ValueExpr);                                   190
)

MakeSubtree(StmtInfixExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new InfixCombinator);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(BracketedExpr,
  insert_tree_last(new ValueExpr);                                   200
)

MakeSubtree(DisambiguatedExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(ResultNaming,
  insert_tree_last(new Binding);
)                                                                    210

MakeSubtree(PostCondition,
  insert_tree_last(new OptResultNaming);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(PostExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new PostCondition);
  insert_tree_last(new OptPreCondition);                             220
)

MakeSubtree(PreCondition,
  insert_tree_last(new ValueExpr);
)

MakeSubtree(EquivalenceExpr,
```

```
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
  insert_tree_last(new OptPreCondition);                                    230
)

MakeSubtree(QuantifiedExpr,
  insert_tree_last(new Quantifier);
  insert_tree_last(new TypingList);
  insert_tree_last(new Restriction);
)

MakeSubtree(ActualFunctionParameter,
  insert_tree_last(new OptValueExprList);                                   240
)

MakeSubtree(ApplicationExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ActualFunctionParameterString);
)

MakeSubtree(LambdaTyping,
  insert_tree_last(new OptTypingList);
)                                                                           250

MakeSubtree(FunctionExpr,
  insert_tree_last(new LambdaParameter);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(ComprehendedMapExpr,
  insert_tree_last(new ValueExprPair);
  insert_tree_last(new SetLimitation);
)                                                                           260

MakeSubtree(ValueExprPair,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(EnumeratedMapExpr,
  insert_tree_last(new OptValueExprPairList);
)
                                                                            270
MakeSubtree(ListLimitation,
  insert_tree_last(new Binding);
  insert_tree_last(new ValueExpr);
  insert_tree_last(new OptRestriction);
)

MakeSubtree(ComprehendedListExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ListLimitation);
)                                                                           280

MakeSubtree(EnumeratedListExpr,
  insert_tree_last(new OptValueExprList);
)

MakeSubtree(RangedListExpr,
  insert_tree_last(new ValueExpr);
  insert_tree_last(new ValueExpr);
)
                                                                            290
MakeSubtree(Restriction,
  insert_tree_last(new ValueExpr);
)
```

```
MakeSubtree(SetLimitation,
  insert_tree_last(new TypingList);
  insert_tree_last(new OptRestriction);
)

MakeSubtree(ComprehendedSetExpr,                                    300
  insert_tree_last(new ValueExpr);
  insert_tree_last(new SetLimitation);
)

MakeSubtree(EnumeratedSetExpr,
  insert_tree_last(new ValueExprList);
)

MakeSubtree(RangedSetExpr,
  insert_tree_last(new ValueExpr);                                  310
  insert_tree_last(new ValueExpr);
)

MakeSubtree(ProductExpr,
  insert_tree_last(new ValueExprList2);
)

MakeSubtree(PreName,
  insert_tree_last(new Name);
)                                                                   320

MakeSubtree(ComprehendedAccess,
  insert_tree_last(new Access);
  insert_tree_last(new SetLimitation);
)

MakeSubtree(CompletedAccess,
  insert_tree_last(new OptQualification);
)
                                                                    330
MakeSubtree(EnumeratedAccess,
  insert_tree_last(new OptAccessList);
)

MakeSubtree(AccessDesc,
  insert_tree_last(new AccessMode);
  insert_tree_last(new AccessList);
)

MakeSubtree(BracketedTypeExpr,                                      340
  insert_tree_last(new TypeExpr);
)

MakeSubtree(SubTypeExpr,
  insert_tree_last(new SingleTyping);
  insert_tree_last(new Restriction);
)

MakeSubtree(ResultDesc,
  insert_tree_last(new OptAccessDescString);                        350
  insert_tree_last(new TypeExpr);
)

MakeSubtree(FunctionTypeExpr,
  insert_tree_last(new TypeExpr);
  insert_tree_last(new FunctionArrow);
  insert_tree_last(new ResultDesc);
)

MakeSubtree(MapTypeExpr,                                            360
  insert_tree_last(new TypeExpr);
```

```
  insert_tree_last(new TypeExpr);
)

MakeSubtree(InfiniteListTypeExpr,
  insert_tree_last(new TypeExpr);
)

MakeSubtree(FiniteListTypeExpr,
  insert_tree_last(new TypeExpr);                                    370
)

MakeSubtree(InfiniteSetTypeExpr,
  insert_tree_last(new TypeExpr);
)

MakeSubtree(FiniteSetTypeExpr,
  insert_tree_last(new TypeExpr);
)                                                                    380

MakeSubtree(ProductTypeExpr,
  insert_tree_last(new TypeExprProduct2);
)

MakeSubtree(FittingObjectExpr,
  insert_tree_last(new ObjectExpr);
  insert_tree_last(new RenamePairList);
)

MakeSubtree(ArrayObjectExpr,                                         390
  insert_tree_last(new TypingList);
  insert_tree_last(new ObjectExpr);
)

MakeSubtree(ActualArrayParameter,
  insert_tree_last(new ValueExprList);
)

MakeSubtree(ElementObjectExpr,
  insert_tree_last(new ObjectExpr);                                  400
  insert_tree_last(new ActualArrayParameter);
)

MakeSubtree(DisambiguatedItem,
  insert_tree_last(new IdOrOp);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(RenamePair,
  insert_tree_last(new DefinedItem);                                 410
  insert_tree_last(new DefinedItem);
)

MakeSubtree(ActualSchemeParameter,
  insert_tree_last(new ObjectExprList);
)

MakeSubtree(SchemeInstantiation,
  insert_tree_last(new Name);
  insert_tree_last(new OptActualSchemeParameter);                    420
)

MakeSubtree(RenamingClassExpr,
  insert_tree_last(new RenamePairList);
  insert_tree_last(new ClassExpr);
)

MakeSubtree(HidingClassExpr,
```

```
  insert_tree_last(new DefinedItemList);
  insert_tree_last(new ClassExpr);
)

MakeSubtree(ExtendingClassExpr,
  insert_tree_last(new ClassExpr);
  insert_tree_last(new ClassExpr);
)

MakeSubtree(BasicClassExpr,
  insert_tree_last(new OptDeclString);
)

MakeSubtree(AxiomNaming,
  insert_tree_last(new Id);
)

MakeSubtree(AxiomDef,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new OptAxiomNaming);
  insert_tree_last(new ValueExpr);
)

MakeSubtree(AxiomQuantification,
  insert_tree_last(new TypingList);
)

MakeSubtree(AxiomDecl,
  insert_tree_last(new OptAxiomQuantification);
  insert_tree_last(new AxiomDefList);
)

MakeSubtree(MultipleChannelDef,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new IdList2);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(SingleChannelDef,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new Id);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(ChannelDecl,
  insert_tree_last(new ChannelDefList);
)

MakeSubtree(MultipleVariableDef,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new IdList2);
  insert_tree_last(new TypeExpr);
)

MakeSubtree(Initialisation,
  insert_tree_last(new ValueExpr);
)

MakeSubtree(SingleVariableDef,
  insert_tree_last(new OptCommentString);
  insert_tree_last(new Id);
  insert_tree_last(new TypeExpr);
  insert_tree_last(new OptInitialisation);
)

MakeSubtree(VariableDecl,
  insert_tree_last(new VariableDefList);
```

```
)

MakeSubtree(ImplicitFunctionDef,
   insert_tree_last(new OptCommentString);
   insert_tree_last(new SingleTyping);                                    500
   insert_tree_last(new FormalFunctionApplication);
   insert_tree_last(new PostCondition);
   insert_tree_last(new OptPreCondition);
)

MakeSubtree(InfixApplication,
 insert_tree_last(new Id);
 insert_tree_last(new InfixOp);
 insert_tree_last(new Id);
)                                                                         510

MakeSubtree(PrefixApplication,
   insert_tree_last(new PrefixOp);
   insert_tree_last(new Id);
)

MakeSubtree(FormalFunctionParameter,
   insert_tree_last(new OptBindingList);
)
                                                                          520
MakeSubtree(IdApplication,
   insert_tree_last(new Id);
   insert_tree_last(new FormalFunctionParameterString);
)

MakeSubtree(ExplicitFunctionDef,
   insert_tree_last(new OptCommentString);
   insert_tree_last(new SingleTyping);
   insert_tree_last(new FormalFunctionApplication);
   insert_tree_last(new ValueExpr);                                       530
   insert_tree_last(new OptPreCondition);
)

MakeSubtree(ImplicitValueDef,
   insert_tree_last(new OptCommentString);
   insert_tree_last(new SingleTyping);
   insert_tree_last(new Restriction);
)

MakeSubtree(ExplicitValueDef,                                             540
   insert_tree_last(new OptCommentString);
   insert_tree_last(new SingleTyping);
   insert_tree_last(new ValueExpr);
)

MakeSubtree(ValueDecl,
   insert_tree_last(new ValueDefList);
)

MakeSubtree(AbbreviationDef,                                              550
   insert_tree_last(new OptCommentString);
   insert_tree_last(new Id);
   insert_tree_last(new TypeExpr);
)

MakeSubtree(ShortRecordDef,
   insert_tree_last(new OptCommentString);
   insert_tree_last(new Id);
   insert_tree_last(new ComponentKindString);
)                                                                         560

MakeSubtree(UnionDef,
```

```
    insert_tree_last(new OptCommentString);
    insert_tree_last(new Id);
    insert_tree_last(new NameOrWildcardChoice2);
)

MakeSubtree(Reconstructor,
    insert_tree_last(new IdOrOp);
)                                                                    570

MakeSubtree(Destructor,
    insert_tree_last(new IdOrOp);
)

MakeSubtree(ComponentKind,
    insert_tree_last(new OptDestructor);
    insert_tree_last(new TypeExpr);
    insert_tree_last(new OptReconstructor);
)                                                                    580

MakeSubtree(RecordVariant,
    insert_tree_last(new Constructor);
    insert_tree_last(new ComponentKindList);
)

MakeSubtree(VariantDef,
    insert_tree_last(new OptCommentString);
    insert_tree_last(new Id);
    insert_tree_last(new VariantChoice);                             590
)

MakeSubtree(SortDef,
    insert_tree_last(new OptCommentString);
    insert_tree_last(new Id);
)

MakeSubtree(TypeDecl,
    insert_tree_last(new TypeDefList);
)                                                                    600

MakeSubtree(FormalArrayParameter,
    insert_tree_last(new TypingList);
)

MakeSubtree(ObjectDef,
    insert_tree_last(new OptCommentString);
    insert_tree_last(new Id);
    insert_tree_last(new OptFormalArrayParameter);
    insert_tree_last(new ClassExpr);                                 610
)

MakeSubtree(ObjectDecl,
    insert_tree_last(new ObjectDefList);
)

MakeSubtree(FormalSchemeParameter,
    insert_tree_last(new ObjectDefList);
)
                                                                     620
MakeSubtree(SchemeDef,
    insert_tree_last(new OptCommentString);
    insert_tree_last(new Id);
    insert_tree_last(new OptFormalSchemeParameter);
    insert_tree_last(new ClassExpr);
)

MakeSubtree(SchemeDecl,
    insert_tree_last(new SchemeDefList);
```

```
)

MakeSubtree(Specification,
  insert_tree_last(new ModuleDeclString);
)


#endif /* RSL_CPP */
```

G.8 RSL1.CPP

```
/*****************************************************************************
*
* RSL1.CPP
*
* Implementation of constructor function for Class definitions
* Implementation of make_child() function for Class definitions
*
* Created 19 January, 1994, Michael Suodenjoki
*
*****************************************************************************/

#ifndef RSL1_CPP
#define RSL1_CPP

#include "RSL.H"
#include "NODEIDS.H"
#include "MYSTRING.H"
#include "LANGUAGE.H"

#undef NullCl
#define NullCl(x,y) \
x ## Null:: x ## Null() { \
  set_nodeid(NID_ ## y ## NULL); set_resting_place(FALSE); \
};

#undef VarCl
#define VarCl(x,y) \
NullCl(x,y) \
\
x :: x () { \
  set_nodeid(NID_ ## y); make_subtree(); \
};

#undef ListCl
#define ListCl(x,y) \
x ## List:: x ## List() { \
  set_nodeid(NID_ ## y ## LIST); set_listnode(TRUE); \
  set_minfixed(1); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## List::make_child() { return new x; };

#undef ListClRest
#define ListClRest(x,y) \
x ## List:: x ## List() { \
  set_nodeid(NID_ ## y ## LIST); set_listnode(TRUE); \
  set_minfixed(1); make_subtree(); \
}; \
PSYNTAXTREE x ## List::make_child() { return new x; };


#undef ChoiceCl
#define ChoiceCl(x,y) \
```

```
x ## Choice:: x ## Choice() { \
  set_nodeid(NID_ ## y ## CHOICE); set_listnode(TRUE); \
  set_minfixed(1); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## Choice::make_child() { return new x; };

#undef List2Cl
#define List2Cl(x,y) \
x ## List2:: x ## List2() { \
  set_nodeid(NID_ ## y ## LIST2); set_listnode(TRUE); \
  set_minfixed(2); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## List2::make_child() { return new x; }

#undef StringCl
#define StringCl(x,y) \
x ## String:: x ## String() { \
  set_nodeid(NID_ ## y ## STRING); set_listnode(TRUE); \
  set_minfixed(1); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## String::make_child() { return new x; }

#undef Optional
#define Optional(x,y) \
Opt ## x ## :: Opt ## x () { \
  set_nodeid(NID_OPT ## y); set_optional(TRUE); \
  set_resting_place(FALSE); \
}; \
PSYNTAXTREE Opt ## x ::make_child() { return new x; }

#undef Terminal
#define Terminal(x,y) \
x :: x () { \
  set_nodeid(NID_ ## y); set_resting_place(FALSE); \
};

#undef Class
#define Class(x,y) \
x :: x () { \
  set_nodeid(NID_ ## y); set_resting_place(FALSE); make_subtree(); \
};

#undef ClassRest
#define ClassRest(x,y) \
x :: x () { \
  set_nodeid(NID_ ## y); make_subtree(); \
};

#undef OptString
#define OptString(x,y) \
StringCl(x,y) \
\
Opt ## x ## String:: Opt ## x ## String() { \
  set_nodeid(NID_OPT ## y ## STRING); \
  set_optional(TRUE); set_resting_place(FALSE); \
}; \
PSYNTAXTREE Opt ## x ## String::make_child() { return new x ## String; }

#undef OptList
#define OptList(x,y) \
ListCl(x,y) \
\
Opt ## x ## List:: Opt ## x ## List() { \
  set_nodeid(NID_OPT ## y ## LIST); \
  set_optional(TRUE); set_resting_place(FALSE); \
}; \
```

```
PSYNTAXTREE Opt ## x ## List::make_child() { return new x ## List; }

#undef Choice2Cl
#define Choice2Cl(x,y) \
x ## Choice2:: x ## Choice2() { \
  set_nodeid(NID_ ## y ## CHOICE2); set_listnode(TRUE); \
  set_minfixed(2); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## Choice2::make_child() { return new x; }

#undef Product2Cl
#define Product2Cl(x,y) \
x ## Product2:: x ## Product2() { \
  set_nodeid(NID_ ## y ## PRODUCT2); set_listnode(TRUE); \
  set_minfixed(2); set_resting_place(FALSE); make_subtree(); \
}; \
PSYNTAXTREE x ## Product2::make_child() { return new x; }

#undef LexTerminal
#define LexTerminal(x,y) \
x :: x (char *string) { \
  set_nodeid(NID_ ## y); set_resting_place(FALSE); \
  int i=my_strlen(string); \
  i=i > IDLENGTH-1 ? IDLENGTH-1 : i; \
  my_strncpy(str,string,i); \
  str[i]='\0'; \
} \
char * x ::unparse_special(void) const { return str; } \
BOOL x ::compare(const PSYNNODE n1,const PSYNNODE n2) const { \
  return FALSE; \
}

/*****************************************************************
*
* LEXICAL CLASSES
*
*****************************************************************/

LexTerminal(Identifier,IDENTIFIER)
LexTerminal(IntLiteral,INTLITERAL)
LexTerminal(RealLiteral,REALLITERAL)
LexTerminal(TextLiteral,TEXTLITERAL)
LexTerminal(CharLiteral,CHARLITERAL)
LexTerminal(CommentText,COMMENTTEXT)

/*** Others ***/

// Same macro calls as in RSL.H

#endif /* RSL1_CPP */
```

G.9 RSL2.CPP

```
/*****************************************************************************
*
* RSL2.CPP
*
* Implementation of copy_tree function for Class definitions
*
* Created 19 January, 1994, Michael Suodenjoki
*
*****************************************************************************/
```

```
#ifndef RSL2_CPP
#define RSL2_CPP

#include "RSL.H"

#undef NullCl
#define NullCl(x,y) \
PSYNTAXTREE x ## Null::clone(void) const { \
  return new x ## Null (* this); \
};

#undef VarCl
#define VarCl(x,y) \
NullCl(x,y) \
\
PSYNTAXTREE x ## ::clone(void) const { \
  return new x (* this); \
};

#undef ListCl
#define ListCl(x,y) \
PSYNTAXTREE x ## List::clone(void) const { \
  return new x ## List (* this); \
};

#undef ChoiceCl
#define ChoiceCl(x,y) \
PSYNTAXTREE x ## Choice::clone(void) const { \
  return new x ## Choice (* this); \
};

#undef List2Cl
#define List2Cl(x,y) \
PSYNTAXTREE x ## List2::clone(void) const { \
  return new x ## List2 (* this); \
};

#undef StringCl
#define StringCl(x,y) \
PSYNTAXTREE x ## String::clone(void) const { \
  return new x ## String (* this); \
};

#undef Optional
#define Optional(x,y) \
PSYNTAXTREE Opt ## x ## ::clone(void) const { \
  return new Opt ## x (* this); \
};

#undef Terminal
#define Terminal(x,y) \
PSYNTAXTREE x ## ::clone(void) const { \
  return new x (* this); \
};

#undef Class
#define Class(x,y) \
PSYNTAXTREE x ## ::clone(void) const { \
  return new x (* this); \
};

#undef ClassRest
#define ClassRest(x,y) \
PSYNTAXTREE x ## ::clone(void) const { \
  return new x (* this); \
};
```

```
#undef OptString
#define OptString(x,y) \
StringCl(x,y) \
\
PSYNTAXTREE Opt ## x ## String::clone(void) const { \
  return new Opt ## x ## String (* this); \
};

#undef OptList
#define OptList(x,y) \
ListCl(x,y) \
\
PSYNTAXTREE Opt ## x ## List::clone(void) const { \
  return new Opt ## x ## List (* this); \
};

#undef Choice2Cl
#define Choice2Cl(x,y) \
PSYNTAXTREE x ## Choice2::clone(void) const { \
  return new x ## Choice2 (* this); \
};

#undef Product2Cl
#define Product2Cl(x,y) \
PSYNTAXTREE x ## Product2::clone(void) const { \
  return new x ## Product2 (* this); \
};

Terminal(Identifier,IDENTIFIER)
Terminal(CommentText,COMMENTTEXT)

// Same macro calls as in RSL.H

#endif /* RSL2_CPP */
```

G.10 RSL3.CPP

```
/*****************************************************************************
*
* RSL3.CPP
*
* Implementation of make_subtree function for Class definitions
*
* Created 19 January, 1994, Michael Suodenjoki
*
*****************************************************************************/

#ifndef RSL3_CPP
#define RSL3_CPP

#include "RSL.H"

#undef NullCl
#define NullCl(x,y)

#undef VarCl
#define VarCl(x,y) \
void x ##::make_subtree(void) { insert_tree_last(new x ## Null); }

#undef ListCl
#define ListCl(x,y) \
void x ## List::make_subtree(void) { insert_tree_last(new x); }
```

```
#undef ChoiceCl
#define ChoiceCl(x,y) \
void x ## Choice::make_subtree(void) { insert_tree_last(new x); }

#undef List2Cl
#define List2Cl(x,y) \
void x ## List2::make_subtree(void) { \
  insert_tree_last(new x); insert_tree_last(new x); \
}

#undef StringCl

#define StringCl(x,y) \
void x ## String::make_subtree(void) { insert_tree_last(new x); }

#undef Optional
#define Optional(x,y)

#undef Terminal
#define Terminal(x,y)

/* Are defined in RSL.CPP */
#undef Class
#define Class(x,y)

/* Are defined in RSL.CPP */
#undef ClassRest
#define ClassRest(x,y)

#undef OptString
#define OptString(x,y) \
StringCl(x,y)

#undef OptList
#define OptList(x,y) \
ListCl(x,y)

#undef Choice2Cl
#define Choice2Cl(x,y) \
void x ## Choice2::make_subtree(void) { \
   insert_tree_last(new x); insert_tree_last(new x); \
}

#undef Product2Cl
#define Product2Cl(x,y) \
void x ## Product2::make_subtree(void) { \
  insert_tree_last(new x); insert_tree_last(new x); \
}

VarCl(Id,ID)
List2Cl(Id,ID)

Terminal(CommentNewline,COMMENTNEWLINE)
Terminal(CommentSpace,COMMENTSPACE)
VarCl(CommentToken,COMMENTTOKEN)
StringCl(CommentToken,COMMENTTOKEN)

// Same macro calls as in RSL.H

#endif /* RSL3_CPP */
```

G.11 RSLEDWIN.CPP

```
/*************************************************************************
*
* RSLEDWIN.CPP
*
* Main Program Implementation, RAISE Specification Language Editor
*
* Created 20 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#define GLOBAL

#include "RSLEDWIN.H"
#include <NEW.H>
#include "IO.H"

// The height (in pixels) of the transformation window
// (ei. window with buttons to transform)
#define TRANSAREAHEIGHT 100

// Logfont Width, Average width of font characters
#define LFWIDTH 8
// LogFont Height, Height of font characeters
#define LFHEIGHT 16

// Global (private) variables for this module
// Names of window classes
static char szPosClass[]   = "Pos_Child";
static char szTransClass[] = "Trans_Child";
static char szEditClass[]  = "Edit_Child";

// Untitled filename text
const char szUntitled[] = "(untitled)";

// Menu Checks
BOOL asciion = FALSE;

// Logfont variables
LOGFONT lfNormal;
LOGFONT lfKeyword;
LOGFONT lfPlaceholder;
LOGFONT lfSymbol;
LOGFONT lfComment;
LOGFONT lfText;

// See Microsoft Win32 Programmer's reference, Volume 2, pp. 692
static OPENFILENAME ofn;      // Common dialog box structure
static char szDirName[256]    // directory string
  = ".";
static char szFile[256];      // filename string
static char szFileTitle[256]; // file-title string
static char szFilter[256] =   // filter string
  "RSL Language file\0*.rsl\0All files\0*.*\0\0";
static char chReplace;        // string seperator for szFilter
static int i, cbString;       // integer count variables
static HANDLE hf;             // file handle

static BOOL bNeedSave = TRUE; // Does the current editor contents need to
                              // be saved, before leaving.

/*************************************************************************
*
* General Help rutines
*
```

```
*****************************************************************************/

/*********************************************************
* This is the Intro dialog window procedure
* Input:                                                                    70
* hDlg - handle of dialog window
* msg - message
* wParam - window parameter
* lParam - extra window parameter
* Output:
* BOOL - if message is processed
* Side Effects:
* moves the window to the center of physical screen
*********************************************************/
BOOL FAR PASCAL Intro(HWND hDlg,unsigned int msg,WPARAM wParam,LPARAM lParam) {   80
  int    x,y;
  RECT rect;

  switch(msg) {
    case WM_INITDIALOG:
      // Disable (gray out) the OK button
      // EnableWindow(GetDlgItem(hDlg,IDOK),FALSE);

      // First find the size of the dialog box
      GetWindowRect(hDlg,&rect);                                            90
      x=rect.right-rect.left;
      y=rect.bottom-rect.top;

      // Move the dialog to the center of the physical screen
      MoveWindow(hDlg,(GetSystemMetrics(SM_CXSCREEN)-x)/2,
                 (GetSystemMetrics(SM_CYSCREEN)-y)/2,
                 x,y,FALSE);
      return TRUE;
    case WM_COMMAND:
      switch(wParam) {                                                      100
        case IDOK:
          EndDialog(hDlg,0);
          return TRUE;
      }
      return FALSE;
    default:
      // Allow normal processing of any other messages
      return FALSE;
  }
}                                                                           110

/*********************************************************
* This function changed the windows caption e.i. the
* headline of the menu, to contain the current filename
* Input:
* hwnd - handle of window
* szFileName - filename
* Output:
* none
* Side Effects:                                                             120
* changes the window caption
*********************************************************/
void DoCaption(HWND hwnd,const char *szFileName) {
  char szCaption[40];

  wsprintf(szCaption,"%s - %s", (LPSTR) szAppName,
    (LPSTR) (szFileName[0] ? szFileName : szUntitled));

  SetWindowText(hwnd,szCaption);
}                                                                           130
```

```
/*********************************************************
* This function opens a messagebox wuth a question
* about saving current file.
* Input:
* hwnd - handle of window
* szFileName - filename of current file
* Output:
* answer from the user (IDYES, IDNO or IDCANCEL)                    140
* Side Effects:
* none
*********************************************************/
int AskAboutSave(HWND hwnd,const char *szFileName) {
  char szBuffer[255];
  int nReturn;

  wsprintf(szBuffer,"Save current changes: %s",
    (LPSTR) (szFileName[0] ? szFileName : szUntitled));
                                                                    150
  if(IDYES == (nReturn = MessageBox(hwnd,szBuffer,szAppName,
    MB_YESNOCANCEL | MB_ICONQUESTION)))

    if(!SendMessage(hwnd,WM_COMMAND,IDM_SAVE,0L))
      return IDCANCEL;

  return nReturn;
}

/*********************************************************          160
* Initialises the fonts
* Input:
* none
* Output:
* none
* Side Effects:
* Allocates memory and setup fonts
*********************************************************/
void InitFonts(void) {
  memset(&lfNormal,0,sizeof(LOGFONT));                              170
  lfNormal.lfHeight = LFHEIGHT;
  lfNormal.lfWidth = LFWIDTH;
  lstrcpy(lfNormal.lfFaceName,"Tms Rmn");
  hfontNormal = CreateFontIndirect(&lfNormal);

  memset(&lfKeyword,0,sizeof(LOGFONT));
  lfKeyword.lfHeight = LFHEIGHT;
  lfKeyword.lfWidth = LFWIDTH;
  lstrcpy(lfKeyword.lfFaceName,"Tms Rmn");
  lfKeyword.lfWeight = FW_BOLD;                                     180
  hfontKeyword = CreateFontIndirect(&lfKeyword);

  memset(&lfPlaceholder,0,sizeof(LOGFONT));
  lfPlaceholder.lfHeight = LFHEIGHT;
  lfPlaceholder.lfWidth = LFWIDTH;
  lstrcpy(lfPlaceholder.lfFaceName,"Tms Rmn");
  lfPlaceholder.lfItalic = 1;
  hfontPlaceholder = CreateFontIndirect(&lfPlaceholder);

  memset(&lfSymbol,0,sizeof(LOGFONT));                              190
  lfSymbol.lfHeight= -13;
  lfSymbol.lfWidth = 0;
  lfSymbol.lfWeight = FW_NORMAL;
  lfSymbol.lfCharSet = ',';  // can't explain this values !
  lfSymbol.lfPitchAndFamily = '2';  // can't explain this value !
  lstrcpy(lfSymbol.lfFaceName,"SYM14    P-1331874926");
  hfontSymbol = CreateFontIndirect(&lfSymbol);

  memset(&lfComment,0,sizeof(LOGFONT));
```

```
   lfComment.lfHeight = LFHEIGHT;
   lfComment.lfWidth = LFWIDTH;
   lstrcpy(lfComment.lfFaceName,"Tms Rmn");
   lfComment.lfItalic = 1;
   hfontComment = CreateFontIndirect(&lfComment);

   memset(&lfText,0,sizeof(LOGFONT));
   lfText.lfHeight = LFHEIGHT;
   lfText.lfWidth = LFWIDTH;
   lstrcpy(lfText.lfFaceName,"Tms Rmn");
   lfText.lfItalic = 1;
   hfontText = CreateFontIndirect(&lfText);

}

/*********************************************************
* The function open the choose font dialog from common
* dialogs and alters the desired font.
* Input:
* font - font to choose
* lf       - logical font information structure
* Output
* none
* Side Effects
* font - the choosed font
*********************************************************/
void ChangeFont(HFONT &hfont,LOGFONT &lf) {
   CHOOSEFONT cf;     // Common dialog box structure

   /* Initialize the necessary members */

   cf.lStructSize = sizeof(CHOOSEFONT);
   cf.hwndOwner = (HWND)NULL;
   cf.hDC = (HDC)NULL;
   cf.lpLogFont = &lf;
   cf.iPointSize = 0;
   cf.Flags = CF_SCREENFONTS | CF_LIMITSIZE | CF_INITTOLOGFONTSTRUCT;
   cf.rgbColors = RGB(0,0,0);
   cf.lCustData = 0L;
   cf.lpfnHook = NULL;
   cf.lpTemplateName = (LPSTR) NULL;
   cf.hInstance = (HINSTANCE) NULL;
   cf.lpszStyle = (LPSTR) NULL;
   cf.nFontType = SCREEN_FONTTYPE;
   cf.nSizeMin = 10;
   cf.nSizeMax = 20;

   // Access common choose font dialog
   if(ChooseFont(&cf)) {
      hfont=CreateFontIndirect(cf.lpLogFont);
      lf=*cf.lpLogFont;
   };

}

/*********************************************************
* Procedure for opening a file
* Input:
* hwnd - handle of window
* Output:
* none
* Side Effects:
* root may be altered, screen may be updated
*
*********************************************************/
void OpenFile(HWND hwnd) {
   // Place the terminating null character in szFile
```

```
  szFile[0]='\0';
  // Set the members of the OPENFILENAME structure
  ofn.lStructSize = sizeof(OPENFILENAME);
  ofn.hwndOwner=hwnd;                                                    270
  ofn.lpstrFilter=szFilter;
  ofn.nFilterIndex=1;
  ofn.lpstrFile=szFile;
  ofn.nMaxFile=sizeof(szFile);
  ofn.lpstrFileTitle=szFileTitle;
  ofn.nMaxFileTitle=sizeof(szFileTitle);
  ofn.lpstrInitialDir=szDirName;
  ofn.Flags= OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

  // Display the Open File dialog box                                    280
  if(GetOpenFileName(&ofn)) {

    // Open the file
    hf = CreateFile(ofn.lpstrFile, GENERIC_READ, 0,
          (LPSECURITY_ATTRIBUTES) NULL,
           OPEN_EXISTING,
           FILE_ATTRIBUTE_NORMAL,
           (HANDLE) 0);

    // Parse the the buffer into new tree                                290
    if(!Root->parse_file(hf)&&EditTree) {
      Root->delete_subtree(Root);
      Root->paste_subtree(EditTree);
      Root->select();
      SelectedNode=Root;
      // Update Screen
      updateScrollBars(hwndEditChild);
      InvalidateRect(hwndEditChild,NULL,TRUE);
      InvalidateRect(hwndPosChild,NULL,TRUE);
      InvalidateRect(hwndTransChild,NULL,TRUE);                          300

      DoCaption(hwnd,ofn.lpstrFile);
      lstrcpy(szFile,ofn.lpstrFile);
      LPTSTR str;
      GetFullPathName(ofn.lpstrFile,255,szDirName,&str);
    }

    // Close file
    CloseHandle(hf);
                                                                         310
  }
}

/*********************************************************
* Procedure for saving a file
* Input:
* hwnd - handle of window
* Output:
* none
* Side Effects:                                                          320
* root may be altered, screen may be updated
*
*********************************************************/
void SaveFile(HWND hwnd) {
  // Place the terminating null character in szFile
  szFile[0]='\0';
  // Set the members of the OPENFILENAME structure
  ofn.lStructSize = sizeof(OPENFILENAME);
  ofn.hwndOwner=hwnd;
  ofn.lpstrFilter=szFilter;                                              330
  ofn.nFilterIndex=1;
  ofn.lpstrFile=szFile;
  ofn.nMaxFile=sizeof(szFile);
```

```
  ofn.lpstrFileTitle=szFileTitle;
  ofn.nMaxFileTitle=sizeof(szFileTitle);
  ofn.lpstrInitialDir=szDirName;
  ofn.Flags= OFN_OVERWRITEPROMPT;

  // Display the Save As dialog Box
  if(GetSaveFileName(&ofn)) {                                                    340

    // (Re)Create the file
    hf = CreateFile(ofn.lpstrFile, GENERIC_WRITE, 0,
          (LPSECURITY_ATTRIBUTES) NULL,
           CREATE_ALWAYS,
           FILE_ATTRIBUTE_NORMAL,
           (HANDLE) 0);

    Root->unparse(to_file,hf);
                                                                                 350
    DoCaption(hwnd,ofn.lpstrFile);
    lstrcpy(szFile,ofn.lpstrFile);
    LPTSTR str;
    GetFullPathName(ofn.lpstrFile,255,szDirName,&str);

    // Closefile
    CloseHandle(hf);

    bNeedSave=FALSE;
  }                                                                              360
}

/*****************************************************************************
*
* Main Window Procedure
*
*****************************************************************************/
/*********************************************************
* Main window procedure.
* It creates three child windows, reacts on menu commands,                      370
* sends keyboard characters to the edit child window.
* Input:
* hwnd -
* message -
* wParam -
* lParam -
* Output:
* long -
* Side Effects:
*                                                                               380
*********************************************************/
long FAR PASCAL WndProc(HWND hwnd, unsigned message, WPARAM wParam, LPARAM lParam)
{
  HDC         hdc;
  TEXTMETRIC    tm;
  RECT        rect;         // rectangle of client
  static FARPROC lpfnEditUnpProc;
  static HANDLE  hInstance;


                                                                                390
  switch(message) {
    case WM_CREATE:
      hdc = GetDC(hwnd);

      InitFonts();

      GetTextMetrics(hdc,&tm);
      cxChar = tm.tmAveCharWidth;
      cyChar = tm.tmHeight + tm.tmExternalLeading;
                                                                                400
```

```
        ReleaseDC(hwnd,hdc);

        GetClientRect(hwnd,&rect);

        hwndEditChild = CreateWindow(szEditClass,NULL,
                                     WS_CHILD | WS_HSCROLL | WS_VSCROLL | WS_VISIBLE,
                                     0,0,rect.right,rect.bottom-TRANSAREAHEIGHT-16,
                                     hwnd,NULL,GetWindowNumber(hwnd),NULL);

        hwndPosChild = CreateWindow(szPosClass,NULL,                                    410
                                     WS_CHILD | WS_BORDER | WS_VISIBLE,
                                     0,rect.bottom-TRANSAREAHEIGHT-16,rect.right,16,
                                     hwnd,NULL,GetWindowNumber(hwnd),NULL);

        hwndTransChild = CreateWindow(szTransClass,NULL,
                                     WS_CHILD | WS_BORDER | WS_VISIBLE | WS_CLIPCHILDREN,
                                     0,rect.bottom-TRANSAREAHEIGHT,rect.right,TRANSAREAHEIGHT,
                                     hwnd,NULL,GetWindowNumber(hwnd),NULL);

/*                                                                                      420
        hInstance= ((LPCREATESTRUCT) lParam)->hInstance;
        lpfnEditUnpProc = MakeProcInstance((FARPROC)EditUnpProc,hInstance);
*/
        return 0;

      case WM_COMMAND:
        switch(wParam)
        {
          case IDM_NEW:
            if(bNeedSave && IDCANCEL == AskAboutSave(hwnd,szFile))                      430
              break;

            szFile[0] = '\0';
            DoCaption(hwnd,szFile);
            bNeedSave=FALSE;
            Root->delete_subtree(Root);
            Root->make_subtree();
            SelectedNode=Root;
            SelectedNode->select();
            InvalidateRect(hwndEditChild,NULL,TRUE);                                    440
            InvalidateRect(hwndPosChild,NULL,TRUE);
            InvalidateRect(hwndTransChild,NULL,TRUE);
            break;
          case IDM_OPEN:
            OpenFile(hwnd);
            break;
          case IDM_SAVE:
            if(szFile[0]) {
              hf = CreateFile(szFile, GENERIC_WRITE, 0,
                      (LPSECURITY_ATTRIBUTES) NULL,                                     450
                      CREATE_ALWAYS,
                      FILE_ATTRIBUTE_NORMAL,
                      (HANDLE) 0);

              Root->unparse(to_file,hf);

              // Closefile
              CloseHandle(hf);

              bNeedSave=FALSE;                                                          460
              break;
            }
            // fall through to save as dialog
          case IDM_SAVEAS:
            SaveFile(hwnd);
            break;
          case IDM_EXIT:
```

```
    SendMessage(hwnd, WM_CLOSE, NULL, 0L);
    break;
  case IDM_UNDO:                                                          470
    SendMessage(hwndEditChild, WM_UNDO, wParam, lParam);
    break;
  case IDM_CUT:
    SendMessage(hwndEditChild, WM_CUT, wParam, lParam);
    break;
  case IDM_PASTE:
    SendMessage(hwndEditChild, WM_PASTE, wParam, lParam);
    break;
  case IDM_COPY:
    SendMessage(hwndEditChild, WM_COPY, wParam, lParam);                  480
    break;
  case IDM_EDITUNP:
    //DialogBox DialogBox(hInstance, "EDITUNP", hwnd, lpfnEditUnpProc);
    MessageBox(hwnd, "Not implemented", szAppName, MB_ICONINFORMATION | MB_OK);
    break;
  case IDM_FONTS:
    MessageBox(hwnd, "Not implemented", szAppName, MB_ICONINFORMATION | MB_OK);
    break;
  case IDM_COLORS:
    MessageBox(hwnd, "Not implemented", szAppName, MB_ICONINFORMATION | MB_OK);  490
    break;
  case IDM_ABOUT:
    MessageBox(hwnd,
      "This is RSLED\n"
      "A Syntax Directed Editor for the\n"
      "RAISE Specification Language\n"
      "(c) 1993,1994 Michael Suodenjoki",
      szAppName, MB_ICONINFORMATION | MB_OK);
    break;
  case IDM_VERSION:                                                       500
    MessageBox(hwnd,
      "This is RSLED\n Alfa Release\n"
      "(c) 1993,1994 Michael Suodenjoki",
      szAppName, MB_ICONINFORMATION | MB_OK);
    break;
  case IDM_ASCEND_PARENT:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F1, 0L);
    break;
  case IDM_FORWARD_PREORDER:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F2, 0L);                    510
    break;
  case IDM_FORWARD_OPTIONAL:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F3, 0L);
    break;
  case IDM_FORWARD_SIBLING:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F4, 0L);
    break;
  case IDM_FORWARD_SIBLING_OPTIONAL:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F5, 0L);
    break;                                                                520
  case IDM_BACKWARD_PREORDER:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F6, 0L);
    break;
  case IDM_BACKWARD_OPTIONAL:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F7, 0L);
    break;
  case IDM_BACKWARD_SIBLING:
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F8, 0L);
    break;
  case IDM_BACKWARD_SIBLING_OPTIONAL:                                     530
    SendMessage(hwndEditChild, WM_KEYDOWN, VK_F9, 0L);
    break;
  case IDM_ASCIIREPRESENTATION:
    asciion=!asciion;
```

```
        CheckMenuItem(GetMenu(hwnd),IDM_ASCIIREPRESENTATION,
          asciion ? MF_BYCOMMAND | MF_CHECKED :
                    MF_BYCOMMAND | MF_UNCHECKED);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;
      case IDM_NORMALFONT:                                                    540
        ChangeFont(hfontNormal,lfNormal);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        InvalidateRect(hwndPosChild,NULL,TRUE);
        break;
      case IDM_KEYWORDFONT:
        ChangeFont(hfontKeyword,lfKeyword);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;
      case IDM_PLACEHOLDERFONT:
        ChangeFont(hfontPlaceholder,lfPlaceholder);                           550
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;
      case IDM_SYMBOLFONT:
        ChangeFont(hfontSymbol,lfSymbol);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;
      case IDM_COMMENTFONT:
        ChangeFont(hfontComment,lfComment);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;                                                                560
      case IDM_TEXTFONT:
        ChangeFont(hfontText,lfText);
        InvalidateRect(hwndEditChild,NULL,TRUE);
        break;
    }
    return 0;

  case WM_SIZE:
    int widthClient  = LOWORD(lParam);
    int heightClient = HIWORD(lParam);                                        570

    MoveWindow(hwndEditChild,0,0,widthClient,heightClient-TRANSAREAHEIGHT-16,TRUE);
    MoveWindow(hwndPosChild,0,heightClient-TRANSAREAHEIGHT-16,widthClient,16,TRUE);
    MoveWindow(hwndTransChild,0,heightClient-TRANSAREAHEIGHT,widthClient,TRANSAREAHEIGHT,TRUE);

    return 0;

  case WM_CHAR:
    bNeedSave=TRUE;
    SendMessage(hwndEditChild,message,wParam,lParam);                         580
    return 0;

  case WM_KEYDOWN:
    SendMessage(hwndEditChild,message,wParam,lParam);
    return 0;

  case WM_KILLFOCUS:
    bNeedSave=TRUE;
    wndOldFocus=hwnd;
    return 0;                                                                 590

  case WM_CLOSE:
    if(!bNeedSave || (IDCANCEL != AskAboutSave(hwnd,szFile)))
        DestroyWindow(hwnd);
    return 0;

  case WM_QUERYENDSESSION:
    if(!bNeedSave || (IDCANCEL != AskAboutSave(hwnd,szFile)))
      return 1L;
    return 0;                                                                 600
```

```
    case WM_DESTROY:
      DeleteObject(hfontNormal);
      DeleteObject(hfontKeyword);
      DeleteObject(hfontPlaceholder);
      DeleteObject(hfontSymbol);
      DeleteObject(hfontComment);
      DeleteObject(hfontText);

      if(Root) {                                                          610
        Root->delete_subtree(Root);
// if(Root->clip_buffer) Root->delete_subtree(clip_buffer);
// delete Root->clip_buffer;
        delete Root;
      }

      PostQuitMessage(0);
      return 0;
  }
  return DefWindowProc(hwnd,message,wParam,lParam);                       620
}


/*********************************************************
* Registers 4 child windows: Main, Edit, Pos and Transfor-
* mation window. Creates the Main Window, and makes
* appropiate calls to display them.
* Input:
* hInstance -
* hPrevInstance -                                                         630
* nCmdShow      -
* Output:
* int (bool) - Did all registering end ok.
* Side Effects:
* See description of GetMessage, TranslateMessage and
* DispatchMessage in Windows documentation.
*********************************************************/
int InitRSLEDWIN(HANDLE hInstance,HANDLE hPrevInstance,int nCmdShow) {
  WNDCLASS  wndclass;
                                                                          640
  if(!hPrevInstance) {
    // Main wndclass

    wndclass.style          = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNCLIENT;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(hInstance,szAppName);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);                 650
    wndclass.hbrBackground  = GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = "RSLMENU";
    wndclass.lpszClassName  = szAppName;

    if(!RegisterClass(&wndclass)) return FALSE;

    // Edit wndclass

    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = (WNDPROC) EditWndProc;                      660
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szEditClass;
```

```
    if(!RegisterClass(&wndclass)) return FALSE;                                    670

    // Position window

    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = (WNDPROC) PosWndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);                          680
    wndclass.hbrBackground  = GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szPosClass;

    if(!RegisterClass(&wndclass)) return FALSE;

    // Transformation window

    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = (WNDPROC) TransWndProc;                              690
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szTransClass;

    if(!RegisterClass(&wndclass)) return FALSE;                                    700

  }

  hwndMain = CreateWindow(szAppName, "RSLED (alfa release)",
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL,NULL,hInstance,NULL);

  if(!hwndMain) return FALSE;                                                      710

  if(SelectedNode) SelectedNode->select();

  ShowWindow(hwndMain,nCmdShow);
  UpdateWindow(hwndMain);

  return TRUE;
}

/*********************************************************                          720
* This function do the essential message loop in Windows
* Input:
* none
* Output:
* int - return message when program stops
* Side Effects:
* See description of GetMessage, TranslateMessage and
* DispatchMessage in Windows documentation.
*********************************************************/
int MessageLoop(void) {                                                            730
  MSG msg;

  while(GetMessage(&msg,NULL,0,0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
```

```
  }
  return msg.wParam;
}

/*************************************************************
*
* MAIN WINDOW FUNCTION
*
*************************************************************/

// This function is the main window function.
// It registers 4 child windows: Main, Edit, Pos
// and Transformation window.
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdLine, int nCmdShow)
{
  HWND        hDlgIntro;
  FARPROC     lpProcIntro;
  WORD        wCount;
  MSG         aMsg;

  // Make a instance thunk for the Intro dialog procedure
  lpProcIntro = MakeProcInstance((FARPROC)Intro,hInstance);

  // Create the modeless dialog
  DialogBox(hInstance,"INTRO",0,lpProcIntro);

  // Free up the instance thunk. Is obsolete in Win32 !
  #ifndef __NT__
  FreeProcInstance(lpProcIntro);
  #endif

  if (!InitRSLEDWIN(hInstance,hPrevInstance,nCmdShow)) return FALSE;
  return MessageLoop();
}
```

G.12 SYNNODE.CPP

```
/*************************************************************************
*
* SYNNODE.CPP
*
* Implementation of Syntax Tree Nodes
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/

#ifndef SYNNODE_CPP
#define SYNNODE_CPP

#include "SYNNODE.H"

/*********************************************************
* Constructor for SyntaxTreeNode
* Input:
* nid - node identificator
* rp - is node a resting_place
* ln - is node a listnode
* op - is node an optional
* Output:
* none
* Side Effects:
```

```
* Sets the class variables appropiately
*********************************************************/
SyntaxTreeNode::SyntaxTreeNode() : selected(FALSE), listnode(FALSE),
   optional(FALSE), resting_place(TRUE), nodeid(0), elided(FALSE)
{                                                                    30
 set_view(0,0,0,0);
}

/*********************************************************
* Set the view (x,y) coordinates and width and height
* Input:
* x,y - (x,y) coordinate
* width - width of view
* height - height of view
* Output:                                                            40
* none
* Side Effects:
* none
*********************************************************/
void SyntaxTreeNode::set_view(const int x,const int y,
                              const unsigned int width ,
                              const unsigned int height)
{
   view.x=x; view.y=y;
   view.width=width;                                                 50
   view.height=height;
}

/*********************************************************
* Return the current view of the node
* Input:
* none
* Output:
* VIEW - the current view
* Side Effects:                                                      60
* none
*********************************************************/
VIEW SyntaxTreeNode::get_view(void) const { return view; }

/*********************************************************
* Checks whether (x,y) coordinates is inside view
* Input:
* x,y - (x,y) coordinate
* Output:
* BOOL - is TRUE is (x,y) is inside view                             70
* Side Effects:
* none
*********************************************************/
BOOL SyntaxTreeNode::isinview(const int x,const int y) {
   return x>=view.x&&x<=view.x+view.width&&y>=view.y&&y<=view.y+view.height;
}

/*********************************************************
* Checks whether a view overlaps the treenodes view
* Input:                                                             80
* view - view to check
* Output:
* BOOL - is TRUE is the view overlaped
* Side Effects:
* none
*********************************************************/
BOOL SyntaxTreeNode::view_overlaps_view(VIEW v) {
   return v.x < view.x+view.width   &&
          v.x+v.width >= view.x     &&
          v.y < view.y+view.height  &&                               90
          v.y+v.height >= view.y;
}
```

```
/*********************************************************
* Checks if a view isin the tree nodes view
* Input:
* view - view to check
* Output:
* BOOL - returns TRUE is view isin the treenodes view
* Side Effects:                                                    100
* none
*********************************************************/
BOOL SyntaxTreeNode::view_isin_view(VIEW v) {
  return v.x >= view.x && v.x+v.width <= view.x+view.width &&
         v.y >= view.y && v.y+v.height <= view.y+view.height;
}



/*********************************************************          110
* Comares two syntax nodes
* Input:
* sn1, sn2 - the two nodes to compare
* Output:
* BOOL - returns TRUE if the nodes are identical
* Side Effects:
* none
*********************************************************/
BOOL SyntaxTreeNode::compare(const PSYNNODE sn1,const PSYNNODE sn2) const {
  return sn1->get_nodeid()==sn2->get_nodeid();                     120
}

#endif /* SYNNODE_CPP */
```

G.13 SYNTREE.CPP

```
/*************************************************************************
*
* SYNTREE.CPP
*
* Implementation of Abstract Syntax Tree
*
* Created 19 December, 1993, Michael Suodenjoki
*
*************************************************************************/
                                                                   10
#ifndef SYNTREE_CPP
#define SYNTREE_CPP

#include "SYNTREE.H"

#include "MYSTRING.H"
#include "LANGUAGE.H"
#include "SCANNER.H"
#include "PARSER.H"
                                                                   20
// Is ASCII representation on ?
extern BOOL asciion;
extern int ViewOrgY;

// Definition of the Clip Buffer
PSYNTAXTREE SyntaxTree::clip_buffer = new SyntaxTree;
// Is multiple nodes clipped into buffer ?
BOOL SyntaxTree::mult_clipped = FALSE;
```

```
/*********************************************************
* Constructor for Syntax-Tree
* The function should idealy have an arg-list as
* argument.
* Input:
* none
* Output:
* none
* Side Effects
* Calls NaryTree() and SyntaxTreeNode() constructors
*********************************************************/
SyntaxTree::SyntaxTree() {}

/*********************************************************
* Inserts up to 5 trees as child for the (this) node.
* The number of parameter should be variable.
* Input:
* tree1-tree5 - the 5 trees.
* Output:
* none
* Side Effects:
* The tree is changed appropiately
*********************************************************/
void SyntaxTree::insert(PSYNTAXTREE tree1,
                        PSYNTAXTREE tree2,
                        PSYNTAXTREE tree3,
                        PSYNTAXTREE tree4,
                        PSYNTAXTREE tree5)
{
  delete_subtree(this);

  insert_tree_last(tree1);
  if(tree2) insert_tree_last(tree2);
  if(tree3) insert_tree_last(tree3);
  if(tree4) insert_tree_last(tree4);
  if(tree5) insert_tree_last(tree5);
}

/*************************************************************************
*
* MOVEMENTS COMMANDS
*
*************************************************************************/
PSYNTAXTREE SyntaxTree::ascend_parent(BOOL& flag) {
  PSYNTAXTREE temp = this;
  PSYNTAXTREE t;
  flag=FALSE;
  temp->SyntaxTreeNode::de_select();
  do {
    if(temp->get_parent()) {
      t=temp;
      temp=(PSYNTAXTREE) temp->get_parent();
      delete_optional(t,flag);
      delete_listnode(t,flag);
    }
  }
  while(!temp->is_resting_place()&&temp->get_parent());
  temp->select();
  return temp;
}

/** Forward preorder **/

PSYNTAXTREE SyntaxTree::forward_preorder(BOOL& flag) {
  return forward_preorder(FALSE,FALSE,flag);
};
```

```
PSYNTAXTREE SyntaxTree::forward_preorder_with_optionals(BOOL& flag) {
  return forward_preorder(TRUE,FALSE,flag);
}
                                                                        100
PSYNTAXTREE SyntaxTree::forward_sibling(BOOL& flag) {
  return forward_preorder(FALSE,TRUE,flag);
}

PSYNTAXTREE SyntaxTree::forward_sibling_with_optionals(BOOL& flag) {
  return forward_preorder(TRUE,TRUE,flag);
}

/** Backward preorder **/
                                                                        110
PSYNTAXTREE SyntaxTree::backward_preorder(BOOL& flag) {
  return backward_preorder(FALSE,FALSE,flag);
}

PSYNTAXTREE SyntaxTree::backward_preorder_with_optionals(BOOL& flag) {
  return backward_preorder(TRUE,FALSE,flag);
}

PSYNTAXTREE SyntaxTree::backward_sibling(BOOL& flag) {
  return backward_preorder(FALSE,TRUE,flag);                            120
}

PSYNTAXTREE SyntaxTree::backward_sibling_with_optionals(BOOL& flag) {
  return backward_preorder(TRUE,TRUE,flag);
}

/** Select/Deselect **/

/*********************************************************
* Deselect the node. Take in consideration to delete                    130
* optional nodes. If the (x,y)-view-coordinates are outside
* view to deselect, then maybe delete optionals or
* listnodes.
* Input:
* x,y - view coordinates
* multlist - if multiple list selected then deselect
* them all
* flag - return value, has anything been deleted
* Output:
* See flag input parameter                                              140
* Side Effects:
* Maybe some nodes in the tree are deleted => affects
* flag parameter.
*********************************************************/
void SyntaxTree::de_select(const int x,const int y,BOOL &flag)
{
  PSYNTAXTREE temp = this;
  PSYNTAXTREE t =  NULL;
  flag=0;
  while(temp&&!temp->isinview(x,y)) {                                   150
    temp->delete_optional(temp,flag);
    temp->delete_listnode(temp,flag);
    temp=(PSYNTAXTREE) temp->get_parent();
  }
  NaryTree::de_select();
}

/*********************************************************
* Deselect multiple listnodes.
* Input:                                                                160
* multlist - are multiple list selected
* Output:
* (see multlist input parameter)
```

```
* Side Effects:
*
**********************************************************/
void SyntaxTree::de_select_mult(BOOL &multlist)
{
  if(multlist) {
    PSYNTAXTREE temp = (PSYNTAXTREE) get_parent()->get_firstchild();
    while(temp) {
      temp->NaryTree::de_select();
      temp=(PSYNTAXTREE) temp->get_nextsibling();
    }
    NaryTree::de_select();
  }
  multlist=FALSE;
}

/**********************************************************
* The function checks if the coordinates given as parameters
* is lying inside a given view in the tree.
* - a pointer to itself (this), if the function couldn't
* find a view where coords where inside, or
* - a pointer to the node where the coords lies inside
* the nodes view.
* Input:
* x,y - coordinates
* Output:
* PSYNNODE - returning node
* Side Effects:
* none
**********************************************************/
PSYNTAXTREE SyntaxTree::belongs_to(const int x,const int y) {
  PSYNTAXTREE tree = this;
  PSYNTAXTREE temp = this;
  BOOL found;

  do {
    if(tree&&tree->isinview(x,y)&&
      (tree->is_resting_place()||tree->is_elided())) temp=tree;
    found=FALSE;
    if(tree->get_firstchild()&&!tree->is_elided())
      tree=(PSYNTAXTREE) tree->get_firstchild();
    else if(!(tree==temp&&tree->is_elided())) {
      if(tree->get_nextsibling())
        tree=(PSYNTAXTREE) tree->get_nextsibling();
      else while(tree->get_parent()&&!found) {
        tree=(PSYNTAXTREE) tree->get_parent();
        if(tree==temp)
          found=TRUE;
        else
        if(tree->get_nextsibling()) {
          tree=(PSYNTAXTREE) tree->get_nextsibling();
          found=TRUE;
        }
      }
    }
  }
  while(tree!=temp);

  return tree;
}

/**********************************************************
* Unparses a nodes. The function collect the unparsing
* scheme from the NodeInfoTable.
* Input:
*
* Output:
```

```
* none
* Side Effects:
* Can effect the hole screen, a file, a buffer.
* (Write more here)
*******************************************************/
void SyntaxTree::unparse(OUTPUT output,FILEHANDLE file) {
  int x = 0, y =0;

  switch(output) {
    case none:                                                              240
      unparse_symbol_to_screen(NodeInfoTable[get_nodeid()].unp_scheme,
                               (PSYNTAXTREE)get_firstchild(),x,y,FALSE,0,0);
      break;
    case to_screen:
      unparse_symbol_to_screen(NodeInfoTable[get_nodeid()].unp_scheme,
                               (PSYNTAXTREE)get_firstchild(),x,y,TRUE,0,0);
      break;
    case to_file:
      unparse_symbol_to_file(file,NodeInfoTable[get_nodeid()].unp_scheme,
                               (PSYNTAXTREE)get_firstchild(),x,y,TRUE,0,0);  250
  }
}


/* Scanner/Parser */

class MyScanner: public SCANNER {
public:
  FILEHANDLE infile;
  virtual int yy_input(char *buf,int &result,int max_size);                 260
  void yy_fatal_error(char *msg) {error(msg,"Scanner error");}
};

int MyScanner::yy_input(char *buf,int &result,int max_size)
{
  if(!readfile(infile,buf,max_size,result))
    yy_fatal_error("Fileread error");
  return result;
}
                                                                            270
/* Parser */
class MyParser: public PARSER {
private:
  MyScanner myscanner;
public:
  MyParser();
  MyParser(FILEHANDLE file) { myscanner.infile=file; }

  virtual int yylex();
  virtual void yyerror(char *);                                             280
};

int MyParser::yylex() {
  yylloc.first_line=myscanner.theLine;
// yylloc.first_column=myscanner.theColumn;
  int token=myscanner.yylex();
  yylloc.last_line=myscanner.theLine;
// yylloc.last_column=myscanner.theColumn;
  yylloc.text=(char *) myscanner.yytext;
  return token;                                                             290
}

void MyParser::yyerror(char *msg) {
  char msgbuf[255];
  sprintf(msgbuf,"%d: %s at token '%s'\n", yylloc.first_line,msg,yylloc.text);
  error(msgbuf,"Parser error");
}
```

```
/*********************************************************
* Parse a file. The (correct) file contents should
* could be parsed to a tree with node identifier nid.
* Input:
* file - filehandle
* Output:
* BOOL - is TRUE if parsing went ok
* Side Effects:
*
*********************************************************/
BOOL SyntaxTree::parse_file(FILEHANDLE file) {
    // Initialise parser
    MyParser myparser(file);

    // Call parser
    return myparser.yyparse();
}

/*********************************************************
* Creates a number of subtrees for the syntax tree node.
* The main purpose of this function is to make it available
* for inherited classes of SyntaxTreeNode.
* For now it does nothing !
* Input:
* node
* Output:
* node
* Side Effects:
* The pointers to parent, childs and siblings could
* be affected accordingly to what this function do.
*********************************************************/
void SyntaxTree::make_subtree(void) {}

/*********************************************************
* Should unparse the node. The main purpose of this function
* is that it is overriden for special nodes like
* Identifier or Integer. These are all terminals (or one
* could say lexical identifiers), but they all have
* different unparsing.
* Input:
* none
* Output:
* char* - the unparsed text
* Side Effects:
* none
*********************************************************/
char *SyntaxTree::unparse_special(void) const { return NULL; }


/*********************************************************
* It is intended that this function is overwritten for
* derived classes. It is supposed that each syntactic
* node (which have subnodes), should override this funtion
* and return a pointer to a new allocated syntactic node.
* Input:
* none
* Output:
* PSYNTAXTREE - pointer to the new syntax tree/node
* Side Effects:
* Allocates memory on the heap
*********************************************************/
PSYNTAXTREE SyntaxTree::make_child(void) { return NULL; }

/*********************************************************
* Cut/Copy the tree to the clipped buffer
* Input:
```

```
* mult_lists - is TRUE if several nodes are selected
* copy - is TRUE if copy_to_clipped instead
* Output:
* none
* Side Effects:
* The tree and clip buffer changes, allocation on                                    370
* the heap
************************************************************/
void SyntaxTree::cut_to_clipped(BOOL mult_lists,BOOL copy) {
  // Delete previous contents of clip buffer
  if(clip_buffer) clip_buffer->delete_subtree(clip_buffer);

  PSYNTAXTREE temp;
  if(mult_lists) {
    temp = (PSYNTAXTREE) get_parent();
    clip_buffer=temp->copy_tree();                                                   380

    // Remove unselected listnodes from clipbuffer;
    PSYNTAXTREE t;
    temp=(PSYNTAXTREE) clip_buffer->get_firstchild();
    while(temp) {
      t=(PSYNTAXTREE) temp->get_nextsibling();
      if(!temp->is_selected()) {
        temp->delete_subtree(temp);
        temp->delete_treenode(temp);
      }                                                                              390
      temp=t;
    }
    // Cut selected lists
    if(!copy) {
      temp=(PSYNTAXTREE) get_parent()->get_firstchild();
      while(temp) {
        t=(PSYNTAXTREE) temp->get_nextsibling();
        if(temp->is_selected()) {
          temp->delete_subtree(temp);
          if(t&&t->is_selected())                                                    400
            temp->delete_treenode(temp);
          else this=temp;
        }
        temp=t;
      }
      make_subtree();
    }

    mult_clipped=TRUE;
  }                                                                                  410
  else {
    // Insert new contents in clip buffer
    clip_buffer=copy_tree();

    if(!copy) {
      // Remove connections to the cutted subtrees
      cut_subtree();

      make_subtree();
    }                                                                                420
    mult_clipped=FALSE;
  }
}

/*********************************************************
* Paste from the clipped buffer
* Input:
* none
* Output:
* bool - is true if pasting went ok                                                  430
* Side Effects:
```

```
* The tree changed and some allocation on the heap occur
*****************************************************/
BOOL SyntaxTree::paste_from_clipped(BOOL &mult_lists) {
  if(!clip_buffer||
     (!mult_clipped&&clip_buffer->get_nodeid()!=get_nodeid())||
     (mult_clipped&&clip_buffer->get_nodeid()!=get_parent()->get_nodeid()))
  {
    error("Cannot paste clipped buffer here.\nIncompatible trees !","Paste Error");
    return FALSE;                                                          440
  }
  else {
    // Delete previous contents of selected node
    delete_subtree(this);

    // Copy Clipbuffer
    PSYNTAXTREE temp = clip_buffer->copy_tree();

    if(mult_clipped) {
      PSYNTAXTREE t = temp;                                                450
      // Paste multi clipped listnodes into tree
      temp=(PSYNTAXTREE) temp->get_firstchild();
      PSYNTAXTREE t1,t3;
      t1=this;
      do {
        t3=(PSYNTAXTREE) temp->get_nextsibling();
        t1->insert_tree_after(temp);
        t1=(PSYNTAXTREE) t1->get_nextsibling();
        temp=t3;
      } while(temp);                                                       460

      delete_treenode(this);
      this=(PSYNTAXTREE) t->get_firstchild();
      delete t;
      mult_lists=TRUE;
    }
    else paste_subtree(temp);
  }
  return TRUE;
}                                                                          470

/*******************************************************
* It is intended that this function is overwritten for
* derived classes. It is supposed that each syntactic
* node (which have subnodes), should override this funtion
* and return a pointer to a new allocated copy of
* the syntactic node.
* Input:
* none
* Output:                                                                  480
* PSYNTAXTREE - pointer to the new syntax tree/node
* Side Effects:
* Allocates memory on the heap
*******************************************************/
PSYNTAXTREE SyntaxTree::copy_tree(void)  {
  PSYNTAXTREE copyfrom = this;
  PSYNTAXTREE copyto=copyfrom->clone();
  copyto->null();
  PSYNTAXTREE t;
  BOOL found;                                                              490
  do {
    if(copyfrom->get_firstchild()) {
      copyfrom=(PSYNTAXTREE) copyfrom->get_firstchild();
      t=copyfrom->clone();
      t->null();
      copyto->insert_tree_first(t);
      copyto=(PSYNTAXTREE) copyto->get_firstchild();
    }
```

```
    else if(copyfrom->get_nextsibling()) {
      copyfrom=(PSYNTAXTREE) copyfrom->get_nextsibling();
      t=copyfrom->clone();
      t->null();
      copyto->insert_tree_after(t);
      copyto=(PSYNTAXTREE) copyto->get_nextsibling();
    }
    else {
      found=FALSE;
      while(copyfrom->get_parent()&&copyfrom!=this&&!found) {
        copyfrom=(PSYNTAXTREE) copyfrom->get_parent();
        copyto=(PSYNTAXTREE) copyto->get_parent();

        if(copyfrom!=this&&copyfrom->get_nextsibling()) {
          copyfrom=(PSYNTAXTREE) copyfrom->get_nextsibling();
          t=copyfrom->clone();
          t->null();
          copyto->insert_tree_after(t);
          copyto=(PSYNTAXTREE) copyto->get_nextsibling();
         found=TRUE;
        }
      }
    }
  }
  while(copyfrom!=this);
  return copyto;
}

/*********************************************************
* It is intended that this function is overriden for
* derived classes. The function should make a copy of
* it self.
* Input:
* none
* Output:
* pointer to the new copy
* Side Effects:
* memory allocation on the heap
*********************************************************/
PSYNTAXTREE SyntaxTree::clone() const { return NULL; }

/************ PRIVATE ROUTINES ************************************/

// A move from one node to another implicates :
// 1) the node which you move from should be deleted if
// it is an unused optional or an unused listnode and it's
// not the last one.
// 2) the node to which you move could be an optional, which
// then must inserted before the move, or it could be a
// newly inserted listnode.
#define MOVETO(x) t=temp; temp=(PSYNTAXTREE)temp->get_ ## x ##(); \
                  insert_optional(temp,opt,flag);


/*********************************************************
* Return the next node in a forward preorder traversal of
* the tree.
* Input:
* opt - While traversing this parameter specifies
* that optionals or listnodes should be inserted.
* bypass - If TRUE then proceed to siblings instead of
* childs.
* flag -
* Output:
* Pointer to the next node in a forward preorder traversal
* Side Effects:
* Since optionals or listnodes can be inserted or
```

```
* deleted the hole tree can change.
*******************************************************/
PSYNTAXTREE SyntaxTree::forward_preorder(const BOOL opt,
                                         const BOOL bypass,
                                         BOOL& flag)                    570
{
  PSYNTAXTREE temp = this;
  PSYNTAXTREE t;
  BOOL found;

  flag=FALSE;

  temp->SyntaxTreeNode::de_select();
  do {
    found=FALSE;                                                        580
    if(!bypass) temp->insert_first(temp,opt,flag); // maybe insert either optional or listnode
    if(!bypass&&temp->get_firstchild()&&!temp->is_elided()) {
       MOVETO(firstchild);
    }
    else {
      temp->insert_after(temp,opt,flag); // maybe insert listnode
      if(temp->get_nextsibling()) {
        MOVETO(nextsibling)
        temp->delete_listnode(t,flag);
        temp->delete_optional(t,flag);                                  590
      }
      else
      while(temp->get_parent()&&!found) {
        MOVETO(parent);
        temp->delete_listnode(t,flag);
        temp->delete_optional(temp,flag);

        temp->insert_after(temp,opt,flag);
        if(temp->get_nextsibling()) {
          MOVETO(nextsibling);                                          600
          temp->delete_listnode(t,flag);
          temp->delete_optional(t,flag);
          found=TRUE;
        }
      }
    }
  }
  while(!temp->is_resting_place()&&(!opt||!temp->is_optional()||!bypass));
  temp->select();
  return temp;                                                          610
}

/*********************************************************
* Return the next node in a backward preorder traversal of
* the tree.
* Input:
* opt - While traversing this parameter specifies
* that optionals or listnodes should be inserted.
* bypass - If TRUE then proceed to siblings instead of
* childs.                                                               620
* flag -
* Output:
* Pointer to the next node in a backward preorder traversal
* Side Effects:
* Since optionals or listnodes can be inserted or
* deleted the hole tree can change.
*******************************************************/
PSYNTAXTREE SyntaxTree::backward_preorder(const BOOL opt,
                                          const BOOL bypass,
                                          BOOL& flag)                   630
{
  PSYNTAXTREE temp = this;
```

```
  PSYNTAXTREE t;

  flag=FALSE;

  temp->SyntaxTreeNode::de_select();
  do {
    temp->insert_before(temp,opt,flag);
    if(temp->get_prevsibling()) {                                          640
      MOVETO(prevsibling);
      temp->delete_optional(t,flag);
      temp->delete_listnode(t,flag);
      if(!bypass) {
        temp->insert_after((PSYNTAXTREE) temp->get_lastchild(),opt,flag); // maybe insert optional or listnode as
        while(temp->get_lastchild()&&!temp->is_elided()) {
          /*if(!temp->is_elided())*/ MOVETO(lastchild);
          temp->insert_after((PSYNTAXTREE) temp->get_lastchild(),opt,flag);
        }
      }                                                                    650
    }
    else if(temp->get_parent()) {
      MOVETO(parent);
      temp->delete_optional(t,flag);
      temp->delete_listnode(t,flag);
    }
    else if(!bypass) {
      temp->insert_after((PSYNTAXTREE) temp->get_lastchild(),opt,flag);
      while(temp->get_lastchild()&&!temp->is_elided()) {
        MOVETO(lastchild);                                                 660
      }
    }
  }
  while(!temp->is_resting_place()&&(!opt||!temp->is_optional()||!bypass));
  temp->select();
  return temp;
}

/************* PRIVATE FUNCTIONS (1) ************************/
                                                                           670
/*********************************************************
* This function insert childs for unused optionals.
* It is indetended that it is called after a move in the
* tree with first parameter as the node which is moved to.
* Input:
* tree -
* opt - is TRUE if it is allowed to insert optionals
* flag -
* Output:
* none                                                                    680
* Side Effects:
* chould changes flag parameter and the syntax tree
*********************************************************/
void SyntaxTree::insert_optional(const PSYNTAXTREE tree,const BOOL opt,
                                 BOOL &flag)
{
  if(opt&&tree->is_optional()&&!tree->get_firstchild()) {
    tree->insert_tree_first(tree->make_child());
    flag=TRUE;
  }                                                                        690
}

/*********************************************************
* The function deals with non-optional listnodes only and
* insert a new listnode as the first child (first element
* in the list).
* Input:
* tree - place in abstract syntax tree to insert
* opt - is TRUE is optionals should be inserted
```

```
* flag - (no input effect) is TRUE if any node has been
* inserted (or deleted)
* Output:
* see flag input parameter could change
* Side Effects:
*
*************************************************************/
void SyntaxTree::insert_first(const PSYNTAXTREE tree,const BOOL opt,
                              BOOL &flag)
{
  if(opt&&tree&&!tree->get_firstchild()&&tree->is_listnode()&&
     !tree->is_optional())
  {
    tree->make_subtree();
    flag=TRUE;
  }
  else
  if(opt&&tree&&tree->get_parent()&&tree->get_firstchild()&&tree->is_listnode()) {
    PSYNTAXTREE t = ((PSYNTAXTREE) tree)->make_child();
    if(!compare_trees(tree->get_firstchild(),t)) {
      tree->get_firstchild()->insert_tree_before(((PSYNTAXTREE) tree)->make_child());
      flag=1;
    }
    delete_subtree(t);
    delete_treenode(t);
  }
}

/*************************************************************
* Insert an optional node/tree before another tree.
* Input:
* tree - place in abstract syntax tree to insert
* opt - is TRUE if optionals should be inserted
* flag - (no input effect) is TRUE if any node has been
* inserted (or deleted)
* Output:
* see flag input parameter could change
* Side Effects:
*
*************************************************************/
void SyntaxTree::insert_before(const PSYNTAXTREE tree,const BOOL opt,
                               BOOL &flag)
{
  if(opt&&tree&&tree->get_parent()&&tree->get_parent()->is_listnode()) {
    PSYNTAXTREE t = ((PSYNTAXTREE) tree->get_parent())->make_child();
    if(!compare_trees(tree,t)) {
      tree->insert_tree_before(
        ((PSYNTAXTREE) tree->get_parent())->make_child());
      flag=TRUE;
    }
    delete_subtree(t);
    delete_treenode(t);
  }
}

/*************************************************************
* Inserts an optional node/tree after another tree
* Input:
* tree - place in abstract syntax tree to insert
* opt - is TRUE if optionals should be inserted
* flag - (no input effect) is TRUE if any node has been
* inserted (or deleted)
* Output:
* see flag input parameter could change
* Side Effects:
*
*************************************************************/
```

```
void SyntaxTree::insert_after(const PSYNTAXTREE tree,const BOOL opt,
                              BOOL &flag)
{
  if(opt&&tree&&tree->get_parent()&&tree->get_parent()->is_listnode()) {
    PSYNTAXTREE t = ((PSYNTAXTREE) tree->get_parent())->make_child();
    if(!compare_trees(tree,t)) {
      tree->insert_tree_after(
        ((PSYNTAXTREE) tree->get_parent())->make_child());
      flag=TRUE;
    }
    delete_subtree(t);
    delete_treenode(t);
  }
}

/*********************************************************
* Deletes optional and/or listnodes.
* Input:
* tree - place in abstract syntax tree to delete
* flag - (no input effect) is TRUE if any node has been
* inserted (or deleted)
* Output:
* see flag input parameter could change
* Side Effects:
*
*********************************************************/
void SyntaxTree::delete_optional(const PSYNTAXTREE tree,BOOL &flag) {
  // Delete optional
  if(tree->is_optional()) {
    PSYNTAXTREE t = tree->make_child();
    if(compare_trees(tree->get_firstchild(),t)) {
      delete_subtree(tree);
      flag=TRUE;
    }
    delete_subtree(t);
    delete_treenode(t);
  }
}

/*********************************************************
* Deletes listnodes.
* Input:
* tree - place in abstract syntax tree to delete
* flag - (no input effect) is TRUE if any node has been
* inserted (or deleted)
* Output:
* see flag input parameter could change
* Side Effects:
*
*********************************************************/
void SyntaxTree::delete_listnode(const PSYNTAXTREE tree,BOOL &flag) {
  // Delete listnode
  if(tree->get_parent()&&tree->get_parent()->is_listnode()&&
     !tree->is_optional()&&tree->get_parent()->get_firstchild()!=
     tree->get_parent()->get_lastchild())
  {

    PSYNTAXTREE t1 = ((PSYNTAXTREE) tree->get_parent())->make_child();
    unsigned int minfixed,actual;
    minfixed = tree->get_parent()->get_minfixed();
    // find actual number of childs for parent
    actual = count_childs(tree->get_parent());

    if(compare_trees(tree,t1)&&actual>minfixed) {
      delete_subtree(tree);
      delete_treenode(tree);
      flag=TRUE;
```

```
    }
    delete_subtree(t1);
    delete_treenode(t1);
  }
}

/************** PRIVATE FUNCTIONS (2) ************************/



// Prototypes for utility function of unparse
FONT select_symbol(BOOL,BOOL,int &,int &,VIEW &,FILEHANDLE);

// Number of special symbols
const unsigned int NUMSYMBOLS = 70;

typedef struct {
  char *ascii;  // ASCII representation of symbol
  char *symbol; // Special character representation
} SYMBOLINFO;

// Symbol table for special symbols.
// Contains ASCII representation and Special font representation
static SYMBOLINFO SymbolTable[NUMSYMBOLS] = {
  { "!!",       "\x20" },
  { "-inflist", "\x21" },
  { "#",        "\x22" },
  { "<->",      "\x26" },
  { "union",    "\x27" },
  { "<.",       "\x28" },
  { ".>",       "\x29" },
  { "-list",    "\x2A" },
  { "+>",       "\x2B" },
  { "++",       "\x2D" },
  { ":-",       "\x2E" },
  { "/\\",      "\x2F" },
  { "-~->",     "\x39" },
  { "isin",     "\x3A" },
  { "~isin",    "\x3B" },
  { "<=",       "\x3C" },
  { "is",       "\x3D" },
  { ">=",       "\x3E" },
  { "~=",       "\x3F" },
  { "all",      "\x41" },
  { "always",   "\x42" },
  { "|=|",      "\x43" },
  { "`Delta",   "\x44" },
  { "exists"    "\x45" },
  { "`Phi",     "\x46" },
  { "`Gamma",   "\x47" },
  { "=>",       "\x49" },
  { "`Lambda",  "\x4C" },
  { "-m->",     "\x4D" },
  { "|^|",      "\x4E" },
  { "`Omega",   "\x4F" },
  { "`Pi",      "\x50" },
  { "`Theta",   "\x51" },
  { "`Sigma",   "\x53" },
  { "><",       "\x54" },
  { "`Upsilon", "\x55" },
  { "`Xi",      "\x58" },
  { "`Psi",     "\x59" },
  { "<<=",      "\x5B" },
  { "\\/",      "\x5C" },
  { ">>=",      "\x5D" },
  { "**",       "\x5E" },
```

```
  { "->",       "\x5F" },
  { "inter",    "\x60" },
  { "`alpha",   "\x61" },
  { "`beta",    "\x62" },
  { "`chi",     "\x63" },
  { "`delta",   "\x64" },
  { "`epsilon", "\x65" },
  { "`phi",     "\x66" },
  { "`gamma",   "\x67" },
  { "`eta",     "\x68" },
  { "`iota",    "\x69" },
  { "`kappa",   "\x6B" },
  { "-\\",      "\x6C" },
  { "`mu",      "\x6D" },
  { "`nu",      "\x6E" },
  { "`omega",   "\x6F" },
  { "`pi",      "\x70" },
  { "`theta",   "\x71" },
  { "`rho",     "\x72" },
  { "`sigma",   "\x73" },
  { "`tau",     "\x74" },
  { "`upsilon", "\x75" },
  { "`xi",      "\x78" },
  { "`psi",     "\x79" },
  { "`zeta",    "\x7A" },
  { "<<",       "\x7B" },
  { "||",       "\x7C" },
  { ">>",       "\x7D" }};

// Defines for unparsing symbols

#define BeginFont      0
#define EndFont        1
#define Newline        2
#define Indent         3
#define Unindent       4
#define OptNewline     5
#define GrpNewline     6
#define BeginGroup     7
#define EndGroup       8
#define BeginInGroup   9
#define EndInGroup     10
#define Percent        11
#define Colon          12
#define Special        13
#define EOL            14
#define Child          15
#define Text           16
#define Undefined      17

// Table of unparsing symbols. Should be coherent with the numbers
// (defines) of the unparsing symbols.
const char *symbtab[] =
  { "%(", "%)", "%n", "%t", "%b", "%o", "%c",
    "%{",  "%}", "%[", "%]", "%%",  "%:", "%@"};

// Global (private) definition of symbol text
static char symbtxt[200];

// Global variable containing amound of tabbing units (in pixels)
const unsigned int tabbing = 8;
const unsigned int filetab = 2;

/*******************************************************
* Returns the next unparsing symbol.
* Input:
```

```
* txt - search text
* cNb - position to start search in text
* Output:
* SYMBOL - the unparsing symbol returned
* Side Effects:
* Changes the current start search position (cNb) in test
* Changes the out variable to contain the just read text
*********************************************************/
SYMBOL SyntaxTree::nextsymbol(const char *txt,unsigned int &cNb) {
  unsigned int j; // temporary search number/char position

  if(txt==NULL) return EOL;
  switch(txt[cNb]) {
    case '\0': return EOL;
    case '@' : cNb++;
               return Child ;
    case '%' : for(j=0;j<EOL ;j++)
               if(txt[cNb]==symbtab[j][0]&&
                  txt[cNb+1]==symbtab[j][1])
               {
                 cNb += 2;
                 return j;
               }
    default  : if(txt[cNb]!=' ') {
                 j=cNb;
                 while(txt[j]!=' '&&txt[j]!='%'&&txt[j]!='@'&&txt[j]!='\0')
                   j++;
                 //ifif(!symbtxt) delete symbtxt;
                 //symbtxtsymbtxt = new char[j-cNb+1];
                 // j-cNb+1>200 => ERROR
                 my_strncpy(symbtxt,txt+cNb,j-cNb);
                 symbtxt[j-cNb]='\0';
                 cNb=j;
                 return Text;
               }
               cNb++;
               return Undefined;
  }
}


/*********************************************************
* Write rutine for unparse functions.
* Input:
* txt - text to write
* output - should output occur to screen
* x,y - position to write
* view - current view
* Output:
* none
* Side Effects:
* x,y - are changed accordingly to width and size of
* text which is written
* view - update view accordingly to width and size of
* written text.
*********************************************************/
void write(const char *txt,const BOOL output,int &x,int &y,VIEW &view) {
  if(output) text_out(x,y,txt);
  // update current x-position
  x += sizeof_text(txt);
  // if current x-position is greater than the current
  // width of the view then adjust width of view.
  if(x>view.x+view.width) view.width=x-view.x;
  // if current y-position is greater than the current
  // height of the view then adjust height of view.
  if(y+get_char_height()>view.y+view.height)
    view.height=y+get_char_height()-view.y;
```

```
}

#define UPDATE_VIEW \
if(tree->view.x+tree->view.width>view.x+view.width) \
  view.width=tree->view.x+tree->view.width-view.x; \
if(tree->view.y+tree->view.height>view.y+view.height) \
  view.height=tree->view.y+tree->view.height-view.y


/*********************************************************
* This function unparses tree, by parsing several unparsin
* scheme and calling itself recursively.
* Input:
* us - current unparsing scheme
* tree -
* x,y -
* output -
* grpNb -
* margin -
* Output:
* none
* Side Effects:
*
*
*
*********************************************************/
void SyntaxTree::unparse_symbol_to_screen(const char *us,
  PSYNTAXTREE tree,int& x,int& y,BOOL output,
  unsigned int grpNb,unsigned int margin)
{
  unsigned int cPos = 0;  // current character position in unparsing scheme
  VIEW r;                 // temporary view used when calling recursively
  FONT hFontOld;          // holds old font when selecting a new one
  SYMBOL symbol;          // current unparsing symbol

  set_view(x,y,0,0); // initialise view for tree

  // Only for screen output
  if(output&&is_selected()) { set_bk_color(BLACK); set_text_color(WHITE); }

  if(is_elided()) {
    write(". . .",output,x,y,get_view());
  }
  else
  {
  symbol = nextsymbol(us,cPos);
  while(symbol!=EOL) {
    switch(symbol) {
      case BeginFont :
        // %(fontname%: texttowritewithfont %)
        nextsymbol(us,cPos); // read fontname into symbtxt
        nextsymbol(us,cPos); // read colon (does not affect symbtxt!)
        if(!my_strcmp(symbtxt,"keyword"))
          hFontOld=select_font(KEYWORD_FONT);
        if(!my_strcmp(symbtxt,"placeholder")) {
          hFontOld=select_font(NORMAL_FONT);
          write("<:",output,x,y,view);
          hFontOld=select_font(PLACEHOLDER_FONT);
        }
        if(!my_strcmp(symbtxt,"symbol")) {
          nextsymbol(us,cPos); // read symbol into symbtxt variable
          hFontOld=select_symbol(asciion,output,x,y,view,NULL);
        }
        if(!my_strcmp(symbtxt,"comment"))
          hFontOld=select_font(COMMENT_FONT);
        break;
      case EndFont:
```

```
        if(get_font()==PLACEHOLDER_FONT ) {
          select_font(hFontOld);
          write(":>",output,x,y,view);
        }
        else select_font(hFontOld);
        break;
      case Newline:
        x=margin;
        y+=get_char_height();                                              1110
        if(y>view.y+view.height) view.height=y-view.y;
        if(!output&&grpNb>0) cPos=my_strlen(us); // <- Testing
        break;
      case Indent:
        margin+=tabbing;
        break;
      case Unindent:
        margin-=tabbing;
        break;
      case OptNewline:                                                     1120
        r=view;
        unparse_symbol_to_screen(us+cPos,tree,x,y,FALSE,grpNb,margin);
        x=view.x; y=view.y;
//        view.x=r.x; view.y=r.y;
        if(x+view.width>get_window_width()) {
          x=margin;
          y+=get_char_height();
/*        if(y>view.y+view.height)
            view.height=y-view.y;
          else view.height=r.height;*/                                     1130
        }
//        else view.height=r.height;
//        view.width=r.width;
        break;
      case GrpNewline:
        if(grpNb>0) {
          x=margin;
          y+=get_char_height();
          if(y>view.y+view.height) view.height=y-view.y;
        }                                                                  1140
        break;
      case BeginInGroup:
        margin+=tabbing;
      case BeginGroup:
        r=view;
        unparse_symbol_to_screen(us+cPos,tree,x,y,FALSE,grpNb,margin);
        x=view.x; y=view.y;
        view.x=r.x; view.y=r.y;
        if(x+view.width>get_window_width()) grpNb++;
        view.width=r.width;                                                1150
        view.height=r.height;
        break;
      case EndInGroup:
        margin-=tabbing;
      case EndGroup:
        if(grpNb>0) grpNb--;
        break;
      case Child:
        if(tree) {
          tree->unparse_symbol_to_screen(                                  1160
            NodeInfoTable[tree->get_nodeid()].unp_scheme,
            (PSYNTAXTREE)tree->get_firstchild(),x,y,output,grpNb,margin);
          UPDATE_VIEW;
          if(tree->get_nextsibling()&&tree->get_parent()&&
             tree->get_parent()->is_listnode())
          {
            r=tree->get_view();
            ((PSYNTAXTREE)tree->get_parent())->unparse_symbol_to_screen(
```

```
                NodeInfoTable[tree->get_parent()->get_nodeid()].unp_sep_scheme,
                  NULL,x,y,output,grpNb,margin);                                    1170
              UPDATE_VIEW;
              tree->view=r;
              cPos--;  // Decrease character position so that eventually
                       // next sibling in list can be showed
            }
            tree=(PSYNTAXTREE) tree->get_nextsibling();
          }
          break;
        case Percent:
          write("%",output,x,y,view);                                               1180
          break;
        case Special:
          write(unparse_special(),output,x,y,view);
          break;
        default:
          my_strcpy(symbtxt," ");
        case Text:
          write(symbtxt,output,x,y,view);
          break;
      }                                                                             1190
      symbol=nextsymbol(us,cPos);
    }
    }
  if(output&&is_selected()) { set_bk_color(WHITE); set_text_color(BLACK); }
// if(output) draw_rect(get_view()); // temporary must be deleted !
}

/*********************************************************
* Write rutine for unparse functions.
* Input:                                                                        1200
* file - file to write to
* txt - text to write
* output - should output occur to screen
* x,y - position to write
* view - current view
* Output:
* BOOL - is TRUE if writing to file went ok
* Side Effects:
* x,y - are changed accordingly to width and size of
* text which is written                                                         1210
* view - update view accordingly to width and size of
* written text.
*********************************************************/
BOOL write_file(FILEHANDLE file,const char *txt,const BOOL output,int &x,int &y,VIEW &view) {
  if(output) {
    if(!writefile(file,txt,my_strlen(txt))) {
      error("Error writing file !","Writefile error");
      return FALSE;
    }
  }                                                                             1220
  // update current x-position
  x += my_strlen(txt);
  // if current x-position is greater than the current
  // width of the view then adjust width of view.
  if(x>view.x+view.width) view.width=x-view.x;
  // if current y-position is greater than the current
  // height of the view then adjust height of view.
  if(y+1>view.y+view.height)
    view.height=y+1-view.y;
  return TRUE;                                                                  1230
}

static int filecharwidth = 80; // width of file in characters

BOOL SyntaxTree::unparse_symbol_to_file(FILEHANDLE file,const char *us,PSYNTAXTREE tree,
```

```
                                        int &x,int &y,
                                        BOOL output,unsigned int grpNb,
                                        unsigned int margin)
{
  unsigned int cPos = 0;  // current character position in unparsing scheme                1240
  VIEW r;                 // temporary view used when calling recursively
  SYMBOL symbol;
  FONT hFontOld;
  int j;

  set_view(x,y,0,0); // initialise view for tree

  symbol = nextsymbol(us,cPos);
  while(symbol!=EOL) {
    switch(symbol) {                                                                      1250
      case BeginFont :
        // %(fontname%: texttowritewithfont %)
        nextsymbol(us,cPos); // read fontname into symbtxt
        nextsymbol(us,cPos); // read colon (does not affect symbtxt!)
        if(!my_strcmp(symbtxt,"placeholder")) {
          if(!write_file(file,"<:",output,x,y,view)) return FALSE;
          hFontOld=select_font(PLACEHOLDER_FONT);
        }
        if(!my_strcmp(symbtxt,"symbol")) {
          nextsymbol(us,cPos); // read symbol into symbtxt variable                       1260
          hFontOld=select_symbol(TRUE,output,x,y,view,file);
        }
        break;
      case EndFont:
        if(get_font()==PLACEHOLDER_FONT ) {
          select_font(hFontOld);
          if(!write_file(file,":>",output,x,y,view)) return FALSE;
        }
        else select_font(hFontOld);
        break;                                                                            1270
      case Newline:
        if(!write_file(file,"\n",output,x,y,view)) return FALSE;
        for(j=0;j<margin;j++)
          if(!write_file(file," ",output,x,y,view)) return FALSE;
        x=margin;
        y+=1;
        if(y>view.y+view.height) view.height=y-view.y;
        break;
      case Indent:
        margin+=filetab;                                                                  1280
        break;
      case Unindent:
        margin-=filetab;
        break;
      case OptNewline:
        r=view;
        if(!unparse_symbol_to_file(file,us+cPos,tree,x,y,FALSE,grpNb,margin))
          return FALSE;
        x=view.x; y=view.y;
        view.x=r.x; view.y=r.y;                                                           1290
        if(x+view.width>filecharwidth) {
          if(!write_file(file,"\n",output,x,y,view)) return FALSE;
          for(j=0;j<margin;j++)
            if(!write_file(file," ",output,x,y,view)) return FALSE;
          x=margin;
          y+=1;
          if(y>view.y+view.height)
            view.height=y-view.y;
          else view.height=r.height;
        }                                                                                 1300
        else view.height=r.height;
        view.width=r.width;
```

```
        break;
      case GrpNewline:
        if(grpNb>0) {
          if(!write_file(file,"\n",output,x,y,view)) return FALSE;
          for(j=0;j<margin;j++)
            if(!write_file(file," ",output,x,y,view)) return FALSE;
          x=margin;
          y+=1;                                                          1310
          if(y>view.y+view.height) view.height=y-view.y;
        }
        break;
      case BeginInGroup:
        margin+=filetab;
      case BeginGroup:
        r=view;
        if(!unparse_symbol_to_file(file,us+cPos,tree,x,y,FALSE,grpNb,margin))
          return FALSE;
        x=view.x; y=view.y;                                              1320
        view.x=r.x; view.y=r.y;
        if(x+view.width>filecharwidth) grpNb++;
        view.width=r.width;
        view.height=r.height;
        break;
      case EndInGroup:
        margin-=filetab;
      case EndGroup:
        if(grpNb>0) grpNb--;
        break;                                                           1330
      case Child:
        if(tree) {
          if(!tree->unparse_symbol_to_file(file,
            NodeInfoTable[tree->get_nodeid()].unp_scheme,
            (PSYNTAXTREE)tree->get_firstchild(),x,y,output,grpNb,margin))
            return FALSE;
          UPDATE_VIEW;
          if(tree->get_nextsibling()&&tree->get_parent()&&
             tree->get_parent()->is_listnode())
          {                                                              1340
            r=tree->get_view();
            if(!((PSYNTAXTREE)tree->get_parent())->unparse_symbol_to_file(file,
              NodeInfoTable[tree->get_parent()->get_nodeid()].unp_sep_scheme,
              NULL,x,y,output,grpNb,margin))
              return FALSE;
            UPDATE_VIEW;
            tree->view=r;
            cPos--;  // Decrease character position so that eventually
                     // next sibling in list can be showed
          }                                                              1350
          tree=(PSYNTAXTREE) tree->get_nextsibling();
        }
        break;
      case Percent:
        if(!write_file(file,"%",output,x,y,view)) return FALSE;
        break;
      case Special:
        if(!write_file(file,unparse_special(),output,x,y,view)) return FALSE;
        break;
      default:                                                           1360
        my_strcpy(symbtxt," ");
      case Text:
        if(!write_file(file,symbtxt,output,x,y,view)) return FALSE;
        break;
    }
    symbol=nextsymbol(us,cPos);
  }
  return TRUE;
}
```

```
                                                                    1370
/*********************************************************
* Select symbolfont and writes appropiate symbol.
* Input:
* ascii - if TRUE then write the ascii representation of
* the symbol instead of using special font
* Output:
* none
* Side Effects:
* Uses the global variable symbtxt (here supposed to
* contain symbol to write)                                          1380
*********************************************************/
FONT select_symbol(BOOL ascii,BOOL output,int &x,int &y,VIEW &view,FILEHANDLE file) {
  FONT hFontOld = get_font();

  if(ascii)
    if(file!=NULL)
      write_file(file,symbtxt,output,x,y,view);
    else write(symbtxt,output,x,y,view);
  else {
    unsigned int symbNb = 0;                                        1390

    while(symbNb<NUMSYMBOLS&&my_strcmp(SymbolTable[symbNb].ascii,symbtxt))
      symbNb++;
    if(symbNb<NUMSYMBOLS) { // if symbol found then ...
      hFontOld=select_font(SYMBOL_FONT);
      my_strcpy(symbtxt,SymbolTable[symbNb].symbol);
      write(symbtxt,output,x,y,view);
    }
  }
  return hFontOld;                                                  1400
}


#endif /* SYNTREE_CPP */
```

G.14 TEXTBUF.CPP

```
/*************************************************************************
*
* TEXTBUF.CPP
*
* Implementation of text buffer class
*
* Created 6 February, 1994, Michael Suodenjoki
*
*************************************************************************/
                                                                    10
#ifndef TEXTBUF_CPP
#define TEXTBUF_CPP

#include "TEXTBUF.H"
#include "RSLEDWIN.H"

// View origin in Y direction
extern int ViewOrgY;

/*********************************************************          20
* Constructor for Textbuffer. Initialises character
* positions and allocates memory for buffer.
* Input:
* none
* Output:
```

```
* none
* Side Effects:
* memory allocation
**********************************************************/
TextBuffer::TextBuffer() {                                          30
  charPos=0; charMaxPos=0; buffer=new char[MaxPos];
}

/**********************************************************
* Destructor for TextBuffer. Deallocates the buffer and
* hides/destroys the caret.
* Input:
* none
* Output:
* none                                                              40
* Side Effects:
* Deallocation of memory
**********************************************************/
TextBuffer::~TextBuffer() {
  delete buffer; HideCaret(hwnd); DestroyCaret();
}

/**********************************************************
* Return the buffer.
* Input:                                                            50
* none
* Output:
* the buffer
* Side Effects:
* none
**********************************************************/
char *TextBuffer::get_buffer(void) { return buffer; }

/**********************************************************
* Constructor for TextBuffer. Initialises the character             60
* positions, allocates buffer, initialises view, creates
* and shows the caret.
* Input:
* h - handle of window which owe textbufffer
* v - Initial view
* buf - Initial contents of buffer
* Output:
* none
* Side Effects:
* Allocation of memory, displaying of the caret.                    70
**********************************************************/
TextBuffer::TextBuffer(HWND h,VIEW v,char *buf) {
  charPos=0;
  charMaxPos=0;
  buffer=new char[MaxPos];
  buffer[0]='\0';
  hwnd=h; bufview=v;
// buffer=buf;
  erase(0);
  CreateCaret(hwnd,NULL,2,cyChar);                                  80
  SetCaretPos(bufview.x,bufview.y-ViewOrgY);
  ShowCaret(hwnd);
}

/**********************************************************
* Inserts a characeter in the text buffer.
* Input:
* ch - character to insert
* Output:
* none                                                              90
* Side Effects:
* text buffer repainted
```

```
*********************************************************/
void TextBuffer::insert_char(char ch) {
 for(int i=charMaxPos+1; i>charPos; i--) buffer[i]=buffer[i-1];
 charMaxPos++;
 buffer[charPos]=ch;
 if(charPos<MaxPos) charPos++;
 paint();
}                                                                100

/*********************************************************
* Deletes the previous character (Backspace).
* Input:
* none
* Output:
* none
* Side Effects:
* Repainting of the textbuffer
*********************************************************/      110
void TextBuffer::delete_prev_char(void) {
   charPos--;
   strcpy(buffer+charPos,buffer+charPos+1);
   charMaxPos--;
   paint();
   erase(charMaxPos);
}

/*********************************************************
* Deletes a character (Delete).                             120
* Input:
* none
* Output:
* none
* Side Effects:
* Repainting of the textbuffer
*********************************************************/
void TextBuffer::delete_char(void) {
   if(charPos<charMaxPos) {
      strcpy(buffer+charPos,buffer+charPos+1);               130
      charMaxPos--;
      paint();
      erase(charMaxPos);
   }
}

/*********************************************************
* Moves the character position left.
* Input:
* none                                                      140
* Output:
* none
* Side Effects:
* Repainting of the caret
*********************************************************/
void TextBuffer::left(void) {
   if(charPos>0) {
      charPos--;
      update_caret();
   }                                                        150
}

/*********************************************************
* Moves the character position right.
* Input:
* none
* Output:
* none
* Side Effects:
```

```
* Repainting of the caret
*************************************************************/
void TextBuffer::right(void) {
  if(charPos<charMaxPos) {
    charPos++;
    update_caret();
  }
}

/*************************************************************
* Paints the textbuffer at View.
* Input:
* none
* Output:
* none
* Side Effects:
* Screen updated
*************************************************************/
void TextBuffer::paint(void) {
  HDC hdc;
  SIZE size;
  RECT r;

  hdc=GetDC(hwnd);

  SelectObject(hdc,hfontNormal);

  GetTextExtentPoint(hdc,buffer,charPos,&size);
  if(size.cx>bufview.width) {
    SetRect(&r,bufview.x+bufview.width,bufview.y,widthClient,bufview.y+cyChar);
    ScrollWindow(hwnd,-bufview.width+size.cx,0,&r,NULL);
    bufview.width=size.cx;
  }

  HideCaret(hwnd);
  SetBkColor(hdc,BLACK); SetTextColor(hdc,WHITE);
  TextOut(hdc,bufview.x,bufview.y-ViewOrgY,buffer,charMaxPos);
  SetBkColor(hdc,WHITE); SetTextColor(hdc,BLACK);
  ShowCaret(hwnd);

  ReleaseDC(hwnd,hdc);

  update_caret();
}

/*************************************************************
* Update the caret position and show it on screen
* Input:
* none
* Output:
* none
* Side Effects:
* Screen updated
*************************************************************/
void TextBuffer::update_caret(void) {
  SIZE size;
  HDC hdc;

  hdc=GetDC(hwnd);
  SelectObject(hdc,hfontNormal);
  GetTextExtentPoint(hdc,buffer,charPos,&size);
  ReleaseDC(hwnd,hdc);

  HideCaret(hwnd);
  SetCaretPos(bufview.x+size.cx,bufview.y-ViewOrgY);
  ShowCaret(hwnd);
}
```

```
/*********************************************************
* Erase the area from Pos to view width.
* Input:                                                                    230
* Pos - character position in textbuffer to clear from.
* Output:
* none
* Side Effects:
* the area on the screen are cleared
*********************************************************/
void TextBuffer::erase(int Pos) {
  RECT r;
  HDC hdc;
  SIZE size;                                                                240

  hdc=GetDC(hwnd);
  SelectObject(hdc,hfontNormal);
  GetTextExtentPoint(hdc,buffer,Pos,&size);

  SetRect(&r,bufview.x+size.cx,bufview.y-ViewOrgY,bufview.x+bufview.width,bufview.y-ViewOrgY+bufview.height);

  SetBkColor(hdc,WHITE);
  ExtTextOut(hdc,bufview.x,bufview.y-ViewOrgY,ETO_OPAQUE,&r,NULL,NULL,NULL);
  ReleaseDC(hwnd,hdc);                                                      250
}


#endif /* TEXTBUF_CPP */
```

G.15 TRANSWND.CPP

```
/*************************************************************************
*
* TRANSWND.CPP
*
* Transformation Window Procedure Implementation
*
* Created 20 December, 1993, Michael Suodenjoki
*
*************************************************************************/
                                                                            10
#ifndef TRANSWND_CPP
#define TRANSWND_CPP

#include "RSLEDWIN.H"

// Prototypes for process functions
void TransWndProcessCREATE(HWND);
BOOL TransWndProcessCOMMAND(HWND,WPARAM,LPARAM);
void TransWndProcessPAINT(HWND);
                                                                            20
// Prototype for MakeIt function
extern PSYNTAXTREE MakeIt(const unsigned int);

// Variable to communicate from transform to edit window
GLOBAL BOOL justTransformed;

// Local variable for this module
static BOOL redraw = FALSE;

/*************************************************************************************                30
*
* Transformation Window Procedure
```

```
*
*****************************************************************************/

long FAR PASCAL TransWndProc(HWND hwnd, unsigned message, WPARAM wParam, LPARAM lParam)
{
  switch(message) {
    case WM_CREATE:
      TransWndProcessCREATE(hwnd);                                              40
      return 0;
    case WM_COMMAND:
      if(TransWndProcessCOMMAND(hwnd,wParam,lParam)) return 0;
      break;
    case WM_SIZE:
      redraw=TRUE;
      return 0;
    case WM_PAINT:
      TransWndProcessPAINT(hwnd);
      return 0;                                                                 50
    case WM_DESTROY:
      PostQuitMessage(0);
      return 0;
  }
  return DefWindowProc(hwnd,message,wParam,lParam);
}

/*********************************************************
* Process WM_CREATE messages
* Input:                                                                        60
* hwnd - Window handle
* Output:
* none
* Side Effects:
*
*********************************************************/
void TransWndProcessCREATE(HWND hwnd) {
  for(int i=1;i<=NUMTRANS;i++)
    hwndButton[i-1]=CreateWindow("button",TransformationTable[i-1].text,
                    WS_CHILD | BS_PUSHBUTTON,                                   70
                    0,0,0,0,
                    hwnd,(HMENU)i,GetWindowNumber(hwnd),NULL);
}

/*********************************************************
* Process WM_COMMAND messages
* Input:
* hwnd - Window handle
* wParam - Command
* lParam - Identifies Button                                                    80
* Output:
* BOOL - has command been processed
* Side Effects:
*
*********************************************************/
BOOL TransWndProcessCOMMAND(HWND hwnd,WPARAM wParam,LPARAM lParam) {
  RECT r;
  VIEW v;

  if(HIWORD(lParam)==0) {                                                       90
    for(int i=1; i<=NUMTRANS; i++)
      if(wParam==i) {
        SelectedNode->delete_subtree(SelectedNode);
        SelectedNode->insert_tree_last(MakeIt(i));

        justTransformed=TRUE;
        SendMessage(hwndEditChild,WM_KEYDOWN,VK_F3,(LPARAM) 0);
        justTransformed=FALSE;
```

```
        SetFocus(wndOldFocus);                                          100
        return TRUE;
      }
  }
  return FALSE;
}

/*********************************************************
* Process WM_PAINT messages
* Input:
* hwnd - Window handle                                                  110
* Output:
* none
* Side Effects:
*
*********************************************************/
void TransWndProcessPAINT(HWND hwnd) {
  HDC                hdc;
  PAINTSTRUCT        ps;
  TEXTMETRIC         tm;
  RECT               rect;                                              120
  static unsigned int sbn = 1;
  static NODEID      oldnodeid = 0;
  NODEID             nodeid;

  hdc = BeginPaint(hwnd,&ps);

  GetClientRect(hwnd,&rect);

  GetTextMetrics(hdc,&tm);
  int widthChar=tm.tmAveCharWidth;                                      130
  int heightChar=tm.tmHeight+tm.tmExternalLeading;

  int x = 2;
  int y = 2;
  if(SelectedNode) nodeid = SelectedNode->get_nodeid(); else nodeid=NUMNODEIDS;
  if(oldnodeid!=nodeid||redraw) {
    SendMessage(hwnd,WM_ERASEBKGND,(WPARAM) hdc,(LPARAM) 0);

    SIZE dwButtonSize;
                                                                        140
    // hide old buttons;
    int i = sbn;
    while(i<=NUMTRANS&&TransformationTable[i-1].nodeid==oldnodeid) {
      ShowWindow(hwndButton[i-1],SW_HIDE);
      i++;
    }

    // find start button number
    if(!redraw) {
      sbn = 1;                                                          150
      while(sbn<=NUMTRANS&&TransformationTable[sbn-1].nodeid<>nodeid) sbn++;
    }

    // draw new button
    i = sbn;
    while(i<=NUMTRANS&&TransformationTable[i-1].nodeid==nodeid) {
      GetTextExtentPoint(hdc,
        TransformationTable[i-1].text,
        lstrlen(TransformationTable[i-1].text),&dwButtonSize);
      if(x+8+dwButtonSize.cx>rect.right) {                              160
        x = 2; y=y+8+heightChar;
      };
      MoveWindow(hwndButton[i-1],x,y,8+dwButtonSize.cx,heightChar+4,FALSE);
      ShowWindow(hwndButton[i-1],SW_SHOWNORMAL);
      x=x+4+8+dwButtonSize.cx;
      i++;
```

```
    }

    oldnodeid=nodeid;
    redraw=FALSE;                                                    170
  }

  EndPaint(hwnd,&ps);
}

#endif /* TRANSWND_CPP */
```

Appendix H

Flex++ and Bison++ input files

H.1 SCANNER.L

```
%name SCANNER

%define USE_CLASS
%define MEMBERS public : int theLine;
%define CONSTRUCTOR_INIT : theLine(1)

%define INPUT_PURE
%define FATAL_ERROR_PURE

%header{                                                                    10
#include "PARSER.H"
%}

%{

/*************************************************************************
*
* SCANNER.L
*
* FLEX++ input file for RSL scanner                                         20
*
* 1993, 1994 Michael Suodenjoki
*
*************************************************************************/

#include "SYNTREE.H"
#include "STDLIB.H"
#include "IO.H"
#include "MYSTRING.H"
#include "NODEIDS.H"                                                        30

int id_or_keyword(char *);
int placeholder(char *);
int startsymbol(char *);

// Boolean value determine if scanner current is in a comment
BOOL incomment = FALSE;

%}
                                                                            40
ldup            {letter}|{digit}|{underline}|{prime}
letter          {ascii_letter}
```

```
text_character   {character}|{prime}
char_character   {character}|{quote}
character        {ascii_letter}|{digit}|{graphic}|{escape}
digit            [0-9]
ascii_letter     [a-zA-Z]
underline        \x5f
prime            \x27
quote            \x22                                                                    50
backslash        \x5c
graphic          " "|"!"|"#"|"$"|"%"|"&"|"("|")"|"*"|"+"|","|"-"|"."|"/"|":"|";"|"<"|"="|">"|"?"|"@"|"_"|"`"|"{"|"|"|"}"|"~"
escape           \x5c([rntabfv?'"]|{quote}|"x"{hex_constant})
oct_constant     {oct_digit}{1,3}
hex_constant     {hex_digit}+
oct_digit        [0-7]
hex_digit        {digit}|[a-fA-F]
comment_char     {ascii_letter}|{digit}|{prime}|{quote}|{backslash}

%x comment                                                                               60
%x error
%%

        if(incomment) BEGIN(comment); else BEGIN(INITIAL);

"/*"                            { incomment=TRUE; return PARSER::COMMENTSTART; }
"*/"                            { incomment=FALSE; return PARSER::COMMENTEND; }
<comment>"*/"                   { incomment=FALSE; return PARSER::COMMENTEND; }

<comment>{comment_char}+        return PARSER::COMMENT_TEXT;                             70
<comment>\n                     return PARSER::COMMENT_NEWLINE;

"{!"                            BEGIN(error);
<error>[^!]*                    /* Eat anything that's not a '!' */
<error>"!"+[^!}]*               /* Eat up '!'s not followed by '}' */
<error>"!"+"}"                  BEGIN(INITIAL);

{letter}{ldup}*                 return id_or_keyword((char *)yytext);

"<:"{letter}{ldup}*":>"         { int i=placeholder((char *)yytext);                     80
                                  if(i==-1) yyterminate();
                                  return i;
                                };

{digit}+                        return PARSER::INT_LITERAL;
{digit}+"."{digit}+             return PARSER::REAL_LITERAL;
{quote}{text_character}*{quote} return PARSER::TEXT_LITERAL;
"'"{char_character}"'"          return PARSER::CHAR_LITERAL;

"@"{digit}+                     return startsymbol((char *)yytext);                      90

"*"                             return PARSER::AST;
"|"                             return PARSER::BAR;
"`"                             return PARSER::BQUOT;
":"                             return PARSER::COLON;
"::"                            return PARSER::COLONCOLON;
":="                            return PARSER::COLONEQ;
","                             return PARSER::COMMA;
"^"                             return PARSER::CONCAT;
"!!"                            return PARSER::DAGGER;                                   100
"=>"                            return PARSER::DBLRIGHTARROW;
"|=|"                           return PARSER::DETCHOICE;
"."                             return PARSER::DOT;
". ."                           return PARSER::DOTDOT;
"="                             return PARSER::EQ;
"=="                            return PARSER::EQEQ;
"!"                             return PARSER::EXCLAM;
">="                            return PARSER::GEQ;
">"                             return PARSER::GT;
```

```
"#"                          return PARSER::HASH;
"-inflist"                   return PARSER::INFLIST;
"-infset"                    return PARSER::INFSET;
"-"\x5c                      return PARSER::LAMBDA;
"<."                         return PARSER::LANGLE;
"[|"                         return PARSER::LARRBRACKET;
"{"                          return PARSER::LBRACE;
"["                          return PARSER::LBRACKET;
"<->"                        return PARSER::LEFTRIGHTARROW;
"<="                         return PARSER::LEQ;
"-list"                      return PARSER::LIST;
"("                          return PARSER::LPAR;
"<"                          return PARSER::LT;
"{|"                         return PARSER::LTYPEBRACE;
"+>"                         return PARSER::MAPSTO;
"-m->"                       return PARSER::MARROW;
"-"                          return PARSER::MINUS;
"|^|"                        return PARSER::NONDETCHOICE;
"~="|"<>"                    return PARSER::NOTEQ;
"~isin"                      return PARSER::NOTISIN;
"| |"                        return PARSER::PARL;
"-~->"                       return PARSER::PARRIGHTARROW;
"+"                          return PARSER::PLUS;
" "                          return PARSER::PRIM;
"?"                          return PARSER::QUERY;
".>"                         return PARSER::RANGLE;
"|]"                         return PARSER::RARRBRACKET;
"}"                          return PARSER::RBRACE;
"]"                          return PARSER::RBRACKET;
":-"                         return PARSER::RDOT;
"->"                         return PARSER::RIGHTARROW;
")"                          return PARSER::RPAR;
"|}"                         return PARSER::RTYPEBRACE;
";"                          return PARSER::SEMI;
"-set"                       return PARSER::SET;
\x5c                         return PARSER::SETMINUS;
"~"                          return PARSER::SIM;
"/"                          return PARSER::SLASH;
"<<"                         return PARSER::SUBSET;
"<<="                        return PARSER::SUBSETEQ;
">>"                         return PARSER::SUPSET;
">>="                        return PARSER::SUPSETEQ;
"++"                         return PARSER::TIE;
"><"                         return PARSER::TIMES;
"_"                          return PARSER::UNDERLINE;
"**"                         return PARSER::UPARROW;
\x5c"/"                      return PARSER::VEE;
"/"\x5c                      return PARSER::WEDGE;

[ \t]                        ;
\n                           theLine++;
.                            return PARSER::FEOF;
<<EOF>>                      return PARSER::FEOF;

%%

typedef struct {
  char *name;
  int val;
} WORDTOVAL;

// Table of placeholders.
// It is important that this table is ordered wrt. string comparison
// Are used in a binary search
WORDTOVAL PTable[] = {
  { "<:ACCESS:>",          PARSER::PH_ACCESS },
  { "<:ACCESS_MODE:>",     PARSER::PH_ACCESS_MODE },
```

```
  { "<:BASIC_EXPR:>",    PARSER::PH_BASIC_EXPR },
  { "<:BINDING:>",       PARSER::PH_BINDING },
  { "<:BOOL_LITERAL:>",  PARSER::PH_BOOL_LITERAL },
  { "<:CHANNEL_DEF:>",   PARSER::PH_CHANNEL_DEF },                          180
  { "<:CLASS_EXPR:>",    PARSER::PH_CLASS_EXPR },
  { "<:COMMENT:>",       PARSER::PH_COMMENT },
  { "<:COMMENT_TOKEN:>",PARSER::PH_COMMENT_TOKEN },
  { "<:CONNECTIVE:>",    PARSER::PH_CONNECTIVE },
  { "<:CONSTRUCTOR:>",   PARSER::PH_CONSTRUCTOR },
  { "<:DECL:>",          PARSER::PH_DECL },
  { "<:DEFINED_ITEM:>",  PARSER::PH_DEFINED_ITEM },
  { "<:FORMAL_FUNCTION_APPLICATION:>",PARSER::PH_FORMAL_FUNCTION_APPLICATION },
  { "<:FUNCTION_ARROW:>",PARSER::PH_FUNCTION_ARROW },
  { "<:ID:>",            PARSER::PH_ID },                                   190
  { "<:ID_OR_OP:>",      PARSER::PH_ID_OR_OP },
  { "<:INFIX_COMBINATOR:>",PARSER::PH_INFIX_COMBINATOR },
  { "<:INFIX_CONNECTIVE:>",PARSER::PH_INFIX_CONNECTIVE },
  { "<:INFIX_EXPR:>",    PARSER::PH_INFIX_EXPR },
  { "<:INFIX_OP:>",      PARSER::PH_INFIX_OP },
  { "<:INNER_PATTERN:>",PARSER::PH_INNER_PATTERN },
  { "<:LAMBDA_PARAMETER:>",PARSER::PH_LAMBDA_PARAMETER },
  { "<:LET_BINDING:>",   PARSER::PH_LET_BINDING },
  { "<:LET_DEF:>",       PARSER::PH_LET_DEF },
  { "<:LIST_EXPR:>",     PARSER::PH_LIST_EXPR },                            200
  { "<:LIST_PATTERN:>",  PARSER::PH_LIST_PATTERN },
  { "<:LIST_TYPE_EXPR:>",PARSER::PH_LIST_TYPE_EXPR },
  { "<:MAP_EXPR:>",      PARSER::PH_MAP_EXPR },
  { "<:MODULE_DECL:>",   PARSER::PH_MODULE_DECL },
  { "<:NAME:>",          PARSER::PH_NAME },
  { "<:NAME_OR_WILDCARD:>",PARSER::PH_NAME_OR_WILDCARD },
  { "<:OBJECT_EXPR:>",   PARSER::PH_OBJECT_EXPR },
  { "<:OP:>",            PARSER::PH_OP },
  { "<:PATTERN:>",       PARSER::PH_PATTERN },
  { "<:PREFIX_EXPR:>",   PARSER::PH_PREFIX_EXPR },                          210
  { "<:PREFIX_OP:>",     PARSER::PH_PREFIX_OP },
  { "<:QUANTIFIER:>",    PARSER::PH_QUANTIFIER },
  { "<:SET_EXPR:>",      PARSER::PH_SET_EXPR },
  { "<:SET_TYPE_EXPR:>",PARSER::PH_SET_TYPE_EXPR },
  { "<:STRUCTURED_EXPR:>",PARSER::PH_STRUCTURED_EXPR },
  { "<:TYPE_DEF:>",      PARSER::PH_TYPE_DEF },
  { "<:TYPE_EXPR:>",     PARSER::PH_TYPE_EXPR },
  { "<:TYPE_LITERAL:>",  PARSER::PH_TYPE_LITERAL },
  { "<:TYPING:>",        PARSER::PH_TYPING },
  { "<:VALUE_DEF:>",     PARSER::PH_VALUE_DEF },                            220
  { "<:VALUE_EXPR:>",    PARSER::PH_VALUE_EXPR },
  { "<:VALUE_LITERAL:>",PARSER::PH_VALUE_LITERAL },
  { "<:VARIABLE_DEF:>",  PARSER::PH_VARIABLE_DEF },
  { "<:VARIANT:>",       PARSER::PH_VARIANT }};

// Table of keywords.
// It is important that this table is ordered wrt. string comparison
// Are used in a binary search
WORDTOVAL KTable[] = {
  { "Bool",      PARSER::KW_TBOOL },                                        230
  { "Char",      PARSER::KW_TCHAR },
  { "Int",       PARSER::KW_TINT },
  { "Nat",       PARSER::KW_TNAT },
  { "Real",      PARSER::KW_TREAL },
  { "Text",      PARSER::KW_TTEXT },
  { "Unit",      PARSER::KW_TUNIT },
  { "abs",       PARSER::KW_ABS },
  { "all",       PARSER::ALL },
  { "always",    PARSER::ALWAYS },
  { "any",       PARSER::KW_ANY },                                          240
  { "as",        PARSER::KW_AS },
  { "axiom",     PARSER::KW_AXIOM },
  { "card",      PARSER::KW_CARD },
```

```
  { "case",       PARSER::KW_CASE },
  { "channel",    PARSER::KW_CHANNEL },
  { "chaos",      PARSER::KW_CHAOS },
  { "class",      PARSER::KW_CLASS },
  { "do",         PARSER::KW_DO },
  { "dom",        PARSER::KW_DOM },
  { "elems",      PARSER::KW_ELEMS },                                    250
  { "else",       PARSER::KW_ELSE },
  { "elsif",      PARSER::KW_ELSIF },
  { "end",        PARSER::KW_END },
  { "exists",     PARSER::EXISTS },
  { "extend",     PARSER::KW_EXTEND },
  { "false",      PARSER::KW_FALSE },
  { "for",        PARSER::KW_FOR },
  { "forall",     PARSER::KW_FORALL },
  { "hd",         PARSER::KW_HD },
  { "hide",       PARSER::KW_HIDE },                                     260
  { "if",         PARSER::KW_IF },
  { "in",         PARSER::KW_IN },
  { "inds",       PARSER::KW_INDS },
  { "initialise",PARSER::KW_INITIALISE },
  { "int",        PARSER::KW_INT },
  { "inter",      PARSER::INTER },
  { "is",         PARSER::IS },
  { "isin",       PARSER::ISIN },
  { "len",        PARSER::KW_LEN },
  { "let",        PARSER::KW_LET },                                      270
  { "local",      PARSER::KW_LOCAL },
  { "object",     PARSER::KW_OBJECT },
  { "of",         PARSER::KW_OF },
  { "out",        PARSER::KW_OUT },
  { "post",       PARSER::KW_POST },
  { "pre",        PARSER::KW_PRE },
  { "read",       PARSER::KW_READ },
  { "real",       PARSER::KW_REAL },
  { "rng",        PARSER::KW_RNG },
  { "scheme",     PARSER::KW_SCHEME },                                   280
  { "skip",       PARSER::KW_SKIP },
  { "stop",       PARSER::KW_STOP },
  { "swap",       PARSER::KW_SWAP },
  { "then",       PARSER::KW_THEN },
  { "tl",         PARSER::KW_TL },
  { "true",       PARSER::KW_TRUE },
  { "type",       PARSER::KW_TYPE },
  { "union",      PARSER::UNION },
  { "until",      PARSER::KW_UNTIL },
  { "use",        PARSER::KW_USE },                                      290
  { "value",      PARSER::KW_VALUE },
  { "variable",   PARSER::KW_VARIABLE },
  { "while",      PARSER::KW_WHILE },
  { "with",       PARSER::KW_WITH },
  { "write",      PARSER::KW_WRITE }};

/*******************************************************
* This cmp_case function compare to strings (casesensitive)
* and return TRUE if they are equal.
* Input:                                                                300
* a,b - the two string to compare
* Output:
* the same as strcmpi
* Side Effects:
* none
*******************************************************/
int _cdecl cmp_case(const void *a, const void *b) {
  return my_strcmp(((WORDTOVAL*) a)->name,((WORDTOVAL*) b)->name);
}
                                                                        310
```

```
/*********************************************************
* This cmp_not_case function compare to strings (not casesensitive)
* and return TRUE if they are equal.
* Input:
* a,b - the two string to compare
* Output:
* the same as strcmpi
* Side Effects:
* none
*********************************************************/
int _cdecl cmp_not_case(const void *a, const void *b) {
  return my_strcmpi(((WORDTOVAL*) a)->name,((WORDTOVAL*) b)->name);
}


/*********************************************************
* The function examines if input is either an identifier
* or a keyword in the language.
* Input:
* lex - the string containing either an identifier or
* keyword
* Output:
* the appropiate parser token
* Side Effects:
* none
*********************************************************/
int id_or_keyword(char* lex) {
  WORDTOVAL *p;
  WORDTOVAL dummy;

  dummy.name=lex;

  // Search for keywords
  p = bsearch(&dummy, KTable, sizeof(KTable)/sizeof(WORDTOVAL),
       sizeof(WORDTOVAL), cmp_case);

  return (p ? p->val : PARSER::IDENTIFIER);
}

/*********************************************************
* The function examines if input is a placeholder string
* Input:
* lex - the string containing the placeholder string
* Output:
* the appropiate parser token
* Side Effects:
* none
*********************************************************/
int placeholder(char *lex) {
  WORDTOVAL *p;
  WORDTOVAL dummy;

  dummy.name=lex;

  // Search for placeholder
  p = bsearch(&dummy, PTable, sizeof(PTable)/sizeof(WORDTOVAL),
       sizeof(WORDTOVAL), cmp_not_case);

  return (p ? p->val : -1);
}

/*********************************************************
* The function returns the appropiate parser token for
* a startsymbol string.
* Input:
* lex - the string containing start symbol
* Output:
```

```
* the appropiate parser token
*****************************************************/
int startsymbol(char *lex) {                                                    380

  switch(atoi(lex+1)) {
    case NID_SPECIFICATION   : return PARSER::SPECIFICATION_START;
    case NID_MODULEDECL      : return PARSER::MODULE_DECL_START;
    case NID_DECL            : return PARSER::DECL_START;
    case NID_TYPEDEF         : return PARSER::TYPE_DEF_START;
    case NID_VARIANT         : return PARSER::VARIANT_START;
    case NID_CONSTRUCTOR     : return PARSER::CONSTRUCTOR_START;
    case NID_NAMEORWILDCARD  : return PARSER::NAME_OR_WILDCARD_START;
    case NID_VALUEDEF        : return PARSER::VALUE_DEF_START;                  390
    case NID_FORMALFUNCTIONAPPLICATION : return PARSER::FORMAL_FUNCTION_APPLICATION_START;
    case NID_VARIABLEDEF     : return PARSER::VARIABLE_DEF_START;
    case NID_CHANNELDEF      : return PARSER::CHANNEL_DEF_START;
    case NID_CLASSEXPR       : return PARSER::CLASS_EXPR_START;
    case NID_DEFINEDITEM     : return PARSER::DEFINED_ITEM_START;
    case NID_OBJECTEXPR      : return PARSER::OBJECT_EXPR_START;
    case NID_TYPEEXPR        : return PARSER::TYPE_EXPR_START;
    case NID_TYPELITERAL     : return PARSER::TYPE_LITERAL_START;
    case NID_SETTYPEEXPR     : return PARSER::SET_TYPE_EXPR_START;
    case NID_LISTTYPEEXPR    : return PARSER::LIST_TYPE_EXPR_START;             400
    case NID_FUNCTIONARROW   : return PARSER::FUNCTION_ARROW_START;
    case NID_ACCESSMODE      : return PARSER::ACCESS_MODE_START;
    case NID_ACCESS          : return PARSER::ACCESS_START;
    case NID_AXIOMDEF        : return PARSER::AXIOM_DEF_START;
    case NID_VALUEEXPR       : return PARSER::VALUE_EXPR_START;
    case NID_VALUELITERAL    : return PARSER::VALUE_LITERAL_START;
    case NID_BOOLLITERAL     : return PARSER::BOOL_LITERAL_START;
    case NID_BASICEXPR       : return PARSER::BASIC_EXPR_START;
    case NID_SETEXPR         : return PARSER::SET_EXPR_START;
    case NID_LISTEXPR        : return PARSER::LIST_EXPR_START;                  410
    case NID_MAPEXPR         : return PARSER::MAP_EXPR_START;
    case NID_LAMBDAPARAMETER : return PARSER::LAMBDA_PARAMETER_START;
    case NID_QUANTIFIER      : return PARSER::QUANTIFIER_START;
    case NID_INFIXEXPR       : return PARSER::INFIX_EXPR_START;
    case NID_PREFIXEXPR      : return PARSER::PREFIX_EXPR_START;
    case NID_STRUCTUREDEXPR  : return PARSER::STRUCTURED_EXPR_START;
    case NID_LETDEF          : return PARSER::LET_DEF_START;
    case NID_LETBINDING      : return PARSER::LET_BINDING_START;
    case NID_BINDING         : return PARSER::BINDING_START;
    case NID_TYPING          : return PARSER::TYPING_START;                     420
    case NID_PATTERN         : return PARSER::PATTERN_START;
    case NID_LISTPATTERN     : return PARSER::LIST_PATTERN_START;
    case NID_INNERPATTERN    : return PARSER::INNER_PATTERN_START;
    case NID_NAME            : return PARSER::NAME_START;
    case NID_IDOROP          : return PARSER::ID_OR_OP_START;
    case NID_OP              : return PARSER::OP_START;
    case NID_INFIXOP         : return PARSER::INFIX_OP_START;
    case NID_PREFIXOP        : return PARSER::PREFIX_OP_START;
    case NID_CONNECTIVE      : return PARSER::CONNECTIVE_START;
    case NID_INFIXCONNECTIVE : return PARSER::INFIX_CONNECTIVE_START;          430
    case NID_INFIXCOMBINATOR : return PARSER::INFIX_COMBINATOR_START;

    case NID_ID              : return PARSER::ID_START;
    case NID_COMMENT         : return PARSER::COMMENT_START;
  }
  return 0;
}
```

H.2 PARSER.Y

```
%name PARSER

%define LSP_NEEDED

%define STYPE PSYNTAXTREE

%define LEX_BODY =0

%define ERROR_BODY =0
                                                                        10
%{

/************************************************************************
*
* PARSER.Y
*
* Yacc input file for RSL language
*
* Created 15 January, 1994, Michael Suodenjoki
*                                                                       20
* Extended 20 January, 1994, Michael Suodenjoki
* Further constructs allowed, all keywords and placeholders defined
* Extended 21 January, 1994, Michael Suodenjoki
* Nearly the complete grammer. Only missing actions for lists
*
************************************************************************/

#include "SYNTREE.H"
#include "RSL.H"
                                                                        30
extern PSYNTAXTREE EditTree;


PSYNTAXTREE build(PSYNTAXTREE , PSYNTAXTREE = NULL, PSYNTAXTREE = NULL,
                  PSYNTAXTREE = NULL, PSYNTAXTREE = NULL, PSYNTAXTREE = NULL);
%}

%token SPECIFICATION_START
%token MODULE_DECL_START DECL_START TYPE_DEF_START VARIANT_START
%token CONSTRUCTOR_START NAME_OR_WILDCARD_START                         40
%token VALUE_DEF_START FORMAL_FUNCTION_APPLICATION_START
%token VARIABLE_DEF_START CHANNEL_DEF_START CLASS_EXPR_START
%token DEFINED_ITEM_START OBJECT_EXPR_START TYPE_EXPR_START
%token TYPE_LITERAL_START SET_TYPE_EXPR_START LIST_TYPE_EXPR_START
%token FUNCTION_ARROW_START ACCESS_MODE_START ACCESS_START
%token AXIOM_DEF_START VALUE_EXPR_START VALUE_LITERAL_START
%token BOOL_LITERAL_START BASIC_EXPR_START SET_EXPR_START
%token LIST_EXPR_START MAP_EXPR_START LAMBDA_PARAMETER_START
%token QUANTIFIER_START INFIX_EXPR_START PREFIX_EXPR_START
%token STRUCTURED_EXPR_START LET_DEF_START LET_BINDING_START            50
%token BINDING_START TYPING_START PATTERN_START LIST_PATTERN_START
%token INNER_PATTERN_START NAME_START ID_OR_OP_START OP_START
%token INFIX_OP_START PREFIX_OP_START CONNECTIVE_START
%token INFIX_CONNECTIVE_START INFIX_COMBINATOR_START
%token ID_START COMMENT_START

%token PH_MODULE_DECL PH_DECL PH_TYPE_DEF PH_VARIANT PH_CONSTRUCTOR
%token PH_NAME_OR_WILDCARD PH_VALUE_DEF
%token PH_FORMAL_FUNCTION_APPLICATION PH_VARIABLE_DEF PH_CHANNEL_DEF
%token PH_CLASS_EXPR PH_DEFINED_ITEM PH_OBJECT_EXPR PH_TYPE_EXPR        60
%token PH_TYPE_LITERAL PH_SET_TYPE_EXPR PH_LIST_TYPE_EXPR
%token PH_FUNCTION_ARROW PH_ACCESS_MODE PH_ACCESS
%token PH_VALUE_EXPR PH_VALUE_LITERAL PH_BOOL_LITERAL PH_BASIC_EXPR
%token PH_SET_EXPR PH_LIST_EXPR PH_MAP_EXPR PH_LAMBDA_PARAMETER
%token PH_QUANTIFIER PH_INFIX_EXPR PH_PREFIX_EXPR PH_STRUCTURED_EXPR
```

```
%token PH_LET_DEF PH_LET_BINDING PH_BINDING PH_TYPING PH_PATTERN
%token PH_LIST_PATTERN PH_INNER_PATTERN PH_NAME PH_ID_OR_OP PH_OP
%token PH_INFIX_OP PH_PREFIX_OP PH_CONNECTIVE PH_INFIX_CONNECTIVE
%token PH_INFIX_COMBINATOR PH_ID PH_COMMENT PH_COMMENT_TOKEN
                                                                    70
%token KW_TBOOL KW_TCHAR KW_TINT KW_TNAT KW_TREAL KW_TTEXT KW_TUNIT
%token KW_ABS KW_ANY KW_AS KW_AXIOM KW_CARD KW_CASE KW_CHANNEL
%token KW_CHAOS KW_CLASS KW_DO KW_DOM KW_ELEMS KW_ELSE KW_ELSIF
%token KW_END KW_EXTEND KW_FALSE KW_FOR KW_FORALL KW_HD KW_HIDE KW_IF
%token KW_IN KW_INDS KW_INITIALISE KW_INT KW_LEN KW_LET KW_LOCAL
%token KW_OBJECT KW_OF KW_OUT KW_POST KW_PRE KW_READ KW_REAL KW_RNG
%token KW_SCHEME KW_SKIP KW_STOP KW_SWAP KW_THEN KW_TL KW_TRUE
%token KW_TYPE KW_UNTIL KW_USE KW_VALUE KW_VARIABLE KW_WHILE
%token KW_WITH KW_WRITE
                                                                    80
%token ALL ALWAYS AST BAR BQUOT COLON COLONCOLON COLONEQ COMMA CONCAT
%token DAGGER DBLRIGHTARROW DETCHOICE DOT DOTDOT EQ EQEQ EXCLAM
%token EXISTS GEQ GT HASH INFLIST INFSET INTER IS ISIN LAMBDA LANGLE
%token LARRBRACKET LBRACE LBRACKET LEFTRIGHTARROW LEQ LIST LPAR LT
%token LTYPEBRACE MAPSTO MARROW MINUS NONDETCHOICE NOTEQ NOTISIN PARL
%token PARRIGHTARROW PLUS PRIM QUERY RANGLE RARRBRACKET RBRACE RBRACKET
%token RDOT RIGHTARROW RPAR RTYPEBRACE SEMI SET SETMINUS SIM SLASH
%token SUBSET SUBSETEQ SUPSET SUPSETEQ TIE TIMES UNDERLINE UNION
%token UPARROW VEE WEDGE
                                                                    90
%token IDENTIFIER INT_LITERAL REAL_LITERAL TEXT_LITERAL CHAR_LITERAL

%token COMMENTSTART COMMENTEND COMMENT_SPACE COMMENT_NEWLINE COMMENT_TEXT

%token FEOF

%%

start :
  ID_START                  id                  FEOF { EditTree=$2; YYACCEPT; } |                    100
  SPECIFICATION_START       specification       FEOF { EditTree=$2; YYACCEPT; } |
  MODULE_DECL_START         module_decl         FEOF { EditTree=$2; YYACCEPT; } |
  DECL_START                decl                FEOF { EditTree=$2; YYACCEPT; } |
  TYPE_DEF_START            type_def            FEOF { EditTree=$2; YYACCEPT; } |
  VARIANT_START             variant             FEOF { EditTree=$2; YYACCEPT; } |
  CONSTRUCTOR_START         constructor         FEOF { EditTree=$2; YYACCEPT; } |
  NAME_OR_WILDCARD_START    name_or_wildcard    FEOF { EditTree=$2; YYACCEPT; } |
  VALUE_DEF_START           value_def           FEOF { EditTree=$2; YYACCEPT; } |
  FORMAL_FUNCTION_APPLICATION_START formal_function_application FEOF { EditTree=$2; YYACCEPT; } |
  VARIABLE_DEF_START        variable_def        FEOF { EditTree=$2; YYACCEPT; } |                    110
  CHANNEL_DEF_START         channel_def         FEOF { EditTree=$2; YYACCEPT; } |
  CLASS_EXPR_START          class_expr          FEOF { EditTree=$2; YYACCEPT; } |
  DEFINED_ITEM_START        defined_item        FEOF { EditTree=$2; YYACCEPT; } |
  OBJECT_EXPR_START         object_expr         FEOF { EditTree=$2; YYACCEPT; } |
  TYPE_EXPR_START           type_expr           FEOF { EditTree=$2; YYACCEPT; } |
  TYPE_LITERAL_START        type_literal        FEOF { EditTree=$2; YYACCEPT; } |
  SET_TYPE_EXPR_START       set_type_expr       FEOF { EditTree=$2; YYACCEPT; } |
  LIST_TYPE_EXPR_START      list_type_expr      FEOF { EditTree=$2; YYACCEPT; } |
  FUNCTION_ARROW_START      function_arrow      FEOF { EditTree=$2; YYACCEPT; } |
  ACCESS_MODE_START         access_mode         FEOF { EditTree=$2; YYACCEPT; } |                    120
  ACCESS_START              access              FEOF { EditTree=$2; YYACCEPT; } |
  AXIOM_DEF_START           axiom_def           FEOF { EditTree=$2; YYACCEPT; } |
  VALUE_EXPR_START          value_expr          FEOF { EditTree=$2; YYACCEPT; } |
  VALUE_LITERAL_START       value_literal       FEOF { EditTree=$2; YYACCEPT; } |
  BOOL_LITERAL_START        bool_literal        FEOF { EditTree=$2; YYACCEPT; } |
  BASIC_EXPR_START          basic_expr          FEOF { EditTree=$2; YYACCEPT; } |
  SET_EXPR_START            set_expr            FEOF { EditTree=$2; YYACCEPT; } |
  LIST_EXPR_START           list_expr           FEOF { EditTree=$2; YYACCEPT; } |
  MAP_EXPR_START            map_expr            FEOF { EditTree=$2; YYACCEPT; } |
  LAMBDA_PARAMETER_START    lambda_parameter    FEOF { EditTree=$2; YYACCEPT; } |                    130
  QUANTIFIER_START          quantifier          FEOF { EditTree=$2; YYACCEPT; } |
  INFIX_EXPR_START          infix_expr          FEOF { EditTree=$2; YYACCEPT; } |
```

```
  PREFIX_EXPR_START     prefix_expr     FEOF { EditTree=$2; YYACCEPT; } |
  STRUCTURED_EXPR_START structured_expr FEOF { EditTree=$2; YYACCEPT; } |
  LET_DEF_START         let_def         FEOF { EditTree=$2; YYACCEPT; } |
  LET_BINDING_START     let_binding     FEOF { EditTree=$2; YYACCEPT; } |
  BINDING_START         binding         FEOF { EditTree=$2; YYACCEPT; } |
  TYPING_START          typing          FEOF { EditTree=$2; YYACCEPT; } |
  PATTERN_START         pattern         FEOF { EditTree=$2; YYACCEPT; } |
  LIST_PATTERN_START    list_pattern    FEOF { EditTree=$2; YYACCEPT; } |
  INNER_PATTERN_START   inner_pattern   FEOF { EditTree=$2; YYACCEPT; } |
  NAME_START            name            FEOF { EditTree=$2; YYACCEPT; } |
  ID_OR_OP_START        id_or_op        FEOF { EditTree=$2; YYACCEPT; } |
  OP_START              op              FEOF { EditTree=$2; YYACCEPT; } |
  INFIX_OP_START        infix_op        FEOF { EditTree=$2; YYACCEPT; } |
  PREFIX_OP_START       prefix_op       FEOF { EditTree=$2; YYACCEPT; } |
  CONNECTIVE_START      connective      FEOF { EditTree=$2; YYACCEPT; } |
  INFIX_CONNECTIVE_START infix_connective FEOF { EditTree=$2; YYACCEPT; } |
  INFIX_COMBINATOR_START infix_combinator FEOF { EditTree=$2; YYACCEPT; } |
  COMMENT_START         comment         FEOF { EditTree=$2; YYACCEPT; } |
  module_decl                       FEOF { EditTree=$1; YYACCEPT; }
  ;

/* Specifications */

specification :
  module_decl_string { $$=build(new Specification,$1) }
  ;

/* Declarations */

module_decl :
  PH_MODULE_DECL    { $$=new ModuleDecl; }              |
  scheme_decl       { $$=build(new ModuleDecl,$1) } |
  object_decl       { $$=build(new ModuleDecl,$1) }
  ;

module_decl_string :
  module_decl { $$=build(new ModuleDeclString,$1) } |
  module_decl_string module_decl { $1->insert_tree_last($2); $$=$1 }
  ;

/* Declarations */

decl :
  PH_DECL         { $$=new Decl; }                  |
  scheme_decl     { $$=build(new Decl,$1) } |
  object_decl     { $$=build(new Decl,$1) } |
  type_decl       { $$=build(new Decl,$1) } |
  value_decl      { $$=build(new Decl,$1) } |
  variable_decl   { $$=build(new Decl,$1) } |
  channel_decl    { $$=build(new Decl,$1) } |
  axiom_decl      { $$=build(new Decl,$1) }
  ;

decl_string :
  decl { $$=build(new DeclString,$1) } |
  decl_string decl { $1->insert_tree_last($2); $$=$1 }
  ;

/* Scheme Declarations */

scheme_decl :
  KW_SCHEME scheme_def_list { $$=build(new SchemeDecl,$2) }
  ;

scheme_def :
  id EQ class_expr
    { $$=build(new SchemeDef,new OptCommentString,$1,new OptFormalSchemeParameter,$3) } |
```

```
  comment_string id EQ class_expr                                                                    200
    { $$=build(new SchemeDef,build(new OptCommentString,$1),$2,new OptFormalSchemeParameter,$4) } |
  id formal_scheme_parameter EQ class_expr
    { $$=build(new SchemeDef,new OptCommentString,$1,build(new OptFormalSchemeParameter,$2),$4) } |
  comment_string id formal_scheme_parameter EQ class_expr
    { $$=build(new SchemeDef,build(new OptCommentString,$1),$2,build(new OptFormalSchemeParameter,$3),$5) }
  ;

scheme_def_list :
  scheme_def { $$=build(new SchemeDefList,$1) } |
  scheme_def_list COMMA scheme_def { $1->insert_tree_last($3); $$=$1 }                               210
  ;

formal_scheme_parameter :
  LPAR object_def_list RPAR { $$=build(new FormalSchemeParameter,$2) }
  ;

/* Object Declarations */

object_decl :
  KW_OBJECT object_def_list { $$=build(new ObjectDecl,$2) }                                          220
  ;

object_def :
  id COLON class_expr
    { $$=build(new ObjectDef,new OptCommentString,$1,new OptFormalArrayParameter,$3) } |
  comment_string id COLON class_expr
    { $$=build(new ObjectDef,build(new OptCommentString,$1),$2,new OptFormalArrayParameter,$4) } |
  id formal_array_parameter COLON class_expr
    { $$=build(new ObjectDef,new OptCommentString,$1,build(new OptFormalArrayParameter,$2),$4) } |
  comment_string id formal_array_parameter COLON class_expr                                          230
    { $$=build(new ObjectDef,build(new OptCommentString,$1),$2,build(new OptFormalArrayParameter,$3),$5); }
  ;

object_def_list :
  object_def { $$=build(new ObjectDefList,$1) } |
  object_def_list COMMA object_def { $1->insert_tree_last($3); $$=$1 }
  ;

formal_array_parameter :
  LBRACKET typing_list RBRACKET                                                                      240
  { $$=build(new FormalArrayParameter,$2) }
  ;

/* Type Declarations */

type_decl :
  KW_TYPE type_def_list { $$=build(new TypeDecl,$2) }
  ;

type_def :                                                                                           250
  PH_TYPE_DEF       { $$=new TypeDef; } |
  sort_def          { $$=build(new TypeDef,$1) } |
  variant_def       { $$=build(new TypeDef,$1) } |
  union_def         { $$=build(new TypeDef,$1) } |
  short_record_def  { $$=build(new TypeDef,$1) } |
  abbreviation_def  { $$=build(new TypeDef,$1) }
  ;

type_def_list :
  type_def { $$=build(new TypeDefList,$1) } |                                                        260
  type_def_list COMMA type_def { $1->insert_tree_last($3); $$=$1 }
  ;

/* Sort Definitions */

sort_def :
```

```
  id { $$=build(new SortDef,new OptCommentString,$1) } |
  comment_string id { $$=build(new SortDef,build(new OptCommentString,$1),$2) }
  ;
                                                                                    270
/* Variant Definitions */

variant_def :
  id EQEQ variant_choice { $$=build(new VariantDef,new OptCommentString,$1,$3) } |
  comment_string id EQEQ variant_choice { $$=build(new VariantDef,build(new OptCommentString,$1),$2,$4) }
  ;

variant :
  PH_VARIANT       { $$=new Variant; } |
  record_variant { $$=build(new Variant,$1) } |                                     280
  constructor      { $$=build(new Variant,$1) }
  ;

variant_choice :
  variant { $$=build(new VariantChoice,$1) } |
  variant_choice BAR variant { $1->insert_tree_last($3); $$=$1 }
  ;

record_variant :
  constructor LPAR component_kind_list RPAR                                         290
  { $$=build(new RecordVariant,$1,$3); }
  ;

component_kind :
  type_expr
    { $$=build(new ComponentKind,new OptDestructor,$1,new OptReconstructor) } |
  destructor type_expr
    { $$=build(new ComponentKind,build(new OptDestructor,$1),$2,new OptReconstructor) } |
  type_expr reconstructor
    { $$=build(new ComponentKind,new OptDestructor,$2,build(new OptReconstructor,$2)) } |   300
  destructor type_expr reconstructor
    { $$=build(new ComponentKind,build(new OptDestructor,$1),$2,build(new OptReconstructor,$3)) }
  ;

component_kind_list :
  component_kind { $$=build(new ComponentKindList,$1) } |
  component_kind_list COMMA component_kind { $1->insert_tree_last($3); $$=$1 }
  ;

component_kind_string :                                                             310
  component_kind { $$=build(new ComponentKindString,$1) } |
  component_kind_string component_kind { $1->insert_tree_last($2); $$=$1 }
  ;

constructor :
  PH_CONSTRUCTOR { $$=new Constructor } |
  id_or_op   { $$=build(new Constructor,$1) } |
  UNDERLINE { $$=build(new Constructor,new Wildcard) }
  ;
                                                                                    320
destructor :
  id_or_op COLON   { $$=build(new Destructor,$1) }
  ;

reconstructor :
  LEFTRIGHTARROW id_or_op { $$=build(new Reconstructor,$2) }
  ;

/* Union Definitions */
                                                                                    330
union_def :
  id EQ name_or_wildcard_choice2 { $$=build(new UnionDef,new OptCommentString,$1,$3) } |
  comment_string id EQ name_or_wildcard_choice2 { $$=build(new UnionDef,build(new OptCommentString,$1),$2,$4) }
```

```
  ;

name_or_wildcard :
  PH_NAME_OR_WILDCARD { $$=new NameOrWildcard; } |
  name { $$=build(new NameOrWildcard,$1) } |
  UNDERLINE { $$=build(new NameOrWildcard,new Wildcard) }
  ;                                                                    340

name_or_wildcard_choice2 :
  name_or_wildcard BAR name_or_wildcard
    { $$=build(new NameOrWildcardChoice2,$1,$3); } |
  name_or_wildcard_choice2 BAR name_or_wildcard
    { $1->insert_tree_last($3); $$=$1 }
  ;

/* Short Record Definitions */
                                                                       350
short_record_def :
  id COLONCOLON component_kind_string
    { $$=build(new ShortRecordDef,new OptCommentString,$1,$3) } |
  comment_string id COLONCOLON component_kind_string
    { $$=build(new ShortRecordDef,build(new OptCommentString,$1),$2,$4) }
  ;

/* Abbreviation Definitions */

abbreviation_def :                                                     360
  id EQ type_expr { $$=build(new AbbreviationDef,new OptCommentString,$1,$3) } |
  comment_string id EQ type_expr
  { $$=build(new AbbreviationDef,build(new OptCommentString,$1),$2,$4) }
  ;

/* Value Declarations */

value_decl :
  KW_VALUE value_def_list { $$=build(new ValueDecl,$2) }
  ;                                                                    370

value_def :
  PH_VALUE_DEF { $$=new ValueDef; } |
  commented_typing { $$=build(new ValueDef,$1) } |
  explicit_value_def { $$=build(new ValueDef,$1) } |
  implicit_value_def { $$=build(new ValueDef,$1) } |
  explicit_function_def { $$=build(new ValueDef,$1) } |
  implicit_function_def { $$=build(new ValueDef,$1) }
  ;
                                                                       380
value_def_list :
  value_def { $$=build(new ValueDefList,$1) } |
  value_def_list COMMA value_def { $1->insert_tree_last($3); $$=$1 }
  ;

/* Explicit Value Definitions */

explicit_value_def :
  single_typing EQ value_expr { $$=build(new ExplicitValueDef,new OptCommentString,$1,$3) } |
  comment_string single_typing EQ value_expr { $$=build(new ExplicitValueDef,build(new OptCommentString,$1),$2,$4) } 390
  ;

/* Implicit Value Definitions */

implicit_value_def :
  single_typing EQ restriction { $$=build(new ImplicitValueDef,new OptCommentString,$1,$3) } |
  comment_string single_typing EQ restriction { $$=build(new ImplicitValueDef,build(new OptCommentString,$1),$2,$4) }
  ;

/* Explicit Function Definitions */                                    400
```

```
explicit_function_def :
  single_typing formal_function_application IS value_expr12
    { $$=build(new ExplicitFunctionDef,new OptCommentString,$1,$2,$4,new OptPreCondition) } |
  comment_string single_typing formal_function_application IS value_expr12
    { $$=build(new ExplicitFunctionDef,build(new OptCommentString,$1),$2,$3,$5,new OptPreCondition) } |
  single_typing formal_function_application IS value_expr12 pre_condition
    { $$=build(new ExplicitFunctionDef,new OptCommentString,$1,$2,$4,build(new OptPreCondition,$5)) } |
  comment_string single_typing formal_function_application IS value_expr12 pre_condition
    { $$=build(new ExplicitFunctionDef,build(new OptCommentString,$1),$2,$3,$5,build(new OptPreCondition,$6)) }
  ;

formal_function_application :
  PH_FORMAL_FUNCTION_APPLICATION { $$=new FormalFunctionApplication } |
  id_application      { $$=build(new FormalFunctionApplication,$1) } |
  prefix_application { $$=build(new FormalFunctionApplication,$1) } |
  infix_application  { $$=build(new FormalFunctionApplication,$1) }
  ;

id_application :
  id formal_function_parameter_string
  { $$=build(new IdApplication,$1,$2) }
  ;

formal_function_parameter :
  LPAR RPAR { $$=build(new FormalFunctionParameter,new OptBindingList) } |
  LPAR binding_list RPAR
  { $$=build(new FormalFunctionParameter,build(new OptBindingList,$2)) }
  ;

formal_function_parameter_string :
  formal_function_parameter
    { $$=build(new FormalFunctionParameterString,$1) } |
  formal_function_parameter_string formal_function_parameter
    { $1->insert_tree_last($2); $$=$1 }
  ;

prefix_application :
  prefix_op id { $$=build(new PrefixApplication,$1,$2) }
  ;

infix_application :
  id infix_op id { $$=build(new InfixApplication,$1,$2,$3) }
  ;

/* Implicit Function Definitions */

implicit_function_def :
  single_typing formal_function_application post_condition
    { $$=build(new ImplicitFunctionDef,new OptCommentString,$1,$2,$3,new OptPreCondition) }  |
  comment_string single_typing formal_function_application post_condition
    { $$=build(new ImplicitFunctionDef,build(new OptCommentString,$1),$2,$3,$4,new OptPreCondition) } |
  single_typing formal_function_application post_condition pre_condition
    { $$=build(new ImplicitFunctionDef,new OptCommentString,$1,$2,$3,build(new OptPreCondition,$4)) } |
  comment_string single_typing formal_function_application post_condition pre_condition
    { $$=build(new ImplicitFunctionDef,build(new OptCommentString,$1),$2,$3,$4,build(new OptPreCondition,$5)) }
  ;

/* Variable Declarations */

variable_decl :
  KW_VARIABLE variable_def_list { $$=build(new VariableDecl,$2) }
  ;

variable_def :
  PH_VARIABLE_DEF          { $$=new VariableDef; } |
  single_variable_def    { $$=build(new VariableDef,$1) } |
```

```
  multiple_variable_def { $$=build(new VariableDef,$1) }
  ;
                                                                                    470
variable_def_list :
  variable_def { $$=build(new VariableDef,$1) } |
  variable_def_list COMMA variable_def { $1->insert_tree_last($3); $$=$1 }
  ;

single_variable_def :
  id COLON type_expr
    { $$=build(new SingleVariableDef,new OptCommentString,$1,$3,new OptInitialisation) } |
  comment_string id COLON type_expr
    { $$=build(new SingleVariableDef,build(new OptCommentString,$1),$2,$3,new OptInitialisation) } |      480
  id COLON type_expr initialisation
    { $$=build(new SingleVariableDef,new OptCommentString,$1,$3,build(new OptInitialisation,$4)) } |
  comment_string id COLON type_expr initialisation
    { $$=build(new SingleVariableDef,build(new OptCommentString,$1),$2,$4,build(new OptInitialisation,$5)) }
  ;

initialisation :
  COLONEQ value_expr { $$=build(new Initialisation,$2) }
  ;
                                                                                    490
multiple_variable_def :
  id_list2 COLON type_expr { $$=build(new MultipleVariableDef,new OptCommentString,$1,$3) } |
  comment_string id_list2 COLON type_expr { $$=build(new MultipleVariableDef,build(new OptCommentString,$1),$2,$4) }
  ;

/* Channel Declarations */

channel_decl :
  KW_CHANNEL channel_def_list { $$=build(new ChannelDecl,$2) }
  ;                                                                                 500

channel_def :
  PH_CHANNEL_DEF          { $$=new ChannelDef; } |
  single_channel_def    { $$=build(new ChannelDef,$1) } |
  multiple_channel_def { $$=build(new ChannelDef,$1) }
  ;

channel_def_list :
  channel_def { $$=build(new ChannelDefList,$1) } |
  channel_def_list COMMA channel_def { $1->insert_tree_last($3); $$=$1 }                510
  ;

single_channel_def :
  id COLON type_expr { $$=build(new SingleChannelDef,new OptCommentString,$1,$3) } |
  comment_string id COLON type_expr { $$=build(new SingleChannelDef,build(new OptCommentString,$1),$2,$4) }
  ;

multiple_channel_def :
  id_list2 COLON type_expr { $$=build(new MultipleChannelDef,new OptCommentString,$1,$3) } |
  comment_string id_list2 COLON type_expr { $$=build(new MultipleChannelDef,build(new OptCommentString,$1),$2,$4) } 520
  ;

/* Axiom Declarations */

axiom_decl :
  KW_AXIOM axiom_def_list { $$=build(new AxiomDecl,new OptAxiomQuantification,$2) } |
  KW_AXIOM axiom_quantification axiom_def_list { $$=build(new AxiomDecl,build(new OptAxiomQuantification,$2),$3) }
  ;

axiom_quantification :                                                              530
  KW_FORALL typing_list RDOT
  { $$=build(new AxiomQuantification,$2) }
  ;
```

```
axiom_def :
  value_expr
    { $$=build(new AxiomDef,new OptCommentString,new OptAxiomNaming,$1) } |
  comment_string value_expr
    { $$=build(new AxiomDef,build(new OptCommentString,$1),new OptAxiomNaming,$2) } |
  axiom_naming value_expr                                                          540
    { $$=build(new AxiomDef,new OptCommentString,build(new OptAxiomNaming,$1),$2) } |
  comment_string axiom_naming value_expr
    { $$=build(new AxiomDef,build(new OptCommentString,$1),build(new OptAxiomNaming,$2),$3) }
  ;

axiom_def_list :
  axiom_def { $$=build(new AxiomDefList,$1) } |
  axiom_def_list COMMA axiom_def { $1->insert_tree_last($3); $$=$1 }
  ;
                                                                                   550
axiom_naming :
  LBRACKET id RBRACKET
  { $$=build(new AxiomNaming,$2) }
  ;

/* Class Expressions */

class_expr :
  PH_CLASS_EXPR          { $$=new ClassExpr } |
  basic_class_expr       { $$=build(new ClassExpr,$1) } |                          560
  extending_class_expr   { $$=build(new ClassExpr,$1) } |
  hiding_class_expr      { $$=build(new ClassExpr,$1) } |
  renaming_class_expr    { $$=build(new ClassExpr,$1) } |
  scheme_instantiation   { $$=build(new ClassExpr,$1) }
  ;

/* Basic Class Expressions */

basic_class_expr :
  KW_CLASS KW_END { $$=build(new BasicClassExpr,new OptDeclString) } |             570
  KW_CLASS decl_string KW_END
  { $$=build(new BasicClassExpr,build(new OptDeclString,$2)) }
  ;

/* Extending Class Expressions */

extending_class_expr :
  KW_EXTEND class_expr KW_WITH class_expr
  { $$=build(new ExtendingClassExpr,$2,$4) }
  ;                                                                                580

/* Hiding Class Expressions */

hiding_class_expr :
  KW_HIDE defined_item_list KW_IN class_expr
  { $$=build(new HidingClassExpr,$2,$4); }
  ;

/* Renaming Class Expressions */
                                                                                   590
renaming_class_expr :
  KW_USE rename_pair_list KW_IN class_expr
  { $$=build(new RenamingClassExpr,$2,$4); }
  ;

/* Scheme Instantiations */

scheme_instantiation :
  name { $$=build(new SchemeInstantiation,$1,new OptActualSchemeParameter) } |
  name actual_scheme_parameter { $$=build(new SchemeInstantiation,$1,build(new OptActualSchemeParameter,$2)) } 600
  ;
```

```
actual_scheme_parameter :
  LPAR object_expr_list RPAR
  { $$=build(new ActualSchemeParameter,$2) }
  ;

/* Rename Pairs */

rename_pair :                                                                    610
  defined_item KW_FOR defined_item
  { $$=build(new RenamePair,$1,$3) }
  ;

rename_pair_list :
  rename_pair { $$=build(new RenamePairList,$1) } |
  rename_pair_list COMMA rename_pair { $1->insert_tree_last($3); $$=$1 }
  ;

/* Defined Items */                                                              620

defined_item :
  PH_DEFINED_ITEM       { $$=new DefinedItem; } |
  id_or_op              { $$=build(new DefinedItem,$1) } |
  disambiguated_item    { $$=build(new DefinedItem,$1) }
  ;

defined_item_list :
  defined_item { $$=build(new DefinedItemList,$1) } |
  defined_item_list COMMA defined_item { $1->insert_tree_last($3); $$=$1 }       630
  ;

disambiguated_item :
  id_or_op COLON type_expr
  { $$=build(new DisambiguatedItem,$1,$3) }
  ;

/* Object Expressions */

object_expr :                                                                    640
  PH_OBJECT_EXPR        { $$=new ObjectExpr; } |
  name                  { $$=build(new ObjectExpr,$1) } |
  fitting_object_expr { $$=build(new ObjectExpr,$1) } |
  element_object_expr { $$=build(new ObjectExpr,$1) } |
  array_object_expr     { $$=build(new ObjectExpr,$1) }
  ;

object_expr_list :
  object_expr { $$=build(new ObjectExprList,$1) } |
  object_expr_list COMMA object_expr { $1->insert_tree_last($3); $$=$1 }         650
  ;

/* Element Object Expressions */

element_object_expr :
  object_expr actual_array_parameter
  { $$=build(new ElementObjectExpr,$1,$2) }
  ;

actual_array_parameter :                                                         660
  LBRACKET value_expr_list RBRACKET
  { $$=build(new ActualArrayParameter,$2) }
  ;

/* Array Object Expressions */

array_object_expr :
  LARRBRACKET typing_list RDOT object_expr RARRBRACKET
```

```
  { $$=build(new ArrayObjectExpr,$2,$4) }
  ;

/* Fitting Object Expressions */

fitting_object_expr :
  object_expr LBRACE rename_pair_list RBRACE
  { $$=build(new FittingObjectExpr,$1,$3) }
  ;

/* Type Expressions */

type_expr :
  type_expr3              { $$=$1 }
  ;

type_expr3 :
  type_expr2              { $$=$1 } |
  map_type_expr           { $$=build(new TypeExpr,$1) } |
  function_type_expr      { $$=build(new TypeExpr,$1) }
  ;

type_expr2 :
  type_expr1              { $$=$1 } |
  product_type_expr       { $$=build(new TypeExpr,$1) }
  ;

type_expr1 :
  type_expr0              { $$=$1 } |
  set_type_expr           { $$=build(new TypeExpr,$1) } |
  list_type_expr          { $$=build(new TypeExpr,$1) }
  ;

type_expr0 :
  PH_TYPE_EXPR            { $$=new TypeExpr } |
  type_literal            { $$=build(new TypeExpr,$1) } |
  subtype_expr            { $$=build(new TypeExpr,$1) } |
  name                    { $$=build(new TypeExpr,$1) } |
  bracketed_type_expr     { $$=build(new TypeExpr,$1) }
  ;

type_expr_product2 :
  type_expr1 TIMES type_expr1 { $$=build(new TypeExprProduct2,$1,$3) } |
  type_expr_product2 TIMES type_expr1 { $1->insert_tree_last($3); $$=$1 }
  ;

/* Type Literals */

type_literal :
  PH_TYPE_LITERAL { $$=new TypeLiteral } |
  KW_TUNIT  { $$=build(new TypeLiteral,new TUnit) } |
  KW_TBOOL  { $$=build(new TypeLiteral,new TBool) } |
  KW_TINT   { $$=build(new TypeLiteral,new TInt) } |
  KW_TNAT   { $$=build(new TypeLiteral,new TNat) } |
  KW_TREAL  { $$=build(new TypeLiteral,new TReal) } |
  KW_TTEXT  { $$=build(new TypeLiteral,new TText) } |
  KW_TCHAR  { $$=build(new TypeLiteral,new TChar) }
  ;

/* Product Type Expressions */

product_type_expr :
  type_expr_product2 { $$=build(new ProductTypeExpr,$1) }
  ;

/* Set Type Expressions */
```

```
set_type_expr :
  PH_SET_TYPE_EXPR          { $$=new SetTypeExpr } |
  finite_set_type_expr      { $$=build(new SetTypeExpr,$1) } |
  infinite_set_type_expr    { $$=build(new SetTypeExpr,$1) }
  ;                                                                    740

finite_set_type_expr :
  type_expr0 SET { $$=build(new FiniteSetTypeExpr,$1) }
  ;

infinite_set_type_expr :
  type_expr0 INFSET { $$=build(new InfiniteSetTypeExpr,$1) }
  ;

/* List Type Expressions */                                            750

list_type_expr :
  PH_LIST_TYPE_EXPR         { $$=new ListTypeExpr } |
  finite_list_type_expr     { $$=build(new ListTypeExpr,$1) } |
  infinite_list_type_expr   { $$=build(new ListTypeExpr,$1) }
  ;

finite_list_type_expr :
  type_expr0 LIST { $$=build(new FiniteListTypeExpr,$1) }
  ;                                                                    760

infinite_list_type_expr :
  type_expr0 INFLIST { $$=build(new InfiniteListTypeExpr,$1) }
  ;

/* Map Type Expressions */

map_type_expr :
  type_expr2 MARROW type_expr3
  { $$=build(new MapTypeExpr,$1,$3) }                                  770
  ;

/* Function Type Expressions */

function_type_expr :
  type_expr2 function_arrow result_desc
  { $$=build(new FunctionTypeExpr,$1,$2,$3) }
  ;

function_arrow :                                                       780
  PH_FUNCTION_ARROW { $$=new FunctionArrow } |
  PARRIGHTARROW     { $$=build(new FunctionArrow,new ParRightArrow) } |
  RIGHTARROW        { $$=build(new FunctionArrow,new RightArrow) }
  ;

result_desc :
  type_expr3 |
  access_desc_string type_expr3 { $$=build(new ResultDesc,$1,$2) }
  ;
                                                                       790
/* Subtype Expressions */

subtype_expr :
  LTYPEBRACE single_typing restriction RTYPEBRACE
  { $$=build(new SubTypeExpr,$2,$3) }
  ;

/* Bracketed Type Expressions */

bracketed_type_expr :                                                  800
  LPAR type_expr RPAR { $$=build(new BracketedTypeExpr,$2) }
  ;
```

```
/* Access Descriptions */

access_desc :
  access_mode access_list { $$=build(new AccessDesc,$1,$2) }
  ;

access_desc_string :                                                    810
  access_desc { $$=build(new AccessDescString,$1) } |
  access_desc_string access_desc { $1->insert_tree_last($2); $$=$1 }
  ;

access_mode :
  PH_ACCESS_MODE { $$=new AccessMode } |
  KW_READ        { $$=build(new AccessMode,new Read) } |
  KW_WRITE       { $$=build(new AccessMode,new Write) } |
  KW_IN          { $$=build(new AccessMode,new In) }  |
  KW_OUT         { $$=build(new AccessMode,new Out) }                   820
  ;

access :
  PH_ACCESS            { $$=new Access } |
  name                 { $$=build(new Access,$1) } |
  enumerated_access    { $$=build(new Access,$1) } |
  comprehended_access  { $$=build(new Access,$1) } |
  completed_access     { $$=build(new Access,$1) }
  ;
                                                                        830
access_list :
  access { $$=build(new AccessList,$1) } |
  access_list COMMA access { $1->insert_tree_last($3); $$=$1 }
  ;

enumerated_access :
  LBRACE RBRACE { $$=build(new EnumeratedAccess,new OptAccessList) } |
  LBRACE access_list RBRACE
  { $$=build(new EnumeratedAccess,build(new OptAccessList,$2)) }
  ;                                                                     840

completed_access :
  KW_ANY { $$=build(new CompletedAccess,new OptQualification) } |
  qualification KW_ANY { $$=build(new CompletedAccess,build(new OptQualification,$1)) }
  ;

comprehended_access :
  LBRACE access BAR set_limitation RBRACE
  { $$=build(new ComprehendedAccess,$2,$4) }
  ;                                                                     850

/* Value Expr */

value_expr :
/* value_expr15                    { $$=new ValueExpr; $$->insert($1) } |*/
  value_expr14            { $$=$1 }
  ;

/*
value_expr15 :                                                          860
  implementation_relation          |
  implementation_expr              |
  class_scope_expr
  ;
*/

value_expr14 :
  value_expr13            { $$=$1 } |
  quantified_expr         { $$=build(new ValueExpr,$1) } |
```

```
  prefix_expr14          { $$=build(new ValueExpr,$1) } |
  function_expr          { $$=build(new ValueExpr,$1) }
  ;

value_expr13 :
  value_expr12           { $$=$1 } |
  post_expr              { $$=build(new ValueExpr,$1) } |
  equivalence_expr       { $$=build(new ValueExpr,$1) }
  ;

value_expr12 :
  value_expr11           { $$=$1 } |
  infix_expr12           { $$=build(new ValueExpr,$1) }
  ;

value_expr11 :
  value_expr10           { $$=$1 } |
  infix_expr11           { $$=build(new ValueExpr,$1) }
  ;

value_expr10 :
  value_expr9            { $$=$1 } |
  output_expr            { $$=build(new ValueExpr,$1) } |
  assignment_expr        { $$=build(new ValueExpr,$1) }
  ;

value_expr9 :
  value_expr8            { $$=$1 } |
  infix_expr9            { $$=build(new ValueExpr,$1) }
  ;

value_expr8 :
  value_expr7            { $$=$1 } |
  infix_expr8            { $$=build(new ValueExpr,$1) }
  ;

value_expr7 :
  value_expr6            { $$=$1 } |
  infix_expr7            { $$=build(new ValueExpr,$1) }
  ;

value_expr6 :
  value_expr5            { $$=$1 } |
  infix_expr6            { $$=build(new ValueExpr,$1) }
  ;

value_expr5 :
  value_expr4            { $$=$1 } |
  infix_expr5            { $$=build(new ValueExpr,$1) }
  ;

value_expr4 :
  value_expr3            { $$=$1 }|
  infix_expr4            { $$=build(new ValueExpr,$1) }
  ;

value_expr3 :
  value_expr2            { $$=$1 } |
  infix_expr3            { $$=build(new ValueExpr,$1) }
  ;

value_expr2 :
  value_expr1            { $$=$1 } |
  disambiguation_expr    { $$=build(new ValueExpr,$1) }
  ;

value_expr1 :
```

```
  value_expr0              { $$=$1 } |
  prefix_expr1             { $$=build(new ValueExpr,$1) }
  ;
                                                                                    940
value_expr0 :
  value_expr255            { $$=$1 } |
  application_expr         { $$=build(new ValueExpr,$1) }
  ;

value_expr255 :
  PH_VALUE_EXPR            { $$=new ValueExpr } |
  value_literal            { $$=build(new ValueExpr,$1) } |
  structured_expr          { $$=build(new ValueExpr,$1) } |
  set_expr                 { $$=build(new ValueExpr,$1) } |                        950
  product_expr             { $$=build(new ValueExpr,$1) } |
  pre_name                 { $$=build(new ValueExpr,$1) } |
  name                     { $$=build(new ValueExpr,$1) } |
  map_expr                 { $$=build(new ValueExpr,$1) } |
  list_expr                { $$=build(new ValueExpr,$1) } |
  input_expr               { $$=build(new ValueExpr,$1) } |
  initialise_expr          { $$=build(new ValueExpr,$1) } |
  comprehended_expr        { $$=build(new ValueExpr,$1) } |
  bracketed_expr           { $$=build(new ValueExpr,$1) } |
  basic_expr               { $$=build(new ValueExpr,$1) }                          960
  ;

value_expr_list2 :
  value_expr COMMA value_expr { $$=build(new ValueExprList2,$1,$3) } |
  value_expr_list2 COMMA value_expr { $1->insert_tree_last($3); $$=$1 }
  ;

value_expr_list :
  value_expr { $$=build(new ValueExprList,$1) } |
  value_expr_list COMMA value_expr { $1->insert_tree_last($3); $$=$1 }              970
  ;

/* Value Literals */

value_literal :
  PH_VALUE_LITERAL { $$=new ValueLiteral } |
  unit_literal     { $$=build(new ValueLiteral,$1) } |
  bool_literal     { $$=build(new ValueLiteral,$1) } |
  INT_LITERAL      { $$=build(new ValueLiteral,new IntLiteral(yylloc.text)) } |
  REAL_LITERAL     { $$=build(new ValueLiteral,new RealLiteral(yylloc.text)) } |   980
  TEXT_LITERAL     { $$=build(new ValueLiteral,new TextLiteral(yylloc.text)) } |
  CHAR_LITERAL     { $$=build(new ValueLiteral,new CharLiteral(yylloc.text)) }
  ;

unit_literal :
  LPAR RPAR { $$=new UnitLiteral; }
  ;

bool_literal :
  PH_BOOL_LITERAL { $$=new BoolLiteral } |                                          990
  KW_TRUE    { $$=build(new BoolLiteral,new True) } |
  KW_FALSE   { $$=build(new BoolLiteral,new False) }
  ;

/* Pre Names */

pre_name :
  name BQUOT { $$=build(new PreName,$1) }
  ;
                                                                                    1000
/* Basic Expressions */

basic_expr :
```

```
  PH_BASIC_EXPR { $$=new BasicExpr } |
  KW_CHAOS    { $$=build(new BasicExpr,new Chaos) } |
  KW_SKIP     { $$=build(new BasicExpr,new Skip) }  |
  KW_STOP     { $$=build(new BasicExpr,new Stop) }  |
  KW_SWAP     { $$=build(new BasicExpr,new Swap) }
  ;
                                                                                    1010
/* Product Expressions */

product_expr :
  LPAR value_expr_list2 RPAR { $$=build(new ProductExpr,$2) }
  ;

/* Set Expressions */

set_expr :
  PH_SET_EXPR          { $$=new SetExpr } |                                         1020
  ranged_set_expr      { $$=build(new SetExpr,$1) } |
  enumerated_set_expr  { $$=build(new SetExpr,$1) } |
  comprehended_set_expr { $$=build(new SetExpr,$1) }
  ;

/* Ranged Set Expressions */

ranged_set_expr :
  LBRACE value_expr DOTDOT value_expr RBRACE
  { $$=build(new RangedSetExpr,$2,$4) }                                            1030
  ;

/* Enumerated Set Expressions */

enumerated_set_expr :
  LBRACE RBRACE { $$=build(new EnumeratedSetExpr,new OptValueExprList) } |
  LBRACE value_expr_list RBRACE { $$=build(new EnumeratedSetExpr,build(new ValueExprList,$2)) }
  ;

/* Comprehended Set Expressions */                                                  1040

comprehended_set_expr :
  LBRACE value_expr BAR set_limitation RBRACE
  { $$=build(new ComprehendedSetExpr,$2,$4) }
  ;

set_limitation :
  typing_list { $$=build(new SetLimitation,$1,new OptRestriction) } |
  typing_list restriction { $$=build(new SetLimitation,$1,build(new OptRestriction,$2)) }
  ;                                                                                 1050

restriction :
  RDOT value_expr { $$=build(new Restriction,$2) }
  ;

/* List Expressions */

list_expr :
  PH_LIST_EXPR              { $$=new ListExpr } |
  ranged_list_expr          { $$=build(new ListExpr,$1) } |                         1060
  enumerated_list_expr      { $$=build(new ListExpr,$1) } |
  comprehended_list_expr    { $$=build(new ListExpr,$1) }
  ;

/* Ranged List Expressions */

ranged_list_expr :
  LANGLE value_expr DOTDOT value_expr RANGLE
  { $$=build(new RangedListExpr,$2,$4) }
  ;                                                                                 1070
```

```
/* Enumerated List Expressions */

enumerated_list_expr :
  LANGLE RANGLE { $$=build(new EnumeratedListExpr,new OptValueExprList) } |
  LANGLE value_expr_list RANGLE { $$=build(new EnumeratedListExpr,build(new OptValueExprList,$2)) }
  ;

/* Comprehended List Expressions */

comprehended_list_expr :
  LANGLE value_expr BAR list_limitation RANGLE
  { $$=build(new ComprehendedListExpr,$2,$4) }
  ;

list_limitation :
  binding KW_IN value_expr { $$=build(new ListLimitation,$1,$3,new OptRestriction) } |
  binding KW_IN value_expr restriction { $$=build(new ListLimitation,$1,$3,build(new OptRestriction,$4)) }
  ;

/* Map Expressions */

map_expr :
  PH_MAP_EXPR         { $$=new MapExpr } |
  enumerated_map_expr { $$=build(new MapExpr,$1) } |
  comprehended_map_expr { $$=build(new MapExpr,$1) }
  ;

/* Enumerated Map Expressions */

enumerated_map_expr :
  LBRACKET RBRACKET { $$=build(new EnumeratedMapExpr,new OptValueExprPairList) } |
  LBRACKET value_expr_pair_list RBRACKET { $$=build(new EnumeratedMapExpr,build(new OptValueExprPairList,$2)) }
  ;

value_expr_pair :
  value_expr MAPSTO value_expr
  { $$=build(new ValueExprPair,$1,$3) }
  ;

value_expr_pair_list :
  value_expr_pair { $$=build(new ValueExprPairList,$1) } |
  value_expr_pair_list COMMA value_expr_pair
    { $1->insert_tree_last($3); $$=$1 }
  ;

/* Comprehended Map Expressions */

comprehended_map_expr :
  LBRACKET value_expr_pair BAR set_limitation RBRACKET
  { $$=build(new ComprehendedMapExpr,$2,$4) }
  ;

/* Function Expressions */

function_expr :
  LAMBDA lambda_parameter RDOT value_expr14
  { $$=build(new FunctionExpr,$2,$4) }
  ;

lambda_parameter :
  PH_LAMBDA_PARAMETER { $$=new LambdaParameter } |
  lambda_typing       { $$=build(new LambdaParameter,$1) } |
  single_typing       { $$=build(new LambdaParameter,$1) }
  ;

lambda_typing :
```

```
  LPAR RPAR { $$=build(new LambdaTyping,new OptTypingList) } |
  LPAR typing_list RPAR { $$=build(new LambdaTyping,build(new OptTypingList,$2)) }
  ;                                                                                    1140

/* Application Expressions */

application_expr :
  value_expr255 actual_function_parameter_string
  { $$=build(new ApplicationExpr,$1,$2) }
  ;

actual_function_parameter :
  LPAR RPAR { $$=build(new ActualFunctionParameter,new OptValueExprList) } |           1150
  LPAR value_expr_list RPAR
  { $$=build(new ActualFunctionParameter,build(new OptValueExprList,$2)) }
  ;

actual_function_parameter_string :
  actual_function_parameter { $$=build(new ActualFunctionParameterString,$1) } |
  actual_function_parameter_string actual_function_parameter
    { $1->insert_tree_last($2); $$=$1 }
  ;
                                                                                       1160
/* Quantified Expressions */

quantified_expr :
  quantifier typing_list restriction
  { $$=build(new QuantifiedExpr,$1,$2,$3) }
  ;

quantifier :
  PH_QUANTIFIER { $$=new Quantifier } |
  ALL           { $$=build(new Quantifier,new Forall) } |                              1170
  EXISTS        { $$=build(new Quantifier,new Exists) } |
  EXISTS EXCLAM { $$=build(new Quantifier,new ExistsUnique) }
  ;

/* Equivalence Expressions */

equivalence_expr :
  value_expr12 IS value_expr12 { $$=build(new EquivalenceExpr,$1,$3,new OptPreCondition) } |
  value_expr12 IS value_expr12 pre_condition { $$=build(new EquivalenceExpr,$1,$3,build(new OptPreCondition,$4)) }
  ;                                                                                    1180

pre_condition :
  KW_PRE value_expr12
  { $$=build(new PreCondition,$2) }
  ;

/* Post Expressions */

post_expr :
  value_expr12 post_condition { $$=build(new PostExpr,$1,$2,new OptPreCondition) } |   1190
  value_expr12 post_condition pre_condition { $$=build(new PostExpr,$1,$2,build(new OptPreCondition,$3)) }
  ;

post_condition :
  KW_POST value_expr12 { $$=build(new PostCondition,new OptResultNaming,$2) } |
  result_naming KW_POST value_expr12 { $$=build(new PostCondition,build(new OptResultNaming,$1),$3) }
  ;

result_naming :
  KW_AS binding { $$=build(new ResultNaming,$2) }                                      1200
  ;

/* Disambiguation Expressions */
```

```
disambiguation_expr :
  value_expr1 COLON type_expr
  { $$=build(new DisambiguatedExpr,$1,$3) }
  ;

/* Bracketed Expressions */                                                    1210

bracketed_expr :
  LPAR value_expr RPAR
  { $$=build(new BracketedExpr,$2) }
  ;

/* Infix Expressions */

infix_expr :
  PH_INFIX_EXPR     { $$=new InfixExpr } |                                   1220
  infix_expr9       { $$=$1 } |
  infix_expr8       { $$=$1 } |
  infix_expr7       { $$=$1 } |
  infix_expr6       { $$=$1 } |
  infix_expr5       { $$=$1 } |
  infix_expr4       { $$=$1 } |
  infix_expr3       { $$=$1 } |
  infix_expr12      { $$=$1 } |
  infix_expr11      { $$=$1 }
  ;                                                                          1230

infix_expr9 : axiom_infix_expr9 { $$=build(new InfixExpr,$1) } ;
infix_expr8 : axiom_infix_expr8 { $$=build(new InfixExpr,$1) } ;
infix_expr7 : axiom_infix_expr7 { $$=build(new InfixExpr,$1) } ;
infix_expr6 : value_infix_expr6 { $$=build(new InfixExpr,$1) } ;
infix_expr5 : value_infix_expr5 { $$=build(new InfixExpr,$1) } ;
infix_expr4 : value_infix_expr4 { $$=build(new InfixExpr,$1) } ;
infix_expr3 : value_infix_expr3 { $$=build(new InfixExpr,$1) } ;
infix_expr12 : stmt_infix_expr12 { $$=build(new InfixExpr,$1) } ;
infix_expr11 : stmt_infix_expr11 { $$=build(new InfixExpr,$1) } ;          1240


/* Statement Infix Expressions */

/*
stmt_infix_expr :
  stmt_infix_expr12     { $$=$1 } |
  stmt_infix_expr11     { $$=$1 }
  ;
*/                                                                         1250

stmt_infix_expr12 :
  value_expr12 infix_combinator12 value_expr11 { $$=build(new StmtInfixExpr,$1,$2,$3) }
  ;

stmt_infix_expr11 :
  value_expr10 infix_combinator11 value_expr11 { $$=build(new StmtInfixExpr,$1,$2,$3) }
  ;

                                                                           1260
/* Axiom Infix Expressions */

/*
axiom_infix_expr :
  axiom_infix_expr9     { $$=$1 } |
  axiom_infix_expr8     { $$=$1 } |
  axiom_infix_expr7     { $$=$1 }
  ;
*/                                                                         1270

axiom_infix_expr9 :
```

```
  value_expr8 infix_connective9 value_expr9 { $$=build(new AxiomInfixExpr,$1,$2,$3) }
  ;

axiom_infix_expr8 :
  value_expr7 infix_connective8 value_expr8 { $$=build(new AxiomInfixExpr,$1,$2,$3) }
  ;

axiom_infix_expr7 :
  value_expr6 infix_connective7 value_expr7 { $$=build(new AxiomInfixExpr,$1,$2,$3) }
  ;


/* Value Infix Expressions */

/*
value_infix_expr :
  value_infix_expr6       { $$=$1 } |
  value_infix_expr5       { $$=$1 } |
  value_infix_expr4       { $$=$1 } |
  value_infix_expr3       { $$=$1 }
  ;
*/

value_infix_expr6 :
  value_expr5 infix_op6 value_expr5 { $$=build(new ValueInfixExpr,$1,$2,$3) }
  ;

value_infix_expr5 :
  value_expr5 infix_op5 value_expr4 { $$=build(new ValueInfixExpr,$1,$2,$3) }
  ;

value_infix_expr4 :
  value_expr4 infix_op4 value_expr3 { $$=build(new ValueInfixExpr,$1,$2,$3) }
  ;

value_infix_expr3 :
  value_expr2 infix_op3 value_expr2 { $$=build(new ValueInfixExpr,$1,$2,$3) }
  ;


/* Prefix Expressions */

prefix_expr :
  PH_PREFIX_EXPR     { $$=new PrefixExpr } |
  prefix_expr1       { $$=$1 } |
  prefix_expr14      { $$=$1 }
  ;

prefix_expr14 :
  universal_prefix_expr { $$=build(new PrefixExpr,$1) }
  ;

prefix_expr1 :
  axiom_prefix_expr       { $$=build(new PrefixExpr,$1) } |
  value_prefix_expr       { $$=build(new PrefixExpr,$1) }
  ;

/* Axiom Prefix Expressions */

axiom_prefix_expr :
  prefix_connective value_expr1
  { $$=build(new AxiomPrefixExpr,$1,$2) }
  ;

/* Universal Prefix Expressions */
```

```
universal_prefix_expr :
  ALWAYS value_expr14 { $$=build(new UniversalPrefixExpr,$2) }
  ;

/* Value Prefix Expressions */

value_prefix_expr :
  prefix_op value_expr1 { $$=build(new ValuePrefixExpr,$1,$2) }
  ;

/* Comprehended Expressions */

comprehended_expr :
  infix_combinator LBRACE value_expr BAR set_limitation RBRACE
  { $$=build(new ComprehendedExpr,$1,$3,$5) }
  ;

/* Initialise Expressions */

initialise_expr :
  KW_INITIALISE { $$=build(new InitialiseExpr,new OptQualification) } |
  qualification KW_INITIALISE { $$=build(new InitialiseExpr,$1) }
  ;

/* Assignment Expressions */

assignment_expr :
  name COLONEQ value_expr9
  { $$=build(new AssignmentExpr,$1,$3) }
  ;

/* Input Expressions */

input_expr :
  name QUERY { $$=build(new InputExpr,$1) }
  ;

/* Output Expressions */

output_expr :
  name EXCLAM value_expr9 { $$=build(new OutputExpr,$1,$3) }
  ;

/* Structured Expressions */

structured_expr :
  PH_STRUCTURED_EXPR { $$=new StructuredExpr } |
  while_expr    { $$=build(new StructuredExpr,$1) } |
  until_expr    { $$=build(new StructuredExpr,$1) } |
  local_expr    { $$=build(new StructuredExpr,$1) } |
  let_expr      { $$=build(new StructuredExpr,$1) } |
  if_expr       { $$=build(new StructuredExpr,$1) } |
  for_expr      { $$=build(new StructuredExpr,$1) } |
  case_expr     { $$=build(new StructuredExpr,$1) }
  ;

/* Local Expressions */

local_expr :
  KW_LOCAL KW_IN value_expr KW_END { $$=build(new LocalExpr,new OptDeclString,$3) } |
  KW_LOCAL decl_string KW_IN value_expr KW_END
  { $$=build(new LocalExpr,build(new OptDeclString,$2),$4) }
  ;

/* Let Expressions */

let_expr :
```

```
  KW_LET let_def_list KW_IN value_expr KW_END
  { $$=build(new LetExpr,$2,$4) }
  ;

let_def :                                                                    1410
  PH_LET_DEF { $$=new LetDef } |
  typing { $$=build(new LetDef,$1) } |
  implicit_let { $$=build(new LetDef,$1) } |
  explicit_let { $$=build(new LetDef,$1) }
  ;

let_def_list :
  let_def { $$=build(new LetDefList,$1) } |
  let_def_list COMMA let_def { $1->insert_tree_last($3); $$=$1 }
  ;                                                                          1420

explicit_let :
  let_binding EQ value_expr
  { $$=build(new ExplicitLet,$1,$3) }
  ;

implicit_let :
  single_typing restriction { $$=build(new ImplicitLet,$1,$2) }
  ;
                                                                             1430
let_binding :
  PH_LET_BINDING { $$=new LetBinding } |
  record_pattern { $$=build(new LetBinding,$1) } |
  list_pattern { $$=build(new LetBinding,$1) } |
  binding { $$=build(new LetBinding,$1) }
  ;

/* If Expressions */

if_expr :                                                                    1440
  KW_IF value_expr KW_THEN value_expr KW_END
    { $$=build(new IfExpr,$2,$4,new OptElsifBranchString,new OptElseBranch) } |
  KW_IF value_expr KW_THEN value_expr elsif_branch_string KW_END
    { $$=build(new IfExpr,$2,$4,build(new OptElsifBranchString,$3),new OptElseBranch) } |
  KW_IF value_expr KW_THEN value_expr else_branch KW_END
    { $$=build(new IfExpr,$2,$4,new OptElsifBranchString,build(new OptElseBranch,$5)) } |
  KW_IF value_expr KW_THEN value_expr elsif_branch_string else_branch KW_END
    { $$=build(new IfExpr,$2,$4,build(new OptElsifBranchString,$5),build(new OptElseBranch,$6)) }
  ;
                                                                             1450
elsif_branch :
  KW_ELSIF value_expr KW_THEN value_expr
  { $$=build(new ElsifBranch,$2,$4) }
  ;

elsif_branch_string :
  elsif_branch { $$=build(new ElsifBranchString,$1) } |
  elsif_branch_string elsif_branch { $1->insert_tree_last($2); $$=$1 }
  ;
                                                                             1460
else_branch :
  KW_ELSE value_expr { $$=build(new ElseBranch,$2) }
  ;

/* Case Expressions */

case_expr :
  KW_CASE value_expr KW_OF case_branch_list KW_END
  { $$=build(new CaseExpr,$2,$4) }
  ;                                                                          1470

case_branch :
```

```
  pattern RIGHTARROW value_expr
  { $$=build(new CaseBranch,$1,$3) }
  ;

case_branch_list :
  case_branch { $$=build(new CaseBranchList,$1) } |
  case_branch_list COMMA case_branch { $1->insert_tree_last($3); $$=$1 }
  ;

/* While Expression */

while_expr :
  KW_WHILE value_expr KW_DO value_expr KW_END
  { $$=build(new WhileExpr,$2,$4) }
  ;

/* Until Expressions */

until_expr :
  KW_DO value_expr KW_UNTIL value_expr KW_END
  { $$=build(new UntilExpr,$2,$4) }
  ;

/* For Expressions */

for_expr :
  KW_FOR list_limitation KW_DO value_expr KW_END
  { $$=build(new ForExpr,$2,$4) }
  ;

/* Bindings */

binding :
  PH_BINDING          { $$=new Binding } |
  product_binding     { $$=build(new Binding,$1) } |
  id_or_op            { $$=build(new Binding,$1) }
  ;

binding_list :
  binding { $$=build(new BindingList,$1) } |
  binding_list COMMA binding { $1->insert_tree_last($3); $$=$1 }
  ;

binding_list2 :
  binding COMMA binding { $$=build(new BindingList2,$1,$3) } |
  binding_list2 COMMA binding { $1->insert_tree_last($3); $$=$1 }
  ;

product_binding :
  LPAR binding_list2 RPAR
  { $$=build(new ProductBinding,$2) }
  ;

/* Typings */

typing :
  PH_TYPING           { $$=new Typing } |
  single_typing       { $$=build(new Typing,$1) } |
  multiple_typing     { $$=build(new Typing,$1) }
  ;

typing_list :
  typing { $$=build(new TypingList,$1) } |
  typing_list COMMA typing { $1->insert_tree_last($3); $$=$1 }
  ;

single_typing :
```

```
  binding COLON type_expr
  { $$=build(new SingleTyping,$1,$3) }
  ;

multiple_typing :
  binding_list2 COLON type_expr
  { $$=build(new MultipleTyping,$1,$3) }
  ;

commented_typing :
  typing { $$=build(new CommentedTyping,new OptCommentString,$1) } |
  comment_string typing { $$=build(new CommentedTyping,build(new OptCommentString,$1),$2) }
  ;


/* Patterns */

pattern :
  PH_PATTERN          { $$=new Pattern; } |
  wildcard_pattern    { $$=build(new Pattern,$1) } |
  value_literal       { $$=build(new Pattern,$1) } |
  record_pattern      { $$=build(new Pattern,$1) } |
  product_pattern     { $$=build(new Pattern,$1) } |
  name                { $$=build(new Pattern,$1) } |
  list_pattern        { $$=build(new Pattern,$1) }
  ;

/* Wildcard Patterns */

wildcard_pattern :
  UNDERLINE { $$=new Wildcard; }
  ;

/* Product Patterns */

product_pattern :
  LPAR inner_pattern_list2 RPAR
  { $$=build(new ProductPattern,$2) }
  ;

/* Record Patterns */

record_pattern :
  name LPAR inner_pattern_list RPAR
  { $$=build(new RecordPattern,$1,$3) }
  ;

/* List Patterns */

list_pattern :
  PH_LIST_PATTERN           { $$=new ListPattern } |
  right_list_pattern        { $$=build(new ListPattern,$1) } |
  enumerated_list_pattern   { $$=build(new ListPattern,$1) } |
  concatenated_list_pattern { $$=build(new ListPattern,$1) }
  ;

/* Right List Patterns */

right_list_pattern :
  id CONCAT enumerated_list_pattern { $$=build(new RightListPattern,$1,$3) }
  ;

/* Enumerated List Patterns */

enumerated_list_pattern :
  LANGLE RANGLE { $$=build(new EnumeratedListPattern,new OptInnerPatternList) } |
  LANGLE inner_pattern_list RANGLE
```

```
  { $$=build(new EnumeratedListPattern,build(new OptInnerPatternList,$2)) }
  ;

/* Concatenated List Patterns */

concatenated_list_pattern :
  enumerated_list_pattern CONCAT inner_pattern
  { $$=build(new ConcatenatedListPattern,$1,$3) }
  ;

/* Inner Patterns */

inner_pattern :
  PH_INNER_PATTERN { $$=new InnerPattern } |
  wildcard_pattern { $$=build(new InnerPattern,$1) } |
  value_literal { $$=build(new InnerPattern,$1) } |
  record_pattern { $$=build(new InnerPattern,$1) } |
  product_pattern { $$=build(new InnerPattern,$1) } |
  list_pattern { $$=build(new InnerPattern,$1) } |
  id_or_op { $$=build(new InnerPattern,$1) } |
  equality_pattern { $$=build(new InnerPattern,$1) }
  ;

inner_pattern_list :
  inner_pattern { $$=build(new InnerPatternList,$1) } |
  inner_pattern_list COMMA inner_pattern { $1->insert_tree_last($3); $$=$1 }
  ;

inner_pattern_list2 :
  inner_pattern COMMA inner_pattern
    { $$=build(new InnerPatternList2,$1,$2) } |
  inner_pattern_list2 COMMA inner_pattern
    { $1->insert_tree_last($3); $$=$1 }
  ;

/* Equality Patterns */

equality_pattern :
  EQ name { $$=build(new EqualityPattern,$2) }
  ;

/* Names */

name :
  PH_NAME { $$=new Name } |
  qualified_id { $$=build(new Name,$1) } |
  qualified_op { $$=build(new Name,$1) }
  ;

/* Qualified Identifiers */

qualified_id :
  id { $$=build(new QualifiedId,new OptQualification,$1) } |
  qualification id { $$=build(new QualifiedId,build(new OptQualification,$1),$2) }
  ;

qualification :
  object_expr DOT { $$=build(new Qualification,$1) }
  ;

/* Qualified Operators */

qualified_op :
  LPAR op RPAR { $$=build(new QualifiedOp,new OptQualification,$2) } |
  qualification LPAR op RPAR { $$=build(new QualifiedOp,build(new OptQualification,$1),$3) }
  ;
```

```
/* Identifiers and Operators */

id_or_op :
  PH_ID_OR_OP { $$=new IdOrOp } |
  op          { $$=build(new IdOrOp,$1) } |
  id          { $$=build(new IdOrOp,$1) }
  ;

op :
  PH_OP       { $$=new Op } |
  infix_op    { $$=build(new Op,$1) } |
  prefix_op   { $$=build(new Op,$1) }
  ;

/* Infix Operators */

infix_op :
  PH_INFIX_OP { $$=new InfixOp } |
  infix_op6   { $$=$1 } |
  infix_op5   { $$=$1 } |
  infix_op4   { $$=$1 } |
  infix_op3   { $$=$1 }
  ;

infix_op6 :
  ISIN        { $$=build(new InfixOp,new IsIn) } |
  SUPSET      { $$=build(new InfixOp,new SuperSet) } |
  SUBSET      { $$=build(new InfixOp,new Subset) } |
  SUPSETEQ    { $$=build(new InfixOp,new ProperSuperset) } |
  SUBSETEQ    { $$=build(new InfixOp,new ProperSubset) } |
  NOTISIN     { $$=build(new InfixOp,new NotIsIn) } |
  NOTEQ       { $$=build(new InfixOp,new NotEqual) } |
  EQ          { $$=build(new InfixOp,new Equal) } |
  GT          { $$=build(new InfixOp,new Gt) } |
  LT          { $$=build(new InfixOp,new Lt) } |
  GEQ         { $$=build(new InfixOp,new GE) } |
  LEQ         { $$=build(new InfixOp,new LE) }
  ;

infix_op5 :
  UNION       { $$=build(new InfixOp,new Union) } |
  PLUS        { $$=build(new InfixOp,new Plus) } |
  MINUS       { $$=build(new InfixOp,new Minus) } |
  SETMINUS    { $$=build(new InfixOp,new Backslash) } |
  CONCAT      { $$=build(new InfixOp,new Concat) } |
  DAGGER      { $$=build(new InfixOp,new Override) }
  ;

infix_op4 :
  AST         { $$=build(new InfixOp,new Mult) } |
  SLASH       { $$=build(new InfixOp,new Divide) } |
  HASH        { $$=build(new InfixOp,new Composition) } |
  INTER       { $$=build(new InfixOp,new Inter) }
  ;

infix_op3 :
  UPARROW     { $$=build(new InfixOp,new Exp) }
  ;

/* Prefix Operators */

prefix_op :
  PH_PREFIX_OP { $$=new PrefixOp } |
  KW_ABS       { $$=build(new PrefixOp,new Abs) } |
  KW_INT       { $$=build(new PrefixOp,new Int) } |
  KW_REAL      { $$=build(new PrefixOp,new Real) } |
  KW_CARD      { $$=build(new PrefixOp,new Card) } |
```

```
  KW_LEN      { $$=build(new PrefixOp,new Len) } |
  KW_INDS     { $$=build(new PrefixOp,new Inds) } |
  KW_ELEMS    { $$=build(new PrefixOp,new Elems) } |
  KW_HD       { $$=build(new PrefixOp,new Hd) } |
  KW_TL       { $$=build(new PrefixOp,new Tl) } |
  KW_DOM      { $$=build(new PrefixOp,new Dom) } |
  KW_RNG      { $$=build(new PrefixOp,new Rng) }
  ;

/* Connectives */

connective :
  PH_CONNECTIVE     { $$=new Connective } |
  infix_connective  { $$=build(new Connective,$1) } |
  prefix_connective { $$=build(new Connective,$1) }
  ;

/* Infix Connectives */

infix_connective :
  PH_INFIX_CONNECTIVE { $$=new InfixConnective } |
  infix_connective9   { $$=$1 } |
  infix_connective8   { $$=$1 } |
  infix_connective7   { $$=$1 }
  ;

infix_connective9 :
  DBLRIGHTARROW { $$=build(new InfixConnective,new Implication) }
  ;

infix_connective8 :
  VEE           { $$=build(new InfixConnective,new Or) }
  ;

infix_connective7 :
  WEDGE         { $$=build(new InfixConnective,new And) }
  ;

/* Prefix Connectives */

prefix_connective : SIM { $$=build(new PrefixConnective,new Not) }
  ;

/* Infix Combinators */

infix_combinator :
  PH_INFIX_COMBINATOR { $$=new InfixCombinator } |
  infix_combinator12  { $$=$1 } |
  infix_combinator11  { $$=$1 }
  ;

infix_combinator12 :
  DETCHOICE     { $$=build(new InfixCombinator,new ExtChoice) } |
  NONDETCHOICE  { $$=build(new InfixCombinator,new IntChoice) } |
  PARL          { $$=build(new InfixCombinator,new ConComposition) } |
  TIE           { $$=build(new InfixCombinator,new IntComposition) }
  ;

infix_combinator11 :
  SEMI          { $$=build(new InfixCombinator,new SeqComposition) }
  ;

/*** Lexical Matters ***/

comment :
  PH_COMMENT { $$=new Comment } |
  COMMENTSTART comment_token_string COMMENTEND { $$=build(new Comment,$2) }
```

```
  ;

                                                                        1810
comment_string :
  comment { $$=build(new CommentString,$1) } |
  comment_string comment { $1->insert_tree_last($2); $$=$1 }
  ;

comment_token :
  PH_COMMENT_TOKEN { $$=new CommentToken } |
  COMMENT_SPACE { $$=build(new CommentToken,new CommentSpace) } |
  COMMENT_NEWLINE { $$=build(new CommentToken,new CommentNewline) } |
  COMMENT_TEXT { $$=build(new CommentToken,new CommentText(yylloc.text)) }   1820
  ;

comment_token_string :
  comment_token { $$=build(new CommentTokenString,$1) } |
  comment_token_string comment_token { $1->insert_tree_last($2); $$=$1 }
  ;

id :
  PH_ID         { $$=new Id } |
  IDENTIFIER { $$=build(new Id,new Identifier(yylloc.text)) }               1830
  ;

id_list2 :
  id COMMA id { $$=build(new IdList2,$1,$3) } |
  id_list2 COMMA id { $1->insert_tree_last($3); $$=$1 }
  ;

%%

                                                                        1840
PSYNTAXTREE build(PSYNTAXTREE t1, PSYNTAXTREE t2, PSYNTAXTREE t3,
                  PSYNTAXTREE t4, PSYNTAXTREE t5, PSYNTAXTREE t6)
{
  t1->insert(t2,t3,t4,t5,t6);
  return t1;
}
```

Bibliography

[Aho, Sethi & Ullman 86]
Aho, A.V., Sethi, R. and Ullman, J.D.: Compilers, Principles, Techniques and Tools. Addison-Wesley, 1986.

[Allison 83]
Allison, L.: SED: Syntax-directed program editing. Software-Practice & Experience, Vol. 13, No. 5, pp. 453-465, 1983.

[Archer, Conway 81]
Archer, J. and Conway R.: COPE: A Cooperative Programming Environment. Cornell University TR 81-459, June 1981.

[Bahlke, Snelting 85]
Bahlke, R. and Snelting, G.: The PSG - Programming System Generator. Proceedings of the ACM SIGPLAN Symposium on Language Issues in Programming Environments, Vol. 20, No. 7, 1985, pp. 28-33.

[Borras, Clément 90]
Borras, P. and Clément D.: CENTAUR: the system. Internal report, INRIA, Sophia-Antipolis and INRIA, Rocquencourt (ESPRIT Project 348). Request: centaur-request@sophia.inria.fr, 1990.

[Chomsky 56]
Chomsky, N.: Three models for the description of languages. IRE, Transactions on Information Theory, Vol. 2, No, 3, pp. 113-124, 1956.

[Coëtmeur 93a]
Coëtmeur, A.: Bison++ Reference pages for version 1.21-5, 1993.

[Coëtmeur 93b]
Coëtmeur, A.: Flex++ Reference pages for version 2.3.8, 1993.

[Clément 90]
Clément D.: GIPE: Generation of Interactive Programming Environments. Techniques et Science Informatiques, Vol. 9, no. 2, 1990.

[Donnelly, Stallman 90]
Donnelly, C. and Stallman, R.: Bison Reference for the YACC-compatible Parser Generator, ©1988, 1989, 1990 Free Software Foundation.

[Donzeau-Gouge et al. 80]
Donzeau-Gouge, V., Huet, G. Kahn, H. and Lang, B.: Programming environments based on structured editors: The Mentor expreience. INRIA Technical Report 26, May 1980.

[Ekner, Hørlyck 87]
Ekner, H. and Hørlyck P.: KERNEL FOR A SYNTAX-DIRECTED EDITOR SYSTEM. Master Thesis ID-E 371, Department of Computer Science, Danish Technical University, 1987.

[Ellis, Stroustrup 90]
Ellis, M.A. and Stroustrup, B.: THE ANNOTATED C++ REFERENCE MANUAL (ANSI BASE DOCUMENT). Addison-Wesley, 1990.

[Fisher et al. 84]
Fisher, C.N., Johnson, G.F., Mauney, J., Pal, A. and Stock, D.L.: THE POE LANGUAGE-BASED EDITOR PROJECT. Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, (Software Engineering Notes, Vol. 9, No. 3, 1984), (SIGPLAN Notices, Vol. 19, No. 5, 1984, pp. 21-29).

[George et al. 92]
George, C., Haff, P., Havelund, K., Haxthausen, A.E., Milne, R., Nielsen, C.B., Prehn, S. and Wagner, K.R.: THE RAISE SPECIFICATION LANGUAGE. BCS practioner series, Prentice Hall, 1992.

[GrammaTech 92]
THE SYNTHESIZER GENERATOR REFERENCE MANUAL RELEASE 4.0, GrammaTech, Inc., One Hopkins Place, Ithaca, NY 14850, U.S.A.

[Hansen 71]
Hansen, W.J.: USER ENGINEERING PRINCIPLES FOR INTERACTIVE SYSTEMS. Fall Joint Computer Conference, NV, USA, AFIP Conference Prooceedings, Vol. 39, 1971, pp. 523-532.

[Habermann, Notkin 86]
Harbermann, A.N. and Notkin, D.: GANDALF: SOFTWARE DEVELOPMENT ENVIRONMENTS. IEEE Transactions on Software Engineering, Vol. 12, No. 12, 1986, pp. 1117-1125.

[Hurvig 85]
Hurvig, H.: GENIE, A GENERIC SYNTAX DIRECTED EDITOR. Master Thesis ID-E 279, 1985, Department of Computer Science, Danish Technical University.

[Johnson 79]
Johnson, S.C.: YACC: YET ANOTHER COMPILER COMPILER. Unix Programmer's Manual, 7th Edition, Bell Telephone Laboratories, January, 1979.

[Lamport 86]
Lamport, L.: A DOCUMENT PREPARATION SYSTEM LaTeX USER'S GUIDE & REFERENCE MANUAL. Addision-Wesley Publishing Company, 1986.

[Naur 63]
Naur, P.: REVISED REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 60. Communications of the ACM, Vol. 6, No. 1, pp. 1-17, 1963.

[Paxson 90]
Paxson, V.: FLEX Reference pages for version 2.3, 1990.

[Reiss 84]
Reiss, S.P.: GRAPHICAL PROGRAM DEVELOPMENT WITH PECAN PROGRAM DEVELOPMENT SYSTEMS. Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, (Software Engineering Notes, Vol. 9, No. 3, 1984), (SIGPLAN Notices, Vol. 19, No. 5, 1984, pp. 30-37).

[Reps 83]
Reps, T.W.: GENERATING LANGUAGE-BASED ENVIRONMENTS. ACM Doctoral Dissertation Award 1983, MIT Press, 1983.

[Spork 93]
Spork, M.: C++ Dataabstraktion og object-orientret programmering. Polyteknisk Forlag, 1993.

[Symantec C++ 93]
Symantec C++ Professional, Reference Manuals. Copyright © Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014, 408/253-9600, 1993.

[TeitelBaum, Reps 89]
Teitelbaum, T. and Reps, T.: The synthesizer generator: a system for constructing language-based editors. Springer-Verlag 1989.

[Wilcox et al. 76]
Wilcox, T.R., Davis, A.M., and Tindall, M.H.: The design and implemtation of a table driven, interactive diagnostic programming system. Communications of the ACM, Vol. 19, No. 11, pp. 609-616, 1976.

[Win32]
Microsoft Win32 Programmer's Reference. Volume 1-5, Microsoft Press, 1993.