THE DESIGN AND IMPLEMENTATION OF AN EXPANDER FOR THE HIERARCHICAL REAL-TIME CONSTRAINTS OF COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

by

Süleyman Bayramoğlu

September 1991

Thesis Advisor: Valdis Berzins

Approved for public release; distribution is unlimited.
As part of developing the Execution Support System of Computer-Aided Prototyping System (CAPS), there is a need to translate and schedule prototypes of hard real-time systems whose specifications are defined in a hierarchical structure by using the Prototyping System Description Language (PSDL). We present a design and implementation of a PSDL expander in this thesis. The expander translates a PSDL prototype with an arbitrarily deep hierarchical structure into an equivalent two-level form that can be processed by the current implementations of the other CAPS tools. The design of the expander also provides for inheritance of timing constraints and static consistency checking.

To establish a convenient representation of PSDL specifications, we define an Abstract Data Type (ADT) that provides an Ada representation of PSDL to support software tools for analyzing, constructing, and translating PSDL programs. The PSDL ADT is built by using other common abstract data types, i.e., maps, sets, sequences, graphs, and stacks. The construction process of ADT itself is done by an LALR(1) parser, generated in Ada using the tools AYACC and AFLEX, a parser generator and a lexical analyzer.
Approved for public release; distribution is unlimited

THE DESIGN AND IMPLEMENTATION OF AN EXPANDER FOR THE HIERARCHICAL REAL-TIME CONSTRAINTS OF COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

by

Süleyman Bayramoğlu
Lieutenant JG, Turkish Navy
B.S., Turkish Naval Academy, 1984

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1991

Author: Süleyman Bayramoğlu

Approved By: Valdis Berzins, Thesis Advisor

Luqi, Second Reader

Robert B. McShee, Chairman,
Department of Computer Science
ABSTRACT

As part of developing the Execution Support System of Computer-Aided Prototyping System (CAPS), there is a need to translate and schedule prototypes of hard real-time systems whose specifications are defined in a hierarchical structure by using the Prototyping System Description Language (PSDL). We present a design and implementation of a PSDL expander in this thesis. The expander translates a PSDL prototype with an arbitrarily deep hierarchical structure into an equivalent two-level form that can be processed by the current implementations of the other CAPS tools. The design of the expander also provides for inheritance of timing constraints and static consistency checking.

To establish a convenient representation of PSDL specifications, we define an Abstract Data Type (ADT) that provides an Ada representation of PSDL specification. The main idea behind the PSDL ADT is forming an abstract representation of PSDL to support software tools for analyzing, constructing, and translating PSDL programs. The PSDL ADT is built by using other common abstract data types, i.e., maps, sets, sequences, graphs, and stacks. The construction process of ADT itself is done by an LALR(1) parser, generated in Ada using the tools AYACC and AFLEX, a parser generator and a lexical analyzer.
THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been tested for all cases of interest. While every effort has been made within the time available to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.
# TABLE OF CONTENTS

## I. INTRODUCTION

- STATEMENT OF THE PROBLEM ............................................. 1
- SCOPE .............................................................................. 2
- RESEARCH APPROACH ....................................................... 2
- ORGANIZATION .................................................................. 3

## II. BACKGROUND

- SOFTWARE DEVELOPMENT .................................................. 4
  1. The Classical Project Life Cycle: Waterfall Model ............ 5
  2. Prototyping Life Cycle ................................................... 6
  3. Rapid Prototyping ......................................................... 8
- THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS) .......... 10
- THE PROTOTYPING SYSTEM DESIGN LANGUAGE (PSDL) ....... 12
  1. PSDL Computational Model ............................................ 12
    a. Operators .................................................................. 13
    b. Data Streams .......................................................... 13
    c. Timing Constraints .................................................... 14
    d. Control Constraints ................................................... 15
  2. PSDL Prototype Example ................................................. 16

## III. DESIGN OF THE PSDL EXPANDER

- INTRODUCTION ................................................................. 19
- USE OF PSDL ABSTRACT DATA TYPE ................................. 21
  1. Abstract Data Types in General ..................................... 21
  2. Motivation and Benefits of PSDL ADT ............................. 21
  3. What is the Interface to the PSDL Abstract Data Type? ....... 22
- USING AYACC AND AFLEX IN PSDL ADT ......................... 26
  1. Ayacc ........................................................................ 26
  2. Aflex .......................................................................... 27
3. PSDL Parser .................................. 27
4. Known Deficiencies and Limitations of PSDL ADT ............. 29

D. DESIGN OF THE PSDL EXPANSION PROCESS .................. 29
   1. Transformation of the Graph ................................ 29
   2. Propagation of Timing Constraints ......................... 35
       a. Maximum Execution Time and Deadline (Finish Within) ...... 35
       b. Period ........................................ 35
       c. Minimum Calling Period ............................. 36
       d. Maximum Response Time ............................ 36
   3. Other Hierarchical Constraints .............................. 37

IV. IMPLEMENTATION OF THE PSDL EXPANDER .................... 39
   A. PSDL ADT .............................................. 39
   B. PSDL PARSER .......................................... 43
      1. Lexical Analyzer ..................................... 43
      2. Parser ............................................. 45
         a. Ayacc Specification File: psdl.y ..................... 45
         b. Associating Ada Types with the Grammar Symbols: type YYSType . 45
         c. Data Structures Used in the Actions .................. 46
         d. User Supplied Ada Code in the Ayacc Specifications ........... 51
         e. Ada Compilation Units Generated by Ayacc ................ 52
   C. GET OPERATION ........................................... 52
   D. EXPAND OPERATION ...................................... 52
   E. PUT OPERATION .......................................... 53
   F. INVOCATION OF THE PSDL EXPANDER ....................... 55

V. CONCLUSIONS AND RECOMMENDATIONS ............................ 57
   A. SUMMARY ................................................ 57
   B. RECOMMENDATIONS FOR FUTURE WORK ....................... 58
   C. CRITIQUE OF AYACC AND AFLEX .......................... 59

LIST OF REFERENCES ............................................. 60

BIBLIOGRAPHY .................................................. 62

APPENDIX A. PSDL GRAMMAR ..................................... 64
APPENDIX B. AFLEX SPECIFICATION FOR PSDL .................. 69
LIST OF FIGURES

Figure 2.1 The Classic Life Cycle (Waterfall Model) ..................... 5
Figure 2.2 Prototyping Life Cycle ........................................... 7
Figure 2.3 Iterative Prototype Development ................................ 8
Figure 2.4 Main Components of CAPS ....................................... 11
Figure 2.5 CAPS Advanced Rapid Prototyping Environment: ARPE ......... 11
Figure 2.6 The mcp and mrt of an operator ............................... 14
Figure 2.7 The period and deadline of an Operator ......................... 15
Figure 2.8 Scheduling Interval of an Operator .............................. 15
Figure 2.9 PSDL data-flow diagram with control constraints ............... 16
Figure 2.10 Example of an Augmented Data-flow diagram in PSDL .......... 17
Figure 3.1 The Expansion Process ......................................... 20
Figure 3.2 The Steps in the Expanding Process ............................. 20
Figure 3.3 The Abstract Representation of a PSDL_PROGRAM as a map .... 23
Figure 3.4 PSDL ADT Type Hierarchy ...................................... 23
Figure 3.5 Attributes of type Psdl_Component and type Data_Type ........ 24
Figure 3.6 Attributes of Atomic_Operator, Composite_Operator, Atomic_Type and Composite_Operator ........................................... 25
Figure 3.7 Parser Generation Process ...................................... 28
Figure 3.8 PSDL ADT Generation Process ................................... 28
Figure 3.9 Top Level of Example Prototype ................................ 30
Figure 3.10 Expanded Operator Example (level 2) .......................... 30
Figure 3.11 PSDL Code for Operator A ...................................... 31
Figure 3.12 PSDL Code for Operator B ...................................... 32
Figure 3.13 (a) Expanded Operator A (level 3), (b) Expanded Operator A (level 3) ........ 33
I. INTRODUCTION

Conceptual simplicity, tight coupling of tools, and effective support of host-target software development will characterize advanced Ada* programming environments. The demand for large, high quality systems has increased to the point where a jump in software technology is needed. Computer aided, rapid prototyping via specification and reusable components is one of the most promising solutions to this approach. A working model of such an environment is the Computer-Aided Prototyping System (CAPS), which supports rapid prototyping based on abstractions and reusable software components [Ref. 1]. CAPS has been built to help software engineers rapidly construct software prototypes of proposed software systems. It provides a methodology for constructing complex hard real-time prototypes from a data-flow graph of inter-task communications specified through a Prototyping System Description Language (PSDL).

As part of developing the Execution Support System of the Computer-Aided Prototyping System, there is a need to translate and schedule prototypes of hard real-time systems whose specifications are defined in a hierarchical structure by using Prototyping Description Language (PSDL). We present a design and implementation of a PSDL expander in this thesis. The expander translates a PSDL prototype with an arbitrarily depth hierarchical structure into an equivalent two-level form that can be processed by the current implementations of the other CAPS tools. The design of the expander also provides for inheritance of timing constraints and static consistency checking.

A. STATEMENT OF THE PROBLEM

PSDL is a partially-graphical language for specification and design of real-time systems. A PSDL prototype consists of a hierarchically structured collection of definitions for operators and types. Luqi et al. [Ref. 2] mention one of the requirements of the design of PSDL as:

* *ADA is a registered trademark of the US Government (Ada Joint Program Office)
"PSDL should support hierarchically structured prototypes, to simplify prototyping of large and complex systems. The PSDL descriptions at all levels of the designed prototype should be uniform."

The current implementation of Execution Support System within CAPS is limited to hierarchically structured PSDL specifications with, at most, two levels. There is a need for an expander that will expand hierarchical PSDL specifications with arbitrary depth into a two level specification.

Timing constraints are an essential part of specifying real-time systems [Ref 2]. In PSDL, timing constraints impose some constraints between the various levels of a hierarchical specification. The current implementation of CAPS does not guarantee that these constraints are met, and there is a need for consistency checking to pinpoint possible inconsistencies in the timing constraints between various levels. This thesis presents a partial design for such a consistency checker.

B. SCOPE

The design and implementation of an expander that will expand the hierarchical PSDL specifications with arbitrary depth into a two-level specification is the focus of this thesis.

The expander will also check the inconsistencies in the real-time constraints between the various levels of hierarchically structured PSDL specification during the expansion process.

C. RESEARCH APPROACH

To establish a convenient representation of PSDL specifications, we define an Abstract Data Type (ADT) that provides an Ada representation of PSDL specification. The main idea behind the PSDL ADT is forming an abstract representation of PSDL to support software tools for analyzing, constructing, and translating PSDL programs. The PSDL ADT is built by using other common abstract data types, i.e. maps, sets, sequences, graphs, and stacks. The construction process of ADT itself is done by an LALR(1) parser, generated in Ada using the tools AYACC and AFLEX, a parser generator and a lexical analyzer. These

---

1 LALR (Look Ahead Left Recursive) parser is one of the commonly used parsers.
tools have been developed at University of California Irvine as part of the Arcadia Project [Ref. 3, 4].

By processing the generated PSDL ADT for an input PSDL program, we transform the hierarchical structure into a two level specification, which we refer to as the expanded specification. The resulting expanded PSDL program is written into a new file to be processed by the tools in the Execution Support System.

During the expansion process of PSDL program, consistency of the timing constraints between various levels should also be checked and error messages produced as appropriate.

D. ORGANIZATION

Chapter II. provides a brief background on traditional software development methodology, development of real-time systems, and rapid prototyping methodology. It also gives an overview of the CAPS environment, its specification language PSDL, and the tools within CAPS. Chapter III. presents the design, and Chapter IV. presents implementation of the PSDL ADT and expander. Chapter V. provides the conclusions and recommendations for further research to enhance the functionality of the current design.
II. BACKGROUND

A. SOFTWARE DEVELOPMENT

The United States Department of Defense (DoD) is currently the world's largest user of computers. Each year billions of dollars are allocated for the development and maintenance of progressively more complex weapons and communications systems. These systems increasingly rely on information processing, utilizing embedded computer systems. These systems are often characterized by time periods or deadlines within which some event must occur. These are known as "hard real-time constraints". Satellite control systems, missile guidance systems and communications networks are examples of embedded systems with hard real-time constraints. Correctness and reliability of these software systems is critical. Software development of these systems is an immense task with increasingly high costs and potential for mis-development [Ref. 5].

Over the past twenty years, the technological advances in computer hardware technology have reduced the hardware costs of a total system from 85 percent to about 15 percent. In the early 1970s, studies showed that computer software alone comprised approximately 46 percent of the estimated total DoD computer costs. Of this cost, 56 percent was devoted specifically to embedded systems. In spite of the tremendous costs, most large software systems were characterized as not providing the functionality that was desired, took too long to build, cost too much time or space to use, and could not evolve to meet the user's changing needs [Ref 5].

Software engineering evolved in response to the need to design, implement, test, install and maintain more efficiently and correctly larger and more complex software systems. The term software engineering was coined in 1967 by a NATO study group, and endorsed by the 1968 NATO Software Engineering Conference [Ref. 6]. The conference concluded that software engineering should use the philosophies and paradigms of traditional engineering disciplines. Numerous methodologies have been introduced to support software engineering. The major approaches which underlie these different methodologies are the waterfall model [Ref. 7] of
development with its variants such as the spiral model [Ref. 8], and the prototyping [Ref. 9] method of development.

1. The Classical Project Life Cycle: Waterfall Model

The waterfall model describes a sequential approach to software development as shown in Figure 2.1. The requirements are completely determined before the system is designed, implemented and tested. The cost of systems developed using this model is very high. Required modifications which are realized late in the development of a system, such as during the testing phase, have a much greater impact on the cost of the system than they would have if they had been determined during the requirements analysis stage of the development. Requirements analysis may be considered the most critical stage of software development since this is when the system is defined [Ref 10].

![Figure 2.1 The Classic Life Cycle (Waterfall Model)](image)

Requirements are often incompletely or erroneously specified due to the often vast difference in the technical backgrounds of the user and the analyst. It is often the case that the user understands his application area but does not have the technical background to
communicate successfully his needs to the analyst, while the analyst is not familiar enough with the application to detect a misunderstanding between himself and the user. The successful development of a software system is strictly dependent upon this process. The analyst must understand the needs and desires of the user and the performance constraints of the intended software system in order to specify a complete and correct software system. Requirements specifications are still most widely written using the English language, which is an ambiguous and non-specific mode of communication.

Another difficulty of the classical life cycle is that communication between a software development team and the customer or the system's users is weak. Most of the time the customer does not know what he/she wants. In that case it is hard to determine the exact requirements, since the software development is also unfamiliar with the problem domain of the system. Formal specification languages are used to formalize the customer needs to a certain extent. Another disadvantage of the classical project life cycle is that a working model of the software system is not available until late in the project time span. This may cause two things: (1) A major bug undetected until the working program is reviewed can be disastrous [Ref. 11]. (2) The customer will not have an idea of what the system will look like until it is complete.

2. Prototyping Life Cycle

Large real-time systems and systems which have hard real-time constraints are not well supported by traditional software development methods because the designer of this type of system would not know if the system can be built with the timing and control constraints required until much time and effort has been spent on the implementation. A hard real-time constraint is a bound on the response time of a process which must be satisfied under all operating conditions.

To solve the problems raised in requirements analysis for large, parallel, distributed, real-time, or knowledge-based systems, current research suggests a revised software development life cycle based on rapid prototyping [Ref. 11, Ref. 13]. As a software methodology, rapid prototyping provides the user with increasingly refined systems to test and the designer with ever better user feedback between each refinement. The result is more user
involvement and ownership throughout the development/specification process, and consequently better engineered software [Ref. 14].

The prototyping method shown in Figure 2.2 has recently become popular. "It is a method for extracting, presenting, and refining a user's needs by building a working model of the ultimate system - quickly and in context" [Ref. 15]. This approach captures an initial set of needs and implements quickly those needs with the stated intent of iteratively expanding and refining them as the user's and designer's understanding of the system grows. The prototype is only to be used to model the system's requirements; it is not to be used as an operational system [Ref. 16].

![Figure 2.2 Prototyping Life Cycle](image)

To manually construct the prototype still takes too much time and can introduce many errors. Also, it may not accurately reflect the timing constraints placed on the system. What is needed is an automated way to rapidly prototype a hard real-time system which
reflects those constraints and requires minimal development time. Such a system should exploit reusable components and validate timing constraints.

If we are to produce and maintain Ada software that is reliable, affordable, and adoptable, the characteristics of Ada may not be the only important matter to consider. In addition, the characteristics of Ada software development environments may well be critical [Ref. 17].

3. Rapid Prototyping

The demand for large, high-quality systems has increased to the point where a jump in software technology is needed. Rapid prototyping is one of the most promising solutions to this problem. Rapid prototyping is particularly effective for ensuring that the requirements accurately reflect the user's real needs, increasing reliability and reducing costly requirement changes [Ref. 12].

![Diagram of Iterative Prototype Development]

Figure 2.3 Iterative Prototype Development

Figure 2.3 illustrates the iterative prototyping process, also known as "Spiral Model of Software Development". In the prototyping cycle, the system designer and the user work together at the beginning to determine the critical parts of the proposed system. Then, the
designer prepares a prototype of the system based on these critical requirements by using a prototype description language [Ref. 9]. The resulting system is presented to the user for validation. During these demonstrations, the user evaluates if the prototype behaves as it is supposed to do. If errors are found at this point, the user and the designer work together again on the specified requirements and correct them. This process continues until the user determines that the prototype successfully captures the critical aspects of the proposed system. This is the point where precision and accuracy are obtained for the proposed system. Then the designer uses the prototype as a basis for designing the production software.

Some advantages and disadvantages of iterative development methodology are listed below:

Advantages:

- There is a constant customer involvement (revising requirements).
- Software development time is greatly reduced.
- Methodology maps to reality.
- Allows use of common tools.

Disadvantages:

- Configuration control complexities.
- Managing customer enthusiasm.
- Uncertainties in contracting the iterative development.

The rapid, iterative construction of prototypes within a computer aided environment automates the prototyping method of software development and is called rapid prototyping [Ref. 18]. Rapid prototyping provides an efficient and precise means to determine the requirements for the software system, and greatly improves the likelihood that the software system developed from the requirements will be complete, correct and satisfactory to the user. The potential benefits of prototyping depend critically on the ability to modify the behavior of the prototype with less effort than required to modify the production software. Computer aided and object-based rapid prototyping provides a solution to this problem.
B. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

One of the major differences between a real-time system and a conventional system is required precision and accuracy of the application software. The response time of each individual operation may be a significant aspect of the associated requirements, especially for operations whose purpose is to maintain the state of some external system within a specified region. These response times, or deadlines, must be met or the system will fail to function, possibly with catastrophic consequences. These requirements are difficult for the user to provide and for the analysts to determine.

An integrated set of computer aided software tools, the Computer Aided Prototyping System, has been designed to support prototyping of complex real-time systems, such as control systems with hard-real-time constraints. The Computer Aided Prototyping System [Ref. 11] supports rapidly prototyping of such complex systems by using a partially graphical specification language. The designer of a software system uses a graphic editor to create a graphic representation of the proposed system. This graphic representation is used to generate part of an executable description of the proposed system, represented in the specification language. This description is then used to search for the reusable components in the software base to find the components matching the specification of the prototype [Ref. 19]. A translator is used to translate the prototype into a programming language, currently Ada. The prototype is then compiled and executed. The end user of the proposed system will evaluate the prototype's behavior against the expected behavior. If the comparison results are not satisfactory, the designer will modify the prototype and the user will evaluate the prototype again. This process will continue until the user agrees that the prototype meets the requirements.

CAPS is based on the Prototyping System Description Language (PSDL). "It was designed to serve as an executable prototyping language at the specification or design level [Ref. 12]." An overview of PSDL will be presented in the following section. The main components of CAPS are user interface, software database system, and execution support system (Figure 2.4). Figure 2.5 shows CAPS as an Advanced Rapid Prototyping Environment, and the interaction of the tools within the environment.
Figure 2.4 Main Components of CAPS

Figure 2.5 CAPS Advanced Rapid Prototyping Environment: ARPE
C. THE PROTOTYPING SYSTEM DESIGN LANGUAGE (PSDL)

PSDL is a partially graphical specification language developed for designing real-time systems, and specifically for CAPS. It is designed as a prototyping language to provide the designer with a simple way to specify the software systems [Ref. 2]. PSDL places strong emphasis on modularity, simplicity, reuse, adaptability, abstraction, and requirements tracing [Ref. 18].

A PSDL prototype consists of hierarchically structured set of definitions for **OPERATORS** and **TYPES***, containing zero or more of each. Each definition has two parts:

- **Specification part**: Defines the external interfaces of the operator or the type through a series of interface declarations, provides timing constraints, and describes functionality by using informal descriptions and axioms.

- **Implementation part**: Says what the implementation of the component is going to be, either in Ada or PSDL. Ada implementations point to Ada modules which provide the functionality required by the component's specification. PSDL implementations are data flow diagrams augmented with a set of data stream definitions and a set of control constraints.

A PSDL component can be either **atomic** or **composite**. An Atomic component represents a single module and cannot be decomposed into subcomponents. Composite components represent networks of components. The **Implementation** part of the component tells if the component is atomic or composite.

1. PSDL Computational Model

PSDL is based on a computational model containing **OPERATORS** that communicate via **DATA STREAMS**. Modularity is supported through the use of independent operators which can only gain access to other operators when they are connected via data streams.

PSDL is formally represented by the following computational model as an augmented graph by Luqi et al. [Ref. 2]:

\[ G = (V, E, T(v), C(v)) \]

* We will name them as the “psdl component” in the following chapters.
where

\[ V \text{ is a set of vertices} \]
\[ E \text{ is a set of edges} \]
\[ T(v) \text{ is the set of timing constraints for each vertex } v \]
\[ C(v) \text{ is the set of control constraints for each vertex } v \]

Each vertex represents an operator and each edge represents a data stream. The PSDL grammar is given in Appendix A.

### a. Operators

Every operator is a state machine, modeled internally by a set of state variables. Operators that do not have state variables behave like functions, i.e., they give the same response each time they are triggered. A state machine produces output whose value depends upon the input values and on internal state values representing some part of the history of the computation, whereas a function produces output whose value depends on only the current input values [Ref. 17]. Operators can be triggered either by the arrival of input data values or by periodic timing constraints, which specify the time intervals for which an operator must fire.

Operators are also either periodic or sporadic. Periodic operators fire at regular intervals of time while sporadic operators fire when there is new data on a set of input data streams.

### b. Data Streams

Data streams represent sequential data flow mechanisms which move data between operators. There are two kinds of data streams: data-flow and sampled. Data-flow streams are similar to FIFO queues with a length of one. Any value placed into the queue must be read by another operator before any other data value may be placed into the queue. Values read from the queue are removed from the queue. Sampled data streams may be considered as a single cell which may be written to or read from at any time and as often as desired. A value is on the stream until it is replaced by another value. Some values may never be read, because they are replaced before the stream is sampled. Data streams have data-flow queues if and only if they appear in a **TRIGGERED BY ALL** control constraint.
c. Timing Constraints

Timing constraints in PSDL impose an order on operator firing that is based on timing constraints:

- **Maximum Execution Time** (*met*)
- **Deadline (fw) or Maximum Response Time** (*mrt*)
- **Minimum Calling Period** (*mcp*)

Every time-critical sporadic operator has an *mrt* and *mcp* in addition to an *met*. The *met* is an upper bound on the length of time that an operator may use to complete its function.

The *mrt* defines an upper bound on the time that may elapse between the point in time at which an operator is fired to read from its input streams and the time when its write event occurs. The *mrt* applies only sporadic operators.

The *mcp* applies only to sporadic operators and represents a lower bound on the time between the arrival of one set of inputs and the arrival of another set of inputs (i.e. two successive activations of the read transitions of an operator (Figure 2.6). The *mcp* can be considered as the window of opportunity for the operator to use, and the *mrt* as the used portion of it.

![mcp and mrt diagram](image)

*Figure 2.6 The mcp and mrt of an operator*

Periodic operators are triggered by temporal events and must occur at regular time intervals. For each operator *f*, these time intervals are determined by the specified period (OPERATOR *f PERIOD t*) and deadline (OPERATOR *f FINISH WITHIN t*).

The *period* is the time interval between two successive activation times for the read transition of a periodic operator. The *period* applies only periodic operators.
The deadline \((fw)\) defines an upper bound on the occurrence time of the write transition of a periodic operator relative to the activation of its read transition. By default, the deadline is equal to the \(met\), and a static feasibility constraint requires that \(fw \geq met\) (Figure 2.7).

The difference between the activation time of a read transition and the deadline for the corresponding write transition is called the scheduling interval. The scheduling intervals of a periodic operator can be viewed as sliding windows, whose position on time axis relative to each other is fixed by the period, and whose absolute position on the time axis is fixed by the occurrence time \(t_0\) of the first read transition. This time may vary within the interval \(0\) to the period of the operator (Figure 2.8).

\[\text{Figure 2.7 The period and deadline of an Operator}\]

\[\text{Figure 2.8 Scheduling Interval of an Operator}\]

d. **Control Constraints**

The control constraints are the mechanisms which refine and adapt the behavior of PSDL operators. They specify how an operator may be fired, how exceptions may be raised, and how or when data may be placed onto an operator's output data streams by using predicate
expressions. They also control timers, which are "software stopwatches" used to control durations of states.

Triggering conditions and guarded outputs are expressed by predicates. If an input stream is guarded by a triggering condition, input data which do not satisfy the condition are read from the stream but do not fire the operator. Similarly, guarded output streams of an operator prevent the specified output data from being written into the guarded streams if the output guard conditions are not satisfied.

Synchronization between different operators in PSDL is achieved by precedence constraints. These constraints are introduced by data streams as follows:

Data-flow streams ensure that values are not read until they are written, and that a value is not overwritten before it has been read. This property ensures that transactions are not lost or repeated, and can be used to correlate data from different sources, such as processor operators operating in parallel. Sampled streams cannot guarantee that values will never be overwritten before they are read. The purpose of a sampled stream is to provide the most recent available version of data.

The precedence constraints associated with sporadic operators are implicit. Periodic operators are triggered by temporal events rather than by arrival of data values, and in certain conditions the precedence constraints can affect these timing constraints.

2. PSDL Prototype Example

The data-flow diagram in Figure 2.9 shows a fragment of a PSDL design graph with operators $A$ and $B$, and data streams $a$, $b$, $c$, $d$. The graph also indicates maximum execution times, 10 ms for operator $A$, and 20 ms for operator $B$. These timing constraints are the maximum execution times for each operator to process data they receive via the input data streams.

![Figure 2.9 PSDL data-flow diagram with control constraints](image_url)
Figure 2.10 [Ref. 20] shows a simple control system illustrating some typical features of PSDL. The example has a minimal specification part with an informal description. The implementation part contains a graph, making the operator ControlSystem a "composite" operator. The filter operator must be fired periodically, every 100 milliseconds.

```
OPERATOR ControlSystem

SPECIFICATION
  INPUT InputSwitch: BOOLEAN, SensorData: REAL
  OUTPUT ControlSignal: REAL
  STATES StateVariable: REAL INITIALLY 0.0
  DESCRIPTION {top level of a simple embedded system}

END

IMPLEMENTATION

GRAPH

OPERATOR filter PERIOD 100 ms
OPERATOR controller TRIGGERED BY ALL InputSwitch
MAXIMUM RESPONSE TIME 200 ms
MINIMUM CALLING PERIOD 200 ms

END
```

Figure 2.10 Example of an Augmented Data-flow diagram in PSDL

The controller operator is a sporadic operator, it must be fired whenever a new value for the InputSwitch arrives, and must complete execution in 200 milliseconds. The stream InputSwitch is a `data stream`, while SensorData and StateVariable are `sampled streams`. The
triggering conditions state the requirements for the controller and actuator to respond exactly once to every new value in the streams InputSwitch.
III. DESIGN OF THE PSDL EXPANDER

This chapter presents the design of the expander. To establish a convenient representation of PSDL specifications, we define a PSDL Abstract Data Type (ADT) that provides an Ada representation of a PSDL program. The PSDL ADT is built by using other common mathematical data types, like graphs, sets, maps, and sequences. The Ada specifications and implementations of those abstract data types are given in Appendices J, K, L, M, and N for reference.

A. INTRODUCTION

The main program of the expander consists of following operations:

(i) Get PSDL program (get)
(ii) Transform the multi-level PSDL file (expand)
(iii) Output expanded PSDL program (put)

In the first step the input PSDL program is read and parsed by a LALR(1) parser, constructed by using the tools ayacc and afl ex, which are Ada versions of the parser generator tools yacc and lex that are provided by UNIX. A brief overview of the tools ayacc and afl ex is given in the next section. During the parsing process PSDL operator names are mapped to operator descriptions and PSDL ADT representation of the program is created.

The second step is the expanding step; in this step the abstract representation of PSDL program in Ada is used to translate multi-level PSDL program into a two-level one. During this translation process the transformation of the PSDL graph is transformed, and the timing constraints are propagated into the new representation of the PSDL program. The diagram in Figure 3.1 shows a high level diagram of this process. We explain the design of the graph transformation and timing constraint propagation in the following sections. The implementation of the graph transformation is given in Chapter IV. The implementation of the propagation of the timing constraints is left for future research.
In the third step, the Ada representation of expanded PSDL program is written into a text file to be used by other tools in CAPS. In the output file some normalizing conventions are used. For instance all timing values are converted to and output in units of milli sec, and lists of type declarations are output in the format var1: type_name1, var2: type_name1, var3: type_name1.

The steps in the expanding process is shown in Figure 3.2.
B. USE OF PSDL ABSTRACT DATA TYPE

1. Abstract Data Types in General

An abstract data type, by definition, denotes a class of objects whose behavior is defined by a set of values, including a set of operations, constructors, selectors, and iterators. Luqi and Berzins [Ref. 17] describes the abstract data type concept as:

Abstract data types can be defined by the developer or predefined by the programming language. Each abstract data type is itself a system whose interaction interfaces consist of the associated primitive operations. Each interaction with a primitive operation involves the flow of one or more data objects across the boundary of the abstract data type, at least one of which must be an instance of that type.

An abstract data type is a class of data structures described by an external view: available services and properties of these services [Ref. 21]. In the case of the PSDLADT, these services are constructors, iterators, queries, exception definitions, and other type definitions. Using the abstract data type descriptions, we, as the users, do not care about how the implementation has been done, i.e. which data structures have been used; what is important for us is what operations it has – what it can offer to other software elements. This decouples the detailed implementation and storage representation information from program segments that use the abstract data type but have no need to know that information.

2. Motivation and Benefits of PSDL ADT

The main motivations for the PSDL ADT is to provide an Ada representation of the PSDL specifications to support building the expander and other tools within CAPS. The PSDL ADT includes operations for constructing PSDL components†, queries for basic attributes of PSDL components, and outputting the PSDL ADT as a PSDL program in a text file format (put operation), without worrying about how these operations are implemented.

---

* These services are operations, other type definitions, and exceptions, constants, etc.
† Psdl components are operators or PSDL types.
The benefits of the PSDL ADT follow:

- It provides a common input/output facility for PSDL programs for the tools within CAPS.
- It makes the interface between the various CAPS tools cleaner by hiding unnecessary implementation details.
- The whole PSDL program is treated as a single data structure, holding all attributes of PSDL specification. Since the PSDL ADT provides all necessary operations, attributes can be queried easily.
- It improves the efficiency and speed of the whole prototyping process in the CAPS, since there is no need for an external file I/O for reading the PSDL source text files.
- It provides efficient storage usage, since all the memory management issues are managed by the PSDL ADT itself.
- It provides improved exception handling and semantic checking features.

3. What is the Interface to the PSDL Abstract Data Type?

As we mentioned in the previous chapter, a PSDL program is a set of definitions of PSDL components, i.e. operators, and data types. Each component has a unique name and description which is composed of specification and implementation parts. A PSDL component definition can be represented as a function from PSDL id's to PSDL definitions. Thus, a PSDL program can mathematically be represented as a map on PSDL component names as the domain and PSDL component definitions as the range. As part of the PSDL ADT, we define a type PSDL_PROGRAM, which is a map from psdl component names to psdl component definitions, that is a dynamic collection of bindings from the PSDL component names - domain, to PSDL definitions - range. We can view the value of PSDL_PROGRAM as an unordered collection of ordered pairs consisting of component_id's and component_description's.

psdl_program {from :: component_id, to :: component_description}

A graphical representation of a PSDL_PROGRAM as a map is illustrated in Figure 3.3. PSDL_PROGRAM has all the characteristics that a map ADT carries (see [Ref. 17, App. D]), and the operations defined for maps are also valid for PSDL_PROGRAM.
In the PSDL ADT the basic data type is Psdl_Component. Instances of this type hold all the information that a PSDL component (operator or data type) carries. The component hierarchy in PSDL is represented by a type hierarchy which is illustrated in Figure 3.4. The type attributes are shown in Figures 3.5 and 3.6.
type Psdl_Component
SUPERTYPE None
ATTRIBUTES
   Name: string
   Generic: map {string, Type_Name}
   Keywords: set{string}
   Description: string
   Axioms: string

type Data_Type
SUPERTYPE Psdl_Component
ATTRIBUTES
   Model: map {string, Type_Name}
   Operations: map {id, Operator}

type Operator
SUPERTYPE Psdl_Component
ATTRIBUTES
   Input: map {string, Type_Name}
   Output: map {string, Type_Name}
   States: map {string, Type_Name}
   Initialization: map {string, expression}
   Exceptions: set{string}
   Met: millisec

Figure 3.5 Attributes of type Psdl_Component and type Data_Type
Some of the types used in the definitions of Psdl_Component and its subtypes are user-defined, and they are explained in Chapter IV. The formal and informal definitions, and an implementation of maps and sets can be found in [Ref. 17]. Some other implementations can also be found in [Ref. 22]. The map and set implementations we used are based on the ones
that are defined in [Ref. 17] with some improvements. The implementations are given in Appendices L and M.

Four basic operations needed for the PSDL ADT are the constructor operations for the type hierarchy described above. Those are:

- Make_Composite_Operator
- Make_Atomic_Operator
- Make_Composite_Type
- Make_Atomic_Type

The other operations provided with PSDL ADT are operations used for adding attributes to PsdiComponent and query operations for attributes. A set of exceptions are also defined to signal failures of run-time checks for violation of subtype constraints, and to signal some semantic errors. These operations take place in the type hierarchy, and we describe them in Chapter IV.

C. USING AYACC AND AFLEX IN PSDL ADT

We used a LALR(1) parser to parse the PSDL specification to construct the PSDL ADT. The parser is generated by using tools ayacc—a parser generator, and aflex—a lexical analyzer, Ada implementations of popular UNIX\(^1\) tools yacc [Ref. 23] and lex [Ref. 24]. Ayacc and aflex have been implemented as part of the Arcadia Environment Research at Department of Information and Computer Science, University of California, Irvine. Both of the tools generate Ada code, which in our case, provides compatibility with the other tools in CAPS that are implemented in Ada.

1. Ayacc

Ayacc generates a parser from an input of BNF style specification grammar, accompanied by a set of Ada program fragments (actions) to be executed as each grammar rule

\(^1\) UNIX is a trade mark of AT&T, Bell Lab Laboratories.
is recognized. Ayacc uses a push-down automaton to recognize any LALR(1) grammar [Ref. 3], and generates a set of Ada program units that act as a parser for the input grammar.

2. Aflex

Aflex is a lexical analyzer generating tool written in Ada designed for lexical processing of character input streams. It is a successor to the Alex [Ref. 25] tool from UCI, which was inspired by the popular UNIX tool lex and GNU flex. Aflex accepts high level rules written in regular expressions for character string matching, and generates Ada source code for a lexical analyzer, by using a finite state machine to recognize input tokens [Ref. 4]. Aflex can be used alone for simple lexical analysis, or with ayacc to generate a parser front-end, as we have done in constructing the PSDL expander.

3. PSDL Parser

The PSDL parser's primary responsibility is transforming the PSDL prototype source program into the PSDL abstract data type (described in section III.B). The parser has been constructed with ayacc and aflex. We adapted the PSDL grammar to make it suitable for ayacc input. The parser reads the PSDL program and constructs the PSDL ADT by using some auxiliary Ada packages. The top level diagram of the parser and PSDL ADT generation process are illustrated in Figure 3.7 and Figure 3.8 respectively. The implementation strategy of the parser is discussed in detail in Chapter IV.

The parser reads the PSDL program, locates any syntax errors, and if no errors are present, constructs the PSDL ADT by using the auxiliary Ada packages. In the current implementation of the parser error recovery is not implemented and the parser will abort the execution at the first error encountered. This is a reasonable design because the PSDL code will be generated by the Syntax-Directed Editor of CAPS, and this should be syntactically correct. During the PSDL ADT generation process, a limited set of semantic errors in the PSDL specification are also detected, and suitable exceptions are raised.
Figure 3.7 Parser Generation Process

Figure 3.8 PSDL ADT Generation Process
As can be seen from Figure 3.1 the parser acts as a *get* operation in the whole process. The implementation strategy of the parser and the data structures used in the parser are discussed in detail in Chapter IV.

### 4. Known Deficiencies and Limitations of PSDL ADT

In the current version of the PSDL ADT, BY REQUIREMENTS clauses are ignored. The substructure of expressions in PSDL is not represented. Extensive semantic checking of input PSDL specification is not done in parser or in the PSDL ADT, but some explicit run-time checks for violation of subtype constraints are done in the PSDL ADT.

The parser does not have an error recovery scheme, and it aborts its execution at the first syntax error in the input PSDL specification file, by giving the line number and the most recent token recognized.

### D. DESIGN OF THE PSDL EXPANSION PROCESS

This section describes a single processor design of the expansion process. The expansion of the Ada representation of the PSDL specification is done in two parts:

- Transformation of the *graph*,
- Propagation of *timing constraints*.

The next two sections describe these two models using expansion templates that illustrate typical cases of the transformations.

#### 1. Transformation of the Graph

An example of PSDL specification is shown in Figure 3.9. This represents a top-level operator (level 1) or *root* operator that decomposes into sub modules or operators. A root operator in PSDL does not have any input or output streams, but may have state variables. The implementation part represents the first decomposition or second level. Since the implementation of this operator is given as a *graph*, the operator is a *composite*. We are going to take this PSDL program as an example for our design. In this example, Operator A represents a simulation of an external system, and operator B represents a software system.
This corresponds to the context diagram of the entire system, in which represents a state variable, and \( v \) represents a data stream.

**OPERATOR Example**
**SPECIFICATION**

- **STATES** \( u: \) REAL INITIALLY 0.0
- **DESCRIPTION** [Top-level of a simple prototype, some of the structures are not shown]

**IMPLEMENTATION**
**GRAPH**

- \( u \)
- \( v \)

**CONTROL CONSTRAINTS**

- **OPERATOR A**
  - **PERIOD** 100 ms
- **OPERATOR B**
  - TRIGGERED BY ALL \( u \)
  - **MAXIMUM RESPONSE TIME** 200 ms
  - **MINIMUM CALLING PERIOD** 200 ms

Suppose that operator A and operator B have the PSDL specifications as shown in Figures 3.11 and 3.12.

Figure 3.9 Top Level of Example Prototype

Let us assume that the prototype Example is a four-level** prototype. The expanded data-flow graph of prototype Example is shown in Figure 3.10. Suppose that operator A and operator B have the PSDL specifications as shown in Figures 3.11 and 3.12.

Figure 3.10 Expanded Operator Example (level 2)

---

**The number of levels in deepest decomposition of the data-flow graph.**
OPERATOR A

SPECIFICATION
INPUT v: BOOLEAN
OUTPUT u: REAL
DESCRIPTION (this operator represents a simulation of an external system)

END

IMPLEMENTATION

GRAPH

CONTROL CONSTRAINTS
OPERATOR A1
OPERATOR A2
TRIGGERED BY ALL s1
MAXIMUM RESPONSE TIME 200 ms
MINIMUM CALLING PERIOD 200 ms
OPERATOR A3
PERIOD 50 sec
OPERATOR A4
FINISH WITHIN 200 ms

END

END

Figure 3.11 PSDL Code for Operator A
OPERATOR B
SPECIFICATION
INPUT $u$: REAL
OUTPUT $v$: BOOLEAN,
DESCRIPTION <text>
END
IMPLEMENTATION
GRAPH

CONTROL CONSTRAINTS
OPERATOR B1
OPERATOR B2
  TRIGGERED BY ALL
  MINIMUM CALLING PERIOD 200 ms
OPERATOR B3
  PERIOD 50 sec
END
END

Figure 3.12 PSDL Code for Operator B

The operators B1 and B2 are assumed to be atomic, and their PSDL code is not shown here. The expanded diagrams (level 3) of operators A and B are shown below side by side:
Figure 3.13 (a) Expanded Operator A (level 3), (b) Expanded Operator A (level 3)

Now, we assume that operator B3 also has a decomposition and has the PSDL code in Fig 3.14.

![Figure 3.14 PSDL Code for Operator B3](image-url)
This implies that operator B3 decomposes into the data-flow graph shown in Figure 3.15, and we assume that there is no further decomposition, so that the operators B31, B32 and B33 represent atomic operators.

Figure 3.15 Expanded Operator B3 (level 4)

The equivalent two-level prototype consists of the root level operator with a decomposition that is given by the expanded graph shown in Figure 3.16. The shading illustrates the derivation of the expanded graph, but it is not part of the expanded graph that is derived from the composite operators' graphs. In the final expanded graph all of the operators are atomic and their implementations are in Ada.

Figure 3.16 The expanded graph for Operator Example
2. Propagation of Timing Constraints

PSDL timing constraints impose some consistency requirements between the various levels of a hierarchical PSDL design. This section provides the design of a method to propagate these timing constraints into the two-level representation of PSDL program.

We describe each type of timing constraint associated with the hierarchy in the following subsections. Some very basic consistency checking between the timing constraints of various levels is also done, and error messages are produced as appropriate.

a. Maximum Execution Time and Deadline (*finish Within*)

The maximum execution time (*met*) is an upper bound on the length of time between the instant when an operator is executed and the instant when the execution is terminated. The deadline (*fw*) defines an upper bound on the occurrence time of the write transition of a periodic operator relative to the activation of its read transition. The maximum execution time constrains a single operator, and for a single processor execution model, the maximum execution of a composite operator is the sum of the maximum execution times of the child operators. This sum must be no larger than the deadline of the parent operator. Also the maximum execution time of the parent must be no less than the sum of the *met* of the children.

\[
\sum_{i=1}^{n} met_i \leq fw_{\text{parent}}
\]

\[
\sum_{i=1}^{n} met_i \leq met_{\text{parent}} \quad \text{where } i \geq 0, \text{ and } i_1...i_n \text{ denotes the children operators}
\]

For a multiprocessor execution model the above sums are calculated for the operators on each path of the graph.

b. Period

The *period* is the time interval between two successive activation times for the read transition of a periodic operator. The components or the children operators of a composite operator must be periodic, and assigned the same *period* as the parent operator as a default.
value if the designer did not explicitly provide periods for the children operators. This inheritance property is realized by the expanding process. The period of a composite operator is propagated to each child operator with the same value. The consistency check between the period and the met of the operator can be done at this point, and for a single processor operation, the expander should also check that met ≤ period for each operator, to allow the operator to complete its execution within the specified period.

c. Minimum Calling Period

The minimum calling period (mcp) represents a lower bound on the time between the arrival of one set of inputs and the arrival of another set of inputs. The children operators inherit the mcp from the parent composite operator if they do not have an mcp explicitly specified by the designer. So the mcp of the parent operator is propagated to the each child operator with the same value. But a static consistency check between the mcp and met must be done, and in a single processor model the relation met ≤ mcp must be satisfied by each child operator. If this condition is not satisfied an exception should be raised, and an error message produced.

d. Maximum Response Time

The maximum response time defines an upper bound on the time that may elapse between the point in time at which an operator is enabled to read from its input streams and the time when its write event occurs. The sum of mrts of operators on each path of a subgraph must be no larger than the mrt of the parent composite operator, and the met of each child operator must be no larger than the corresponding mrt, otherwise an exception is raised.

\[ \sum_{k=1}^{n} mrt_k \leq mrt_{\text{parent}} \]

\[ \text{met}_f \leq mrt_f \]

where \( k \geq 0 \), and \( k_1..k_n \) denotes the children operators on each path of the composite operator,

where \( f \) is any operator.
3. Other Hierarchical Constraints

A composite operator inherits the exceptions from the children operators, so during expansion process there is nothing to be done for propagating these properties. If there is an exception for a composite operator, that inherits from an atomic operator in the sub-graph.

Input and output guards are inherited by conjunction, as illustrated in Figure 3.17.

Control Constraints before the expansion:

\[
\text{OP A TRIGGERED IF } P(x) \\
\text{OUTPUT } y \text{ IF } Q(y)
\]

\[
\text{OP A1 TRIGGERED IF } P_1(x) \\
\text{OP A2 OUTPUT IF } Q_2(u, y) \\
\text{OP A3 OUTPUT IF } Q_3(v, y)
\]

Control Constraints before the expansion:

\[
\text{OP A1 TRIGGERED IF } P(x) \text{ AND } P_1(x) \\
\text{OP A2 OUTPUT IF } Q(y) \text{ AND } Q_2(u, y) \\
\text{OP A3 OUTPUT IF } Q(y) \text{ AND } Q_3(v, y)
\]

Figure 3.17 The Inheritance of Input and Output Guards
Input guards are propagated to all the sub-operators that read the input streams mentioned in the guard. Output guards are propagated to all the sub-operators that write the output streams mentioned in the guard.
IV. IMPLEMENTATION OF THE PSDL EXPANDER

This chapter describes the implementation of the PSDL expander and its main components, the PSDL ADT, parser, expander and the output operation. The skeleton of the main program for the expander is shown in Figure 4.1. Each line corresponds to one of the main components of the PSDL expander.

```
with Psdl_Component_Pkg, Psdl_io;
use Psdl_Component_Pkg;

procedure Expander is
    The_Psdl_Component: Psdl_Program := Empty_Psdl_Program;
begin
    Psdl_io.Get(The_Psdl_Component);
    Expand(The_Psdl_Component);
    Psdl_io.Put(The_Psdl_Component);
end Expander;
```

Figure 4.1 The Skeleton Main Program

The next four sections describe the purpose, implementation and functionality of each component. We do not describe the implementation of each single routine, rather we emphasize the implementation techniques for some "key" routines. The routines or modules that are not described in this chapter should be easy to follow with comments associated with them in the source files given in the Appendices.

A. PSDL ADT

**Purpose:**

The PSDL ADT provides an abstract representation of a PSDL program in Ada. With the operations provided by the PSDL ADT, components can be constructed and component instance attributes can be queried, changed or added.
Implementation:

The specification for the PSDL ADT is given in Appendix F as Psdl_Component_PKG. The initial version of specifications was written by Valdis Berzins. We made the modifications and enhancements to those specifications during the design and implementation of the PSDL parser. There are still some enhancements that can be done to the specifications, but they have not been done due to lack of time and are left for future work. These enhancements are described in Chapter VI.

The PSDL ADT's main type is Psdl_Component, and defined as a `private` record with discriminants to represent the PSDL component hierarchy in Ada. Information hiding and some encapsulation are provided by making Psdl_Component a `private` type. This limits access to the type to be just the operations provided by the PSDL ADT. For instance, the construction of a new instance of Psdl_Component, modifications or queries of instance attributes can only be done via the operations provided by the PSDL ADT. The main types defined in the PSDL ADT represent the components in the PSDL hierarchy (see Chapter III, Figure 3.4). The Ada declarations are shown in Figure 4.2 and the definition of Psdl_Component is shown in Figure 4.3. The user-defined types used in the definition of Psdl_Component are defined in the package Psdl_Concrete_Type_PKG (Appendix I).

```ada
type Psdl_Component (Category: Component_Type := Psdl_Operator;
                      (Granularity: Implementation_Type := Composite) is private;

subtype Operator is Psdl_Component;
subtype Data_Type is Psdl_Component;

subtype Atomic_Operator is Operator (Category => Psdl_Operator,
                                      Granularity => Atomic);
subtype Composite_Operator is Operator (Category => Psdl_Operator,
                                      Granularity => Composite);
subtype Atomic_Type is Data_Type (Category => Psdl_Operator,
                                      Granularity => Atomic);
subtype Composite_Type is Data_Type (Category => Psdl_Operator,
                                      Granularity => Composite);
```

Figure 4.2 The Main Types in PSDL ADT
Instances of each type shown in Figure 4.2 hold all the information that a corresponding PSDL component carries. Since a PSDL program is a collection of those components, the whole PSDL program is represented by a mapping from component names to component descriptions (the record PslComponent).

```pascal
type PslComponent(Category: Component_Type := Psdl_Operator;
  (Granularity: Implementation_Type := Composite) is
    record
        Name: Psl_id;
        Gen_Par: Type_Declaration;
        Keyw: Id_Set;
        Inf_Desc, Ax: Text;
        case Category is
            when Psdl_Operator =>
                Input, Output, State: Type_Declaration;
                Init: Init_Map;
                Excep: Id_Set;
                Smet: Millisecc;
            case Granularity is
                when Atomic =>
                    O_Ada_Name: Ada_Id;
                when Composite =>
                    G: Psl_Graph;
                    Str: Type_Declaration;
                    Tim: Id_Set;
                    Trig: Trigger_Map;
                    Eg: Exec_Guard_Map;
                    Og: Out_Guard_Map;
                    Et: Excep_Trigger_Map;
                    Tim_Op: Timer_Op_Map;
                    Per, Fw, Mcp, Mrt: Timing_Map;
                    Impl_Desc: Text;
            end case;
            when Psdl_Type =>
                Mdl: Type_Declaration;
                Ops: Operation_Map;
            case Granularity is
                when Atomic =>
                    T_Ada_Name: Ada_Id;
                when Composite =>
                    Data_Str: Type_Name;
            end case;
        end case;
    end record;
```

Figure 4.3 The Definition of PslComponent
We declare a pointer (an access type in Ada) to Psdl_Component to reference a psdl component, and the mapping is from component name to this pointer. The pointer type is necessary to avoid circular dependencies. The mapping is implemented as an instantiation of a generic map package by providing the necessary generic parameters. The Ada declaration of this instantiation is shown in Figure 4.4.

```ada
type Component_Ptr is access Psdl_Component;
package Psdl_Program_Pkg is new Generic_Map_Pkg (Key => Psdl_Id,
                                                   Result => Component_Ptr);

type Psdl_Program is new Psdl_Program_Pkg.Map;
-- A psdl program is an environment that binds psdl component names
-- to psdl component definitions.
-- The operations on Psdl_Program are the same as the operations on map.
```

Figure 4.4 Declaration of type PSDL_PROGRAM

The PSDL ADT uses several other auxiliary Ada packages. These are:

- **Psdl_Concrete_Type_Pkg**: This package provides the data structures and defined types used by the PSDL ADT (Appendices F and G).

- **Psdl_Graph_Pkg**: It provides an abstract data type representation of the data-flow graph portion of the PSDL program, and has a set of operations for constructing a data-flow graph and attribute queries. Specification and implementation are given in Appendices J and K.

- **Generic_Map_Package**: This is a generic mathematical map package, and carries all the typical map operations. This implementation of map is based on the formal definition by Luqi and Berzins [Ref. 17], and was enhanced by adding more features and better memory management. The package uses set as the main data structure, which is also based on the one in [Ref. 17]. This package also utilizes sets and maps in the implementation.

The operations, and exception definitions provided by the PSDL ADT are not listed here, they are self explanatory in the source code listing, which is given in Appendix G.

One of the additions that we have made to the PSDL ADT is the output operation put used in the main program, that outputs the expanded PSDL program by extracting from the PSDL ADT, into a text file for further use by other tools within CAPS. Although this operation is embedded into the PSDL ADT, it is worthwhile to devote a whole section to describe it due to the
complexity of its functionality. The implementation of the output operation put is described in Section D of this Chapter.

B. PSDL PARSER

Purpose:

To implement the get operation for the PSDL expander, and to construct the abstract representation of the PSDL program in Ada by using the PSDL ADT. In other words, the PSDL parser and the PSDL ADT comprise the get operation for the PSDL expander. The parser reads in the PSDL source program from a text file, and builds an instance of type PSDL_PROGRAM representing the whole PSDL program as an Ada object.

Implementation:

We generated the parser by using the tools ayacc and aflex, a parser generator and a lexical analyzer. The detail of the tools and how they are used to generate a parser can be found in [Ref. 3 and Ref. 4]. The parser generated by ayacc is an LALR(1) parser. For the characteristics of LALR(1) parsers and their constructions refer to [Ref. 5 and Ref. 6].

The PSDL parser or get operation has two basic parts, which are explained in the next two sections:

- Lexical analyzer
- Parser

1. Lexical Analyzer

The Lexical analyzer is written in aflex. Aflex generates a file containing a lexical analyzer function (YYlex) along with two auxiliary packages. Since our purpose was to generate a parser, we implemented the lexical analyzer as an Ada package (package Psdl_Lex in file psdl_lex.a, given in Appendix R), containing the lexical analyzer function YYlex which is called by the parser function YYParse. The file psdlLex (Appendix B) is the input to aflex, and defines the lexical classes and the regular expressions used in the PSDL grammar.

* LookAhead Left Recursive parser that can look ahead one token.
Each regular expression has an associated action, written in Ada, which is executed when the regular expression is matched. Each call (by the parser procedure `YYParse`) to `YYlex` returns a single token. The type `Token` is an enumeration type defined in a package called `Psdl_Tokens` (Appendix X), that is generated by `ayacc` from the token declarations part of the `ayacc` specification file.

The auxiliary packages include `Psdl_Lex_Dfa` and `Psdl_Lex_Io` packages. The package `Psdl_Lex_Dfa` contains functions and variables that are externally visible from the scanner. One of the most frequently used ones in our implementation is `YYText`, which returns a textual string representation of the matched input token in type `string`. We used this function extensively in the actions of the parser to get the string value of the tokens recognized. One of the problems that we encountered was, in the case when the input token is a literal (string, integer or real literal), or an identifier, `YYText` sometimes returns the string value of the lookahead token. To work around this problem (as it is suggested by John Self, the author of the tool), we declared one global variable for each type of token we mentioned above, and assigned the value returned by `YYText` as soon as the token is recognized, and we used those global variables, in the `ayacc` actions instead of `YYText` when needed. This works except when two identifiers come after another. To compensate for this special case, we had to declare two global variables of type `Psdl_Id` in the user declarations part of the `aflex` specification: one representing the most recently scanned identifier, and the other the previously scanned identifier. This special case arises in the production for `type_name`. A reference to the previous identifier is needed in the case where there are two consecutive type declarations after keyword "generic" in a `psdl` type specification part of the rules. The package `Psdl_Lex_Dfa` also contains another frequently used function `YYLength` which returns the length of the string representation of the matched token.

The package `Psdl_Lex_Io` contains routines which allow `yylex` to scan the input source file. These are described in [Ref. 3].

We added two procedures in the package `Psdl_Lex` by putting them in the “user defined” section of the `aflex` specification file `psdl_lex.1` and the generated file `psdl_lex.a`. These are `Linenum` and `Myecho`. `Linenum` keeps track of the number of lines in the input file, using the global variable `lines` - type `positive`, and used for giving the location of the syntax errors.
Myecho writes the textual string representation of each matched token into a text file by appending the line numbers at the beginning of each line. This file is named as `<input-file>.lst`, and is used to provide a listing file for the input PSDL source file.

2. Parser

The parser is written in ayacc, a parser generator tool. Ayacc constructs a parser which recognizes a language specified by an LR(1) grammar. The main parser procedure `YYParse` makes a call to lexical analyzer function `YYLex` to get an input token, and then matches the grammar rules and executes the actions associated with these grammar rules. Although it is simple we will not explain how the parser works (see [Ref. 4]), since it is not our concern, instead we will concentrate on the semantic actions for the rules in the input specification file.

a. Ayacc Specification File: `psdl.y`

This file is a collection of grammar rules and actions associated with them, along with the Ada subprograms we provided to be used in the semantic actions. A detailed description of the ayacc specification file in general can be found in [Ref. 4]. The following sections explain the most important aspects in the specification file. The specification file is given in Appendix C.

b. Associating Ada Types with the Grammar Symbols: type `YYSType`

Ayacc provides a way to associate an Ada data type with nonterminals and tokens. The data type is defined by associating an Ada type declaration with the identifier `YYSType`. Once this type is defined, actions can access the values associated with the grammar symbols. This declaration appears in the tokens section of the ayacc specification file.

We declared `YYSType` as a record with discriminants. This provides a way to use pseudo-variable notation ($$) to denote the values associated with non-terminal and token symbols. This makes possible use of ayacc's internal stack to associate actions that are attached to the grammar rules with the tokens of different type when they are recognized. The declaration of `YYSType` is shown in Figure 4.5. The types used here are defined in the package `Psdl_Concrete_Type`. 

45
type TOKENCATEGORYTYPE is (INTEGERLITERAL,
  PSDL_ID_STRING,
  EXPRESSION_STRING,
  TYPE_NAME_STRING,
  TYPE_DECLARATION_STRING,
  TIME_STRING,
  TIMER_OP_ID_STRING,
  NO_VALUE);

type YYStype (Token_Category:TOKENCATEGORYTYPE := NO_VALUE) is record
  case Token_Category is
    when INTEGERLITERAL =>
      Integer_Value: INTEGER;
    when PSDL_ID_STRING =>
      Psdl_Id_Value: Psdl_Id;
    when TYPE_NAME_STRING =>
      Type_Name_Value: Type_Name;
    when TYPE_DECLARATION_STRING =>
      Type_Declaration_Value: Type_Declaration;
    when EXPRESSION_STRING =>
      Expression_Value: Expression;
    when TIME_STRING =>
      Time_Value: Millisec;
    when TIMER_OP_ID_STRING =>
      Timer_Op_Id_Value: Timer_Op_Id;
    when NO.VALUE =>
      White_Space: Text := Empty_Text;
  end case;
end record;

Figure 4.5 The Declaration of YYStype

c. Data Structures Used in the Actions

We declared one global variable corresponding to each field in the PSDL_Component record, to hold their values until a call is made to constructing operation in the PSDL ADT. After this call is made, we reset their values back to their default values as specified in the PSDL ADT.
We also used several data structures and abstract data types to store the aggregate values temporarily. These are:

- sets,
- sequences,
- stacks

We used sets when we needed temporary storage to hold the tokens read but the order of those tokens is not important. For instance, in Figure 4.6 (where a fragment of PSDL code and corresponding `ayacc` specification is shown), the order of IDENTIFIERs is not important,

```plaintext
... CONTROL CONSTRAINTS
OPERATOR navigation_system
    OUTPUT CPA, bearing, track_id, datum IF range < 5000
...

constraint_options
    :constraint_options OUTPUT_TOKEN
    {
        The_Id_Set := Empty_Id_Set;
        The_Expression_String := Expression(A_Strings.Empty);
        The_Output_Id.Op := The_Operator_Name;
    }

id_list IF_TOKEN
    {The_Expression_String := Expression(A_Strings.Empty);}
expression reqmts_trace
    {
        declare
        procedure Loop_Body(Id : PSDL_Id) is
            begin
                The_Output_Id.Stream := Id;
                Bind_Out_Guard(The_Output_Id, The_Expression_String,
                                The_Out_Guard);
            end Loop_Body;
        procedure Execute_Loop is
            new Id_Set_Pkg.Generic_Scan(Loop_BODY);
            begin
                Execute_Loop(The_Id_Set);
            end;
    }

Figure 4.6 The Use of sets in the Semantic Actions
```

so we add each IDENTIFIER in a set (this is done in the production `id_list`), and when we are done reading we process each member of the set. In this case the sets are used to avoid the need
for lookahead or multiple passes. In Figure 4.6, we have to bind each IDENTIFIER in the set to an expression that is not known at the time the IDENTIFIER is scanned, because the expression occurs later in the input files. This technique is known as "back patching" in compiler design. The Ada code for a generic set is given in Appendix L.

When the order of the tokens read is important for later processing we use sequences (defined in Appendix N) for temporary storage. A similar example to the one we gave for set case, is given in Figure 4.7. Here, the order of state declarations is important because the initialization of the states are given in an order corresponding to the order of declarations,

```ada
OPERATOR weapons_interface
SPECIFICATION
... STATES
  ciws_status,
  gun_status,
  sonar_status,
  ecm_status : weapons_status_type INITIALLY ready, loaded, ready, passive
... END

attribute :
  ... STATES_TOKEN
  |
  Type_Decl_Stack_Pkg.Push (The_Type_Decl_Stack,
    Empty_Type_Declaration);
  Id_Seq_Pkg.Empty(The_Id_Seq);
  }
list_of_type_decl
  |
  Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,The_State);
  The_Init_Map_Id_Seq := The_Id_Seq;
  }
INITIALLY_TOKEN
  |
  Init_Exp_Seq_Stack_Pkg.Push(The_Init_Exp_Seq_Stack,
    Empty_Exp_Seq);
  The_Expression_String := Expression(A_Strings.Empty);
  }
initial_expression_list
  |
  Init_Exp_Seq_Stack_Pkg.Pop(The_Init_Exp_Seq_Stack, The_Init_Expr_Seq);
  Bind_Initial_State(The_State,The_Init_Expr_Seq,
     The_Initial_Expression);
  }

Figure 4.7 The Use of sequences in the Semantic Actions
```

and at the time we read the state declarations, the initializations are not known. So we need to hold these declarations in a buffer in the order that they are read. We use the same
technique for the initial_expression. When the whole rule is parsed, we do the binding of each state to the corresponding initial_expression.

Another data structure we used frequently in the parser is the stack, one of the most essential data structures in every compiler, operating system, editor, and many other applications. The Ada code for a generic stack is given in Appendix O. The need for using a stack arises when there are nested read and write operations, (i.e, when there is a set of read and write operations and between a write and the corresponding read, as it is shown in Figure 4.8).

This technique is especially convenient when there are recursive rules in the grammar. The parser uses a stack to hold or to stack the input tokens for later use. Initially the stack is empty, and we push the first “object” that needs to be held onto the stack, then we if need to “hold” some other objects before the first object is processed, we push and pop them. After each pair of push-pop operation the content of the stack becomes the same as it was before the push.

Let us now illustrate the above thought with a typical structure in the PSDL grammar. One good example is the evaluation of the initial_expression as a string that we used in Figure 4.7 for state initialization. Figure 4.9 shows a fragment of ayacc specification and corresponding PSDL source lines. In this example, we have an expression of the familiar type, grouped and nested using left and right parentheses. The expression inside first pair of parentheses is another initial_expression_list, and should be parsed by the corresponding rule again. If we do not save the contents of the previous sequence (TN.On in the sample input file at the top of Figure 4.9), it will be overwritten by the next value generated by a nested sub-expression (wp1 in Figure 4.9). To work around this problem, we use a temporary sequence, and put the value of the expression in this sequence, and push the sequence onto the stack.
Initial_expression_list

initial_expression_list `'`,_ initial_expression
{
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
                       Temp_Init_Expr_Seq);
  Exp_Seq_Pkg.Add($4.Expression_Value,Temp_Init_Expr_Seq);
  Init_Exp_Seq_Pkg.Push (The_Init_Exp_Seq_Stack,
                          Temp_Init_Expr_Seq);
}

initial_expression

| type_name `'`,_ IDENTIFIER
|
| The_Expression_String := The_Expression_String & "," & Expression(The_Id_Token);
| $\$ := (Token_Category => Expression_String,
          Expression_Value => The_Expression_String);
|
| type_name `'`,_ IDENTIFIER
|
| $\$ := (Token_Category => Expression_String,
          Expression_Value => The_Expression_String & "," & Expression(The_Id_Token));
|
| `'`
| Init_Exp_Seq_Pkg.Push (The_Init_Exp_Seq_Pkg, Empty_Exp_Seq);
|
| initial_expression_list `'`
| Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
                       Temp_Init_Expr_Seq);
| The_Expression_String := Expression($Strings.Empty);
| for i in 1 .. Exp_Seq_Pkg.Length(Temp_Init_Expr_Seq) loop
|   if i > 1 then
|     The_Expression_String := The_Expression_String & ",";
|   end if;
|   The_Expression_String := The_Expression_String &
|   Exp_Seq_Pkg.Fetch(Temp_Init_Expr_Seq, i);
| end loop;
| Exp_Seq_Pkg.Recycle (Temp_Init_Expr_Seq); -- throw it away
| $\$ := (Token_Category => Expression_String,
          Expression_Value => $4.Expression_Value & "(" & The_Expression_String & ")");

Figure 4.9 The Use of stacks for Evaluating the String Value of Expressions
When we evaluate the expression in the first pair of parentheses, we use the sequence at the top of the stack and add new expression to the content of the sequence. We assign the content of the sequence to the value of this production ($$) to be used by the parent rule, and we reclaim the heap space used by the temporary sequence. The evaluation of the expression in the second (more deeply nested) pair of parentheses is done in the same way.

In addition to the data structures we mentioned above, we made use of the internal stack provided by ayacc to evaluate the productions. In the cases similar to the one above, the internal stack is not sufficient. As it can be seen from the specification of the example given above, the internal stack is being also used. Another typical case is the rule list_of_type_declaration, where there are multiple recursive productions. We used stacks in a similar way to evaluate these productions.

### d. User Supplied Ada Code in the Ayacc Specifications

The Ada code (package Parser) at the end of the ayacc specification file is composed of:

- Global variable declarations corresponding to each field in the record Psdl_Component, for the types defined in package Psdl_Concrete_Type_Pkg, other temporary variables.
- Generic package instantiations.
- Generic procedure renaming.
- Ada local subprograms that are used in the actions. These are simple routines used to modularize the code and to improve the readability. Their functionality is clear from the Ada code and the comments associated with them.
- procedure YYParse, a parameterless procedure declaration for the main parsing procedure with the key marker, ## in the package body. The body of YYParse is generated by ayacc, and inserted where the marker is located.
- procedure YYError, an error reporting procedure. It takes a string, defaulted to "Syntax Error", corresponding an error message, as an argument. YYError is automatically called by the parser when it detects a syntax error.
• **procedure** Get is the driver procedure of the parser, and explained in the next section.

**e. Ada Compilation Units Generated by Ayacc**

Ayacc generates four Ada compilation units (packages) in four files, from the input specification file psdl.y. A brief description of each of these follows:

- **psdl.a**: This is the primary output of ayacc and contains the procedure **YYParse** along with the Ada code we provided in the “optional user declarations” section of psdl.y. The file psdl.a is given in Appendix U.

- **psdl_tokens.a**: This file contains **package** Psdl_Tokens which provides the type and variable declarations needed by both the parser procedure **YYParse** and lexical analyzer function **YYLex**. This package is extracted from the “declarations” section of the ayacc specification file, and provides a way to associate PSDL concrete types with nonterminals and tokens used in the specification file, to be able to use $$ convention in the semantic actions. This type association is done via the **type** YYSType (see Chapter IV, Section B.2.a), a record with discriminants which has fields for the value of each different token that we use in the semantic actions. The package is given in file psdl_tokens.a (Appendix X).

- **psdl_shift_reduce.a** and **psdl_goto.a**: These two files contain the static parser tables used by **YYParse**, and are given in Appendices V and W.

**C. GET OPERATION**

The procedure **Get** provided in the package Parser is nothing but a driver procedure for the parser. We overloaded the standard Ada procedure name **Get**. The first **Get** procedure reads the standard input. The other **Get** procedure takes a string as the input file name. The syntax errors are displayed on the standard output with the line numbers and the string representing the most recent token read.

To provide a standard I/O package, we wrote an I/O **package** Psdl_IO. This package contains the renaming of these two procedure and a **Put** procedure that is explained in the next section. **Package** Psdl_IO is given in Appendix E.

**D. EXPAND OPERATION**

In this implementation of the expande only the implementation of transformation of the **graph** portion of the PSDL specification is done. The implementation of the propagation of the timing constraints is left for future research.
The expansion of the graph is done level by level and in three passes for each node in one level.

- Replace the node with the nodes in the sub-graph
- Connect the edges
- Connect input/output streams to the expanded graph

In the first pass, each vertex or operator at the top level data-flow graph is expanded or replaced by the vertices in its corresponding subgraph.

After the vertices replaced, in the second pass, the edges (streams) are connected (added) to those vertices. Actually the process is done at the first and second passes is nothing but replacing the vertex with the corresponding subgraph. But since, there is no such operation provided with the PSDL ADT, we have to realize this process in two passes. An enhancement can be done to the PSDL ADT to provide this operation directly.

In the third pass external interfaces to the vertices are connected (input and output streams). The problem here is to decide where the input and streams are going to be connected. This information is taken from the specification part of the composite operator that has been expanded.

The above process is repeated for each vertex in one level. After all the vertices are replaced with their corresponding sub-graphs, each vertex in the resulted expanded level is checked if it has a decomposition or if it is composite. If there are operators which are composite, then each composite operator is expanded in the same way by using the process explained above. This “level by level” expansion is done till all the levels have only atomic operators, except the top-level, which is the root operator.

**E. PUT OPERATION**

The Put operation is implemented as one of the operations in PSDL ADT. Although this operation did not exist in the original specification of the PSDL ADT written by Berzins, it is reasonable and useful to keep the I/O operations within the PSDL ADT. The other advantage is the ease of implementation. Since access to the private part is allowed only within the body of the package, each attribute of the PSDL_Component is obtained by the “dot notation” of Ada.
We implemented the `put` operation as a separate procedure of the `package PSDL_Component_Pkg`. It is composed of several nested procedures to provide a suitable solution for converting the Ada representation of the expanded PSDL program into a formatted or pretty printed PSDL source file. The body of the procedure is shown in Figure 4.10 as a pseudo-code.

```
(1) foreach [( Id: PSDL_Id; Cp: Component_Ptr) in The_PSDL_Program] loop
(2)   Component := Component_Ptr.all;   /* dereferencing the pointer */
(3)   Put_Component_Name (Component);
(4)   if Component is PSDL_Operator then
(5)     Put_Operator_Specification (Component);
(6)     Put_OperatorIMPLEMENTATION (Component);
(7)   else
(8)     Put_Type_Specification (Component);
(9)     Put_TypeIMPLEMENTATION (Component);
(10)  end if;

Figure 4.10 The Body of Put Operation
```

For the implementation of the `foreach` construct shown in Figure 4.10, the `m4` macro preprocessor of UNIX is used. Implementation of this transformation from `foreach` notation into the equivalent Ada representation is done by using a set of `m4` macros, and a generator [Ref. 17]. This provides an easy way to use the `generic_scan` procedure to scan the all pairs in the `map` representing the PSDL program. Since each pair is composed of an id and a pointer to `PSDL_Component`, the lines 2-10 in Figure 4.10 are executed for each pair.

Lines 3, 5, 6, 8, and 9 are procedure calls. Line 3, `Put_Component_Name` is easy to implement and is basically outputs the name attribute of the component with the suitable keyword `TYPE` or `OPERATOR` depending of the component's category and suitable formatting characters. The implementations of the other four procedures are not that easy, since complex data structures like `maps`, `sets`, `graphs` are involved in the Ada representation of corresponding attributes in the `PSDL_Component` record. We use the same technique to extract the elements or attributes of these data structures or abstract data types as we did with the `PSDL_Program` in the above paragraph. And we add some formatting characters to give a pretty printed look to the extracted PSDL output.
In the case of the graph attribute of the Psdl_Component we use the attribute query operations provided by the Psdl_Graph ADT, to extract the attributes of the graph.

The put operation is in file psdl_put.a and is given in Appendix H.

Like we did with the get operation, to provide a standard way for Psdl I/O, we renamed procedure Put_Psdl in package Psdl_io as procedure Put.

The output is written to standard output, unless the output is redirected to a file with switch -o and a file name at the invocation of the expander. The output file is a pretty printed legal PSDL specification ready to be processed by the other tools in CAPS.

F. INVOCATION OF THE PSDL EXPANDER

The PSDL expander is a stand-alone program and is invoked on the command line. The command syntax is:

\texttt{expander [input-file] [-h] [-o output-file]}

When no arguments are provided, the expander reads the standard input, and outputs to the standard output. If the standard output is the keyboard ^D is used to signal end of input. The input to expander can be piped through the output of another program.

The -h switch prints a short message describing the usage of the expander command.

The default output file for expander is the standard output. The switch -o with a file name directs the output to a UNIX file. If the -o switch is used the output file should have write permission if the file already exists or the directory should be "writable". Otherwise expander will abort with an error message:

\texttt{Error: can't create output file. Permission denied.}

Each time the expander is invoked a listing of the input file is created in the directory that the input file exists or if the input is standard input, in the current working directory when the expander is invoked. The name of the listing file will be stdin.pSDL.lst for the standard input, or a pipe. If the input file is specified on the command line, then the name of
the listing file will be the concatenation of the name of the input file and ".lst". If the directory is not "writable" then expander will abort with an error message like the following:

Error: can't create listing file. Permission denied.
V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

This thesis research has contributed towards the development of a “better” CAPS environment by providing a tool that can support hierarchically structured PSDL prototypes, to simplify prototyping of large and complex systems.

The current implementation of the Execution Support System within CAPS is limited to hierarchically structured PSDL specifications with at most two levels. There has been a need to translate a multi-level PSDL source code into a two-level one to extend the domain of the entire system by providing a tool that can do this translation.

Our work has been the first attempt to make hierarchically structured multi-level PSDL programs available for the CAPS, and to provide a modular/top-down prototype development. We designed and implemented a PSDL expander that translates a PSDL prototype with an arbitrary depth hierarchical structure into an equivalent two-level form that can be processed by the other CAPS tools with their current implementations.

The two issues studied in expanding the multi-level PSDL source code:

• Transformation of the data-flow graph,
• Propagating the timing constraints into the new representation.

We did the design and implementation of the transformation of the data-flow graph by replacing all composite operators with their corresponding subgraphs with only atomic operators by preserving the data-flow streams.

We provided a partial design for propagating the timing constraints into the expanded form of the PSDL program. The implementation of this part the design is left for future research.

As part of our research we designed and implemented a PSDL abstract data type representing the whole PSDL program. The PSDL ADT provides an abstract representation of a PSDL program in Ada. all of the necessary operations, and all of the supporting types associ-
ated with it. The PSDL ADT makes the interface between the various CAPS tools cleaner by hiding unnecessary implementation details, thus providing a common input/output facility.

We used a LALR(1) parser to parse the PSDL specification to construct the PSDL ADT. We generated the parser by using the tools ayacc and aflex, a parser generator and a lexical analyzer developed at University of California Irvine as part of the Arcadia Project.

This research did not provide any work for expanding the PSDL specifications including DataTypes, and is recommended for a future thesis project.

B. RECOMMENDATIONS FOR FUTURE WORK

This thesis research has provided an initial design and implementation of the PSDL Expander and PSDL ADT. Further research is needed to complete full implementation of the expander, and identify the potential weaknesses. We recommend future work in the following specific areas:

- The design and implementation of an efficient method for inheritance of timing constraints and static consistency checking.
- The design and full implementation of a consistency checker that will pinpoint possible inconsistencies in the timing constraints between various levels of a PSDL program.
- Improving the capabilities of the PSDL expander by adding the ability to expand the PSDL programs containing PSDL Types.
- Enhancement of the PSDL ADT by providing more semantic checks and exceptions, adding the missing attributes (i.e., by requirements clauses) to the definition of type PsdLComponent, and more operations to access the attributes directly (for example, the existing operations are not well suited to implement the put operation as a stand-alone procedure, and because of the Put procedure was implemented as part of the PSDL ADT).
- Improvement of PSDL graph ADT by adding exception handlers and more operations. The current implementation does not provide any exception handling.
- Adding an error recovery scheme (for syntax errors) to the PSDL parser. The current implementation does not have an error recovery scheme, and the parser aborts at the first syntax errors encountered by signalling the line number and the
erroneous token read. Dain's study can be a good reference for realizing an error recovery scheme for the PSDL parser [Ref. 27].

C. CRITIQUE OF AYACC AND AFLEX

The current interface between ayacc and aflex complicates programming considerably because of the possibility that the parser may have to read a lookahead token in order to determine which production to reduce. This results in hard-to-predict behavior and considerably complicates the code in the semantic actions.

A cleaner design would allow the tokens returned by the lexical analyzer to have attributes (such as the matching but currently returned by YYtext, the current line number, or the current column number). This would require the introduction of a user defined type XXSType in the lexical scanner that is analogous to the YYSType currently provided by the parser. Currently the token type is an Ada enumeration type whose definition is generated by the tools and is beyond the user's control.

This recommendation also applies to the UNIX tools lex and yacc.
LIST OF REFERENCES


BIBLIOGRAPHY


APPENDIX A. PSDL GRAMMAR

This grammar uses standard symbology conventions. Optional items are enclosed in [ square brackets ]. Items which may appear zero or more times appear in { curly braces }. Terminal symbols appear in bold face. Groupings appear in ( parentheses ). Items contained in “double quotes” are character literals. The “|” vertical bar indicates a list of options from which no more than one item may be selected. This grammar represents the current version of the PSDL grammar as of 1 September 1991. All previous versions are obsolete.

\[
\begin{align*}
\text{start} & = \text{psdl} \\
\text{psdl} & = \{ \text{component} \} \\
\text{component} & = \text{data_type} \\
& | \text{operator} \\
\text{data_type} & = \text{type id type_spec type_impl} \\
\text{type_spec} & = \text{specification [generic type_decl] [type_decl]} \\
& \quad \text{operator id operator_spec} \\
& \quad [\text{functionality}] \text{ end} \\
\text{operator} & = \text{operator id operator_spec operator_impl} \\
\text{operator_spec} & = \text{specification [interface] [functionality] end} \\
\text{interface} & = \text{attribute [reqmts_trace]} \\
\text{attribute} & = \text{generic type_decl} \\
& \quad | \text{input type_decl} \\
& \quad | \text{output type_decl} \\
& \quad | \text{states type_decl initially initial_expression_list}
\end{align*}
\]
exceptions id_list
max execution time time

type decl
  = id_list ":=" type_name ":, id_list ":=" type_name

type_name
  = id
  | id "[" type_decl "]

id_list
  = id ["," id]

reqmts_trace
  = by requirements id_list

functionality
  = [keywords] [informal_desc] [formal_desc]

keywords
  = keywords id_list

informal_desc
  = description "(" text ")"

formal_desc
  = axioms "(" text ")"

type_impl
  = implementation ada id end
  | implementation type_name (operator id operator_impl) end

operator_impl
  = implementation ada id end
  | implementation psdl_impl end

psdl_impl
  = data_flow_diagram [streams] [timers] [control_constraints]
    [informal_desc]

data_flow_diagram
  = graph (vertex) (edge)

vertex
  = vertex op_id [":" time]
  -- time is the maximum execution time
edge = edge id [";" time] op_id ["->" op_id
-- time is the latency

op_id = id ["(" [id_list] ["!" [id_list] ")]"

streams = data stream type_decl

timers = timer id_list

control_constraints = control constraints constraint (constraint)

constraint = operator op_id
[triggered [trigger] [if expression] [reqmts_trace]]
[period time [reqmts_trace]]
[finish within time [reqmts_trace]]
[minimunm calling period time [reqmts_trace]]
[maximum response time time [reqmts_trace]]
[constraint_options]

constraint_options = output id_list [if expression] [reqmts_trace]
| exception id [if expression] [reqmts_trace]
| timer_op id [if expression] [reqmts_trace]

trigger = by all id_list
| by some id_list

timer_op = reset timer
| start timer
| stop timer

initial_expression_list = initial_expression ["," initial_expression]

initial_expression = true
| false
| integer_literal
I real_literal
| string_literal
| id
| type_name "." id ["(" initial_expression_list ")"]
| "(" initial_expression ")"
| initial_expression binary_op initial_expression
| unary_op initial_expression

binary_op
= and | or | xor
| "<" | ">" | "=" | ">=" | "<=" | "/=
| "+" | "-" | "&" | "&&" | "/" | mod | rem | "**"

unary_op
= not | abs | "+" | "-"

time
= integer_literal unit

unit
= microsec
| ms
| sec
| min
| hours

expression_list
= expression ("," expression)

expression
= true
| false
| integer_literal
| time
| real_literal
| string_literal
| id
| type_name "." id ["(" expression_list ")"]
| "(" expression ")"
| initial_expression binary_op initial_expression
| unary_op initial_expression

id
= letter {alpha_numeric}

real_literal
= integer_literal "." integer_literal
integer_literal
  = digit (digit)

string_literal
  = """" (char) """

char
  = any printable character except ")"

digit
  = "0 .. 9"

letter
  = "a .. z"
  | "A .. Z"
  | " "

alpha_numberic
  = letter
  | digit

text
  = {char}
APPENDIX B. AFLEX SPECIFICATION FOR PSDL

---:-------------------:
-- psdl_lex.l
--:-------------------:

-- Unit name : Aflex specification file for PSDL parser
-- File name : psdl_lex.l
-- Author : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Date Created : May 1991
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1,
-- Aflex Ver. 1.1 (May 1990)

---:-------------------:
-- Keywords : lexical analyzer, parser, PSDL
--
-- Abstract :
-- This file is the Aflex input file for PSDL grammar,
-- For more information
-- refer to the file psdl_lex.prologue

------------------------ Revision history ------------------------
--
--$Source: /n/gemini/work/bayram/AYACC/parser/RCS/psdl_lex.l,v $
--$Revision: 1.13 $
--$Date: 1991/09/24 04:51:13 $
--$Author: bayram $
--

------------------------ Definitions of lexical classes ------------------------

Digit   [0-9]
Int     {Digit}+
Letter[a-zA-Z_]
Alpha   {(Letter)|(Digit)}
Blank   [ \
\]
Text    ['{}]
StrLit  ['""']\[\]\[""]
% ada|Ada|ADA
axioms|AXIOMS
by|BY|ALL
by|BY|REQU.
by|BY|SOME
control|CONTROL
constraints|CONSTRAINTS
data|DATA
stream|STREAM
description|DESCRIPTION
description|EDGE
end|END
exceptions|EXCEPTIONS
exception|EXCEPTION
finish|FINISH
within|WITHIN
generic|GENERIC
graph|GRAPH
hours|HOURS
if|IF
implementation|IMPLEMENTATION
initially|INITIALLY
input|INPUT
key|KEYWORDS
maximum|MAXIMUM
execution|EXECUTION
time|TIME
response|RESPONSE
microsec|MICROSEC
minimum|MINIMUM
calling|PERIOD
min|MINUTES
ms|MILLISECONDS
operator|OPERATOR
output|OUTPUT
period|PERIOD
reset|TIMER
specification|SPECIFICATION
start|START
states|STATES
stop|STOP
70
timer | TIMER
trigger | TRIGGERED
type | TYPE
vertex | VERTEX
"and" | "AND"
"or" | "OR"
"xor" | "XOR"
">=" | "<=
"/=" | "-="
">" | "->
"=" | "=
"+" | "+
"-" | "-
"*" | "*
"/" | "/
"%" | "%"
"(" | "的优点"
")" | "的优点"
"[]" | "的优点"
"|=" | "的优点"
">" | "的优点"
"<" | "的优点"
"mod" | "MOD"
"rem" | "REM"
"**" | "exp" | "EXP"
"abs" | "ABS"
"not" | "NOT"
true | TRUE
false | FALSE

{Letter}{Alpha}*

{Quote}{StrLit}*[Quote]
package Psdl_Lex

Lines : Positive := 1;
Num_Errors : Natural := 0;
List_File : Text_Io.File_Type;
in the case that one id comes right after another id
-- we save the previous one to get around the problem
-- that look ahead token is saved into yytext
-- This problem occurs in the optional_generic_param if
-- an optinal type declaration comes after that.
-- IDENTIFIER

The_Prev_Id_Token: Pdsl_Id := Pdsl_Id(A_Strings.Empty);
The_Id_Token : Pdsl_Id := Pdsl_Id(A_Strings.Empty);

-- STRING_LITERAL
The_String_Token : Expression := Expression(A_Strings.Empty);

-- INTEGER_LITERAL (psdl_id or expression)
The_Integer_Token: A_String := A_Strings.Empty;

-- REAL_LITERAL
The_Real_Token : Expression := Expression(A_Strings.Empty);

-- TEXT_TOKEN
The_Text_Token : Text := Empty_Text,

Last_Yylength: Integer;

-- This procedure keeps track of the line numbers in
-- the input file, by using the global variable "lines"
procedure Linenum;

-- This procedure writes the input file in a file
-- <input-file>.lst.ls- prepending the line numbers,
procedure Myecho;

-- Lexical analyzer function generated by aflex
function YYlex return Token;

end Pdsl_Lex;

package body Pdsl_Lex is
procedure Myecho is
begin
    Text_IO.Put(List_File, Psdl_Lex_Dfa.Yytext);
end Myecho;

procedure Linenum is
begin
    Text_IO.Put(List_File, Integer'Image(Lines) & ":");
    Lines := Lines + 1;
end Linenum;

end Psdl_Lex;
APPENDIX C. AYACC SPECIFICATION FOR PSDL

-- psdl.y

-- Unit name : Ayacc specification file for PSDL parser
-- File name : psdl.y
-- Author : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Date Created : May 1991
-- Last Update : (Mon Sep 23 22:59:33 1991 - bayram)
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1, Ayacc Ver. 1.0 (May 1988)

-- Keywords : parser, PSDL
--
-- Abstract :
-- This file is the ayacc input file for PSDL grammar. For more information
-- refer to the file psdl.y.prologue

--- Revision history -----------------------------
---
--- $Source: /n/gemini/work/bayram/AYACC/parser/RCS/psdl.y,v$
--- $Revision: 1.1$
--- $Author: bayram$
---
--- /* token declarations section */

$token (, ) , ',' '[' ']' ':' '. ' '\']' \\
$token ARROW

$token ARROW

$token TRUE FALSE

$token ADA_TOKEN AXIOMS_TOKEN

$token BY_ALL_TOKEN BY_REQ_TOKEN BY_SOME_TOKEN

$token CALL_PERIOD_TOKEN CONTROL_TOKEN

$token CONSTRAINTS_TOKEN
-- /* operator precedences */
-- /* left means group and evaluate from the left */
%left AND_TOKEN OR_TOKEN XOR_TOKEN LOGICAL_OPERATOR
%left '<' '>' '=' GREATER_THAN OR_EQUAL LESS_THAN OR_EQUAL INEQUALITY RELATIONAL_OPERATOR
%left '+' '-' '&' BINARY_ADDING_OPERATOR
%left UNARY_ADDING_OPERATOR
%left '::' '/' MOD_TOKEN REM_TOKEN MULTIPLYING_OPERATOR
%left EXP_TOKEN ABS_TOKEN NOT_TOKEN HIGHEST_PRECEDENCE_OPERATOR

%start start_symbol -- this is an artificial start symbol, for initialization

%with Psdl_Concrete_Type_Pkg;
%use Psdl_Concrete_Type_Pkg;

{ type TOKEN_CATEGORY_TYPE is (INTEGER_LITERAL,
PSDL_ID_STRING,
EXPRESSION_STRING,
TYPE_NAME_STRING,
TYPE_DECLARATION_STRING,
TIME_STRING,
TIMER_OP_ID_STRING,
NO_VALUE);

type YYStype (Token_Category : TOKEN_CATEGORY_TYPE := NO_VALUE) is record
  case Token_Category is
    when INTEGER_LITERAL =>
      Integer_Value : INTEGER;
    when PSDL_ID_STRING =>
      PSDL_Id_Value : PSDL_Id;
    when TYPE_NAME_STRING =>
      Type_Name_Value : Type_Name;
    when TYPE_DECLARATION_STRING =>
      Type_Declaration_Value : Type_Declaration;
    when EXPRESSION_STRING =>
      Expression_Value : Expression;
    when TIME_STRING =>

Time_Value : Millisec;

when TIMER_OP_ID_STRING =>
    Timer_Op_Id_Value : Timer_Op_Id;

when NO_VALUE =>
    White_Space : Text := Empty_Text;
end case;
end record;

--- package PSDL_Program_Pkg is
new Generic_Map_Pkg(Key => PSDL_ID, Result => COMPONENT_PTR);

type PSDL_PROGRAM is new PSDL_Program_Pkg.Map;

type Component_Ptr is access PSDL_COMPONENT;

A pSDL program is an environment that binds
pSDL component names to pSDL component definitions.
The operations on pSDL programs are the same
as the operations on maps.

start_symbol
:
    { The_Program := Empty_PSDL_Program; }

psdl
;

psdl
:
    psdl
    { the_component_ptr := new PSDL_COMPONENT; }

component
{
    --/* the created object should always be constrained */
    --/* since object is a record with discriminants. */

    The_Component_Ptr :=
        new PSDL_Component
        (Category => Component_Category(The_Component),
         Granularity => Component_Granularity(The_Component));

    The_Component_Ptr.all := The_Component;
    Bind_Program (Name(The_Component),
                  The_Component_Ptr,
                  The_Program);
}

--- empty ';'
component

data_type --/* subtype Data_Type is PSDL_COMPONENT */

operator --/* subtype Data_Type is PSDL_COMPONENT */

data_type :

TYPE_TOKEN IDENTIFIER

| $\$ := (Token_Category => Psdl_Id_String,
| Psdl_Id_Value => The_IdToken);
| The_Operation_Map := Empty_Operation_Map;
|

type_spec type_impl

| -- construct the psdl type using global variables
| -- psdl component record fields that have default values
| -- are passed as in out parameters, so that after
| -- building the component, they are initialized
| -- back to their default values.

Build_Psdl_Type($3.Psdl_Id_Value,

| The_Ada_NAME,
| The_Model,
| The_Data_Structure,
| The_Operation_Map,
| The_Type_Gen_Par,
| The_Keywords,
| The_Description,
| The_Axioms,
| Is_Atomic_Type,
| The_Component);
|

type_spec

| SPECIFICATION_TOKEN optional_generic_param optional_type_decl
op_spec_list functionality END_TOKEN
|

| --/* C.Gen_Par:Type_Declaration:=Empty_Type_Declaration */
| optional_generic_param
|
| GENERIC_TOKEN


```solution

{ Type_DecI_Stack_Pkg.Push(The_Type_Decl(Stack,
    Empty_Type_Declaration);
    Type_SPEC_Gen_Par := TRUE;
}

list_of_type_decl
{
    Type_DecI_Stack_Pkg.Pop(The_Type_Decl(Stack,
        The_Type_Gen_Par);
    Type_SPEC_Gen_Par := FALSE;
}
| --/* empty */
|

optional_type_decl
:
{
    Type_DecI_Stack_Pkg.Push(The_Type_Decl(Stack,
        Empty_Type_Declaration);
}

list_of_type_decl
{
    Type_DecI_Stack_Pkg.Pop(The_Type_Decl(Stack,
        The_Model);
}
| |
|

op_spec_list
: op_spec_list
{
    The_Op_Ptr := new Operator;
}

OPERATOR_TOKEN IDENTIFIER
{
    $5 := (Token_Category => Pdsl_Id_String,
        Pdsl_Id_Value => The_Id_Token);
    -- create a new operator(composite) to put in ops map
    -- make it composite because we don't know what
    -- the granularity is at this point.
    The_Op_Ptr := new Operator(Category => Pdsl_Operator,
        Granularity => Composite);
}

Operator_Spec
{
    Build_Pdsl_Operator($5.Pdsl_Id_Value,
        The_Ada_Name,

80
```
{| SS := (Token_Category => Psdl_Id_String, Psdl_Id_Value => The_Id_Token); } operator_spec operator_impl |
| -- construct the psdl operator
| -- using the global variables Build_Psdl_Operator($3.Psdl_Id_Value, The_Ada_Name, The_Gen_Par, The_Keywords, The_Description, The_Axioms, The_Input, The_Output,
list_of_type_decl
{
    TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack, The_Input);
}

-- /* O.Output: Type_Declaration:=Empty_Type_Declaration */
| OUTPUT_TOKEN
{
    TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack, Empty_Type_Declaration);
}

list_of_type_decl
{
    TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack, The_Output);
}

-- /* O.State: Type_Declaration:=Empty_Type_Declaration */
| STATES_TOKEN
{
    TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack, Empty_Type_Declaration);
    IdSeq_Pkg.Empty(The_ExprSeq);
    -- empty id seq, to use with init map
}

list_of_type_decl
{
    TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack, The_State);
    The_InitMap_IdSeq := The_IdSeq;
    -- hold the id's for init map.
}

-- /* O.Init: Init_Map:=Empty_Init_Map */
-- /* Init_Map is Map(Psdl_Id, Expression) */
INITIALLY_TOKEN
{
    InitExpSeqStack_Pkg.Pop(The_InitExpSeq_Stack, Empty_InitSeq);
    The_Expr := Expression(A_Strings.Empty);
}

-- /* Expression is new A_Strings.A_String */
initial_expression_list
{
    InitExpSeqStack_Pkg.Pop(The_InitExpSeq_Stack, The_InitExpSeq);
    Bind_InitialState(The_State, The_InitExpSeq,

The_Initial_Expression);

-- /* O.Excep: Id_Set:= Empty_Id_Set; */
| EXCEPTIONS_TOKEN
|   Id_Set_Pkg.Empty(The_Id_Sets);
|

id_list
|   Id_Set_Pkg.Assign(The_Exceptions, The_Id_Sets);
|

-- /* O.Smet: Millisec */
|   -- /* everything is converted into msec */
|   MAXIMUM_TOKEN EXECUTION_TOKEN TIME_TOKEN time
|   The_Specified_Met := $4.Integer_Value;
|

-- /* initialization is made by the callers of this rule */

list_of_type_decl
|   list_of_type_decl ',,’ type_decl
|   type_decl
|

type_decl
|   { The_Id_Sets := Empty_Id_Set;
|    };

type_name
|   { TypeDecl_Stack_Pkg.Pop(The_TypeDecl_Stack, 
|    Temp_TypeDecl);

   --/* Bind each id in id the id set to the type name */
   --/* in the internal stack($5), return temp_type_decl */
   Bind_Type_Declaration(
      Id_Set_Stack_Pkg.Top(The_Id_Set_Stack),
      $5.Type_Name_Value,

84
Temp_TypeDecl;

TypeDeclStack_Pkg.Push(The_TypeDecl_Stack, Temp_TypeDecl);

/* pop the stack after bind */
IdSetStack_Pkg.Pop(TheIdSetStack);

;

type_name:
IDENTIFIER
{
  $$ := (Token_Category => PdslIdString,
         PdslIdValue => TheIdToken);

  The_Expression_String := The_Expression_String & " " & Expression(TheIdToken);
}

'{'

  TypeDeclStack_Pkg.Push(The_TypeDecl_Stack, Empty_TypeDeclaration);
  The_Expression_String := The_Expression_String & " [
"
}

list_of_type_decl
{

  The_Type_Name := New Type_Name_Record;
  The_Type_Name.Name := $2.PdslIdValue;
  The_Type_Name.Gen_Par := TypeDeclStack_Pkg.Top(The_TypeDecl_Stack);
  $$ := (Token_Category => TypeNameString,
         TypeNameValue => The_Type_Name);
  TypeDeclStack_Pkg.Pop(The_TypeDecl_Stack);

}

'}'

[ The_Expression_String := The_Expression_String & " ] " ]

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- this an awkward way of working around the -- problem we get when we have two identifiers -- one after another</td>
</tr>
<tr>
<td>if Type_Spec.Gen_Par and</td>
</tr>
<tr>
<td>not IdSet_Pkg.Member(ThePrev_Id_Token, The_Id_Set) then</td>
</tr>
<tr>
<td>The_Type_Name :=</td>
</tr>
<tr>
<td>New Type_Name_Record(ThePrev_Id_Token,</td>
</tr>
</tbody>
</table>
The Expression String := The Expression String & " " & Expression(The_Prev_Id_Token);
else
  The_Type_Name :=
    New_Type_Name_Record'(The_Id_Token, Empty_Type_Declaration);
The(Expression_String := The Expression String & " " & Expression(The_Id_Token);
end if;

$$_ := (Token_Category => Type_Name_String, Type_Name_Value => The_Type_Name);
}

id_list
  : id_list ','
  { The Expression String := The Expression String & ", " ;}
IDENTIFIER
  { Id_Set_Pkg.Add(The_Id_Token, The_Id_Set);
The.String := The_String & "," & The_Id_Token;
Id_Seq_Pkg.Add(The_Id_Token, The_Id_Seq);
The Expression String := The Expression String & " " & Expression(The_Id_Token);}
| IDENTIFIER
  { Id_Set_Pkg.Add(The_Id_Token, The_Id_Set);
The.String := The_Id_Token;
Id_Seq_Pkg.Add(The_Id_Token, The_Id_Seq);
The Expression String := The Expression String & " " & Expression(The_Id_Token);}

reqmts_trace -- Ignored In This Version
  : BY_REQ_TOKEN id_list
  |

functionality
  : keywords informal_desc formal_desc
  ;
IdSetPkg.Empty(TheIdSet);

id_list
{  
  IdSetPkg.Assign(TheKeywords, The_id_set);
  
  TheKeywords := Empty_Id_Set;
}

TheKeywords := Empty_Id_Set;

TheDescription := TheText_Token;
TheImplDesc := TheText_Token;

TheAxioms := TheText_Token;

IsAtomicType := True;
TheAdaName := Ada_Id(TheIdToken);

END_TOKEN

IsAtomicType := False;
TheDataStructure := $2.Type_Name_Value;

op_impl_list END TOKEN
;

END_TOKEN


The_Op_Ptr := New Operator;

OPERATOR_TOKEN IDENTIFIER
{
  $$ := (Token_Category => Psdl_Id_String,
          Psdl_Id_Value => The_IdToken);
}

operator_impl
{
  -- add implementation part to the operator in the operation map
  Add_Op_Impl_To_Op_Map($$.Psdl_Id_Value,
                          The_Ada_Name,
                          IsAtomicOperator,
                          TheOperation_Map,
                          The_Graph,
                          The_Streams,
                          The_Timers,
                          The_Trigger,
                          The_Exec_Guard,
                          The_Out_Guard,
                          The_ExpTrigger,
                          The_Timer_Op,
                          The_Per,
                          The_Fw,
                          The_Mcp,
                          The_Mrt,
                          The_Impl_Desc );

  }

operator_impl :
  IMPLEMENTATION_TOKEN ADA_TOKEN IDENTIFIER
  {  
    IsAtomicOperator := True; 
    The_Ada_Name := Ada_Id(The_IdToken); 
  }

END_TOKEN

| IMPLEMENTATION_TOKEN psdl_impl 
  {  
    IsAtomicOperator := False; 
  }

END_TOKEN

psdl_impl
  : data_flow_diagram streams timers control_constraints
    {  
      The_Impl_Desc := Empty_Text; 
    }
informal_desc
;

data_flow_diagram
;
{ The_Graph := Empty_Psdl_Graph; }

GRAPH_TOKEN vertex_list edge_list
;

-- /* Time Is The Maximum Execution Time */
vertex_list : vertex_list VERTEX_TOKEN op_id optional_time
{ The_Graph := Psdl_Graph_Pkg.Add_Vertex($3.Psdl_Id_Value, The_Graph, $4.Integer_Value);
}
|--/* empty */
;

-- /* Time Is The Latency */
edge_list : edge_list EDGE_TOKEN IDENTIFIER
{ The_Edge_Name := The_Id_TOKEN; }

optional_time op_id ARROW op_id
{ The_Graph := Psdl_Graph_Pkg.Add_Edge($6.Psdl_Id_Value, $8.Psdl_Id_Value, The_Edge_Name, The_Graph, $5.Integer_Value);
}

;
op_id : IDENTIFIER
{ $$ := (Token_Category => Psdl_Id_String, Psdl_Id_Value => The_Id_TOKEN);
}

opt_arg
{ $$ := { Token_Category => Psdl_Id_String, Psdl_Id_Value => $2.Psdl_Id_Value 
& $3.Psdl_Id_Value }; 
}
opt_arg

  : { The_String := Psdl_Id(A_Strings.Empty); }

'(' optional_id_list

  { $S := ( Token_Category => Psdl_Id_String,
    Psdl_Id_Value => "{ The_String};
    The_String := Psdl_Id(A_Strings.Empty); }

')' optional_id_list ')'

  { $S := ( Token_Category => Psdl_Id_String,
    Psdl_Id_Value => $4.Psdl_Id_Value
    & "|" & The_String & ")" );

  )

optional_id_list

  : id_list

  |

optional_time

  : ':' time

    { $S := (Token_Category => Integer_Literal,
      Integer_Value => $2.Integer_Value); }

  |

    { $S := (Token_Category => Integer_Literal,
      Integer_Value => 0); }

  ;

streams

  : DATA_TOKEN STREAM_TOKEN

    { Type_Decl_Stack_Pkg.Push(The_Type_Decl_Stack,
      Empty_Type_Declaration); }

  |

list_of_type_decl

  {
Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack, The_Streams);

|--/* The order of id’s is not important, so */
|--/* we use Id_Set as the data structure */
|--/* to store the timers. */

{-


timers:

  TIMER_TOKEN
  {
    Id_Set_Pkg.Empty(The_Id_Set);
  }

  id_list
  {
    Id_Set_Pkg.Assign(The_Timers, The_Id_Set);
  }

  {
    Id_Set_Pkg.Assign(The_Timers, Empty_Id_Set);
  }

};

control_constraints:

  CONTROL_TOKEN: CONSTRAINTS_TOKEN
  {
    The_Operator_Name := The_IdToken;
    The_Trigger := Empty_Trigger_Map;
    The_Per := Empty_Timing_Map;
    The_Fw := Empty_Timing_Map;
    The_Mcp := Empty_Timing_Map;
    The_Mrt := Empty_Timing_Map;
    The_Exec_Guard := Empty_Exec_Guard_Map;
    The_Out_Guard := Empty_Out_Guard_Map;
    The_Excep_Trigger := Empty_Excep_Trigger_Map;
    The_Timer_Op := Empty_Timer_Op_Map;
  }

};

constraints:

  constraints OPERATOR_TOKEN IDENTIFIER
  {
    The_Operator_Name := The_IdToken;
  }

Opt_Trigger Opt_Period Opt_Finish_Within
Opt_Mcp Opt_Mrt Constraint_Options
constraint_options
: constraint_options OUTPUT_TOKEN
{
   The_Id_Set := Empty_Id_Set;
   The_Expression_String := Expression(A.Strings.Empty);
   The_Output_Id.Op := The_Operator_Name;
}

id_list IF_TOKEN
{
   The_Expression_String := Expression(A.Strings.Empty);
}

expression reqmts_trace
{
   -- Begin Expansion Of Foreach Loop Macro.
   declare
      procedure Loop_Body(Id : PSDL_Id) is
      begin
         The_Output_Id.Stream := Id;
         Bind_Out_Guard(The_Output_Id,__
            The_Expression_String, __
            The_Out_Guard);
      end Loop_Body;
      procedure Execute_Loop is
         new Id_Set_Pkg.Generic_Scan(Loop_Body);
      begin
         Execute_Loop(The_Id_Set);
      end;
   }

constraint_options EXCEPTION_TOKEN IDENTIFIER
{
   $S := (Token_Category => PSDL_Id_String,
          PSDL_Id_Value => The_Id_token);
   The_Expression_String := Expression(A.Strings.Empty);
}

opt_if Predicate reqmts_trace
{
   The_Except_Id.Op := The_Operator_Name;
}
The_Except_Id.Excep := $4.Psdl_Id_Value;
Bind_Except_Trigger(The_Except_Id,
The_Expression_String,
The_Except_Trigger);
}

| constraint_options timer_op IDENTIFIER
|
| $5 := (Token_Category => Psdl_Id_String,
| Psdl_Id_Value => The_Id_Token);
The_Expression_String := Expression(A_Strings.Empty);
}

opt_if_predicate reqmts_trace
|
| The_Timer_Op_Record.Op_Id := $2.Timer_Op_Id_Value;
The_Timer_Op_Record.Timer_Id := $4.Psdl_Id_Value;
The_Timer_Op_Record.Guard := The_Expression_String;

| Timer_Op_Set_Pkg.Add(The_Timer_Op_Record,
The_Timer_Op_Set);
Bind_Timer_Op(The_Operator_Name,
The_Timer_Op_Set,
The_Timer_Op);
|
|

opt_trigger
|
| : TRIGGERED_TOKEN trigger
|
| The_Expression_String := Expression(A_Strings.Empty);
|

opt_if_predicate reqmts_trace
|
| Bind_Exec_Guard(The_Operator_Name,
The_Expression_String,
The_Exec_Guard);
|
|

trigger
|
| : BY_ALL_TOKEN
|
| The_Id_Set := Empty_Id_Set;
|

id_list
|
| The_Trigger_Record.Idt := By_All;
The_Trigger_Record.Streams := The_Id_Set;
Bind_Trigger(The_Operator_Name,
The_Trigger_Record,
The_Trigger);
}

<table>
<thead>
<tr>
<th>BY_SOME_TOKEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>The_Id_Set := Empty_Id_Set;</td>
</tr>
</tbody>
</table>

```plaintext
id_list
{
    The_Trigger_Record.Tt := By_Some;
The_Trigger_Record.Streams := The_Id_Set;
    Bind_Trigger(The_Operator_Name,
The_Trigger_Record,
The_Trigger);
}
```

```plaintext
-- we don't care what is in the id set
The_Trigger_Record.Tt := None;
The_Trigger_Record.Streams := The_Id_Set;
    Bind_Trigger(The_Operator_Name,
The_Trigger_Record,
The_Trigger);
```

```plaintext
opt_period
: PERIOD_TOKEN Time Reqmts_Trace
{
    Bind_Timing(The_Operator_Name,
$3.Integer_Value,
The_Per);
}
```

```plaintext
opt_finish_within
: FINISH_TOKEN WITHIN_TOKEN time reqmts_trace
{
    Bind_Timing(The_Operator_Name,
$3.Integer_Value,
The_Fw);
}
```
opt_mcp
  : MINIMUM_TOKEN CALL_PERIOD_TOKEN time reqmts_trace
       { Bind_Timing(The_Operator_Name,
                    $3.Integer_Value,
                    The_Mcp); }

Opt_Mrt
  : max_resp_time time reqmts_trace
       { Bind_Timing(The_Operator_Name,
                    $3.Integer_Value,
                    The_Mrt); }

max_resp_time
  : MAXIMUM_TOKEN RESPONSE_TOKEN TIME_TOKEN

timer_op
  : RESET_TOKEN
     { $S := (Token_Category => Timer_Op_Id_String,
              Timer_Op_Id_Value => Reset); }

  | START_TOKEN
     { $S := (Token_Category => Timer_Op_Id_String,
              Timer_Op_Id_Value => Start); }

  | STOP_TOKEN
     { $S := (Token_Category => Timer_Op_Id_String,
              Timer_Op_Id_Value => Stop); }

opt_if_predicate
  : IF_TOKEN expression
We add each expression in the _mit_expr_seq_ to preserve the order of expressions corresponding each state. This sequence is used by procedure _Bind_Initial_Expression_ together with states map to construct the _Init_Map_.

Initialization of the sequence is done before (by the parent rule).

```c
initial_expression_list
  : initial_expression_list ','
  |
  |    The_Expression_String := Expression(A_Strings.Empty);
  |
initial_expression
  |
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    Temp_Init_Expr_Seq);
  Exp_Seq_Pkg.Add ($4<Expression_Value,
    Temp_Init_Expr_Seq);
  Init_Exp_Seq_Pkg.Push (The_Init_Exp_Seq_Stack,
    Temp_Init_Expr_Seq);
  |
  |
initial_expression
  |
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    Temp_Init_Expr_Seq);
  Exp_Seq_Pkg.Add ($2<Expression_Value,
    Temp_Init_Expr_Seq);
  Init_Exp_Seq_Pkg.Push (The_Init_Exp_Seq_Stack,
    Temp_Init_Expr_Seq);
  |
```

There is one and only one initial state (initial expression) for each state variable. This production return one expression to the parent rule corresponding to one state. This is done by using the internal stack ($$ convention)

The global variable the_expression_string also holds the value of the initial expression, and is needed to get the string value of the expression resulted by the type name and type_decl productions. The _initial_expression_string_
-- /* variable is initialized in the same way by the parent rule */
-- /* to empty_expression. */

initial_expression
  : TRUE
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => To_A("True"));
  }

  | FALSE
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => To_A("False"));
  }

  | INTEGER_LITERAL
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => Expression(The_Integer-Token));
  }

  | REAL_LITERAL
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => The_Real-Token);
  }

  | STRING_LITERAL
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => The_String-Token);
  }

  | IDENTIFIER
  {
    $$ := (Token_Category => Expression_String,
           Expression_Value => Expression(The_Id-Token));
  }

-- /* We Initialized The(Expression_String To Empty */
-- /* At The Parent Rule, So That Type_Name Production */
-- /* Will Get The(Expression_String As An Empty Variable */

  | type_name . IDENTIFIER
  {
    The(Expression_String) := The(Expression_String) & "." & Expression(The_Id-Token);
    $$ := (Token_Category => Expression_String,
           Expression_Value => The(Expression_String));
  }

97
<table>
<thead>
<tr>
<th>type_name ' . ' IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
</tbody>
</table>
|   $S$ := (Token_Category => Expression_String,
|     Expression_Value => The_Expression_String & ".",
|         & Expression(The_Id_Token));      |
| }                          |
| ' ( ' Init_Exp_Seq_Stack_Pkg.Push(The_Init_Exp_Seq_Stack,
|   Empty_Exp_Seq);           |
| )                          |
| initial_expression_list ')' |
| {                          |
|   --/* we remove expression resulted by the */
|   --/* previous rule, since expression will */
|   --/* be concatenation of Type_name.ID and */
|   --/* value of previous production */
|   Init_Exp_Seq_Stack_Pkg.Pop(The_Init_Exp_Seq_Stack,
|     Temp_Init_Expr_Seq);           |
| The_Expression_String := Expression(A_Strings.Empty); |
| for i in 1 .. Exp_Seq_Pkg.Length(Temp_Init_Expr_Seq) loop |
|   if i > 1 then
|     The_Expression_String := The_Expression_String & ",";
|   end if;
|   The_Expression_String :=
|     The_Expression_String &
|     Exp_Seq_Pkg.Fetch(Temp_Init_Expr_Seq, i); |
| end loop;
| Exp_Seq_Pkg.Recycle(Temp_Init_Expr_Seq); -- throw it away
| $S$ := (Token_Category => Expression_String,
|     Expression_Value => $4.Expression_Value & "(" &
|     The_Expression_String & ")")$); |
| }                          |
| ' ( ' initial_expression ') ' |
| {                          |
|   $S$ := (Token_Category => Expression_String,
|     Expression_Value => To_A("") &
|     $2.Expression_Value &
|     To_A("")); |
| }                          |
| initial_expression log_op |
| {                          |
|   $S$ := (Token_Category => Expression_String,
|     Expression_Value => $1.Expression_Value &
|     $2.Expression_Value); |
| }                          |
initial_expression %prec logical_operator
{ 
  $$ := (Token_Category => Expression_String,
        Expression_Value => $3.Expression_Value &
        $4.Expression_Value);
}

| initial_expression rel_op %prec relational_operator

initial_expression %prec relational_operator
{ 
  $$ := (Token_Category => Expression_String,
        Expression_Value => $1.Expression_Value &
        $2.Expression_Value &
        $3.Expression_Value);
}

| '-' initial_expression %prec unary_adding_operator
{ 
  $$ := (Token_Category => Expression_String,
        Expression_Value => To_A("-") & $2.Expression_Value);
}

| '+' initial_expression %prec unary_adding_operator
{ 
  $$ := (Token_Category => Expression_String,
        Expression_Value => To_A("+") & $2.Expression_Value);
}

| initial_expression bin_add_op %prec multiplying_operator

| initial_expression bin_mul_op %prec multiplying_operator
{ 
  $$ := (Token_Category => Expression_String,
        Expression_Value => $1.Expression_Value &
        $2.Expression_Value &
        $3.Expression_Value);
}
initial_expression EXP_TOKEN

initial_expression %prec highest_precedence_operator
{
    $S := (Token_Category => Expression_String, Expression_Value => $1.Expression_Value & To_A(" EXP ") & $3.Expression_Value);
}

NOT_TOKEN

initial_expression %prec highest_precedence_operator
{
    --Exp_Seq_PKG.Add( The_Expression_String, The_Exp_Seq);
    $S := (Token_Category => Expression_String, Expression_Value => To_A(" NOT ") & $2.Expression_Value);
}

ABS_TOKEN

initial_expression %prec highest_precedence_operator
{
    $S := (Token_Category => Expression_String, Expression_Value => To_A(" NOT ") & $2.Expression_Value);
}

log_op

AND_TOKEN
{
    $S := (Token_Category => Expression_String, Expression_Value => To_A(" AND "));
}

OR_TOKEN
{
    $S := (Token_Category => Expression_String, Expression_Value => To_A(" OR "));
}

XOR_TOKEN
{
    $S := (Token_Category => Expression_String, Expression_Value => To_A(" XOR "));
}
rel_op

: (Token-Category => Expression-String,
Expression-Value => To_A(" < "));

<table>
<thead>
<tr>
<th>'&gt;'</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" > ")); |

<table>
<thead>
<tr>
<th>'='</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" = ")); |

<table>
<thead>
<tr>
<th>GREATER_THAN_OR_EQUAL</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" >= ")); |

<table>
<thead>
<tr>
<th>LESS_THAN_OR_EQUAL</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" <= ")); |

<table>
<thead>
<tr>
<th>INEQUALITY</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" /= ")); |

bin_add_op

: '+' |
| --- |
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" + ")); |

<table>
<thead>
<tr>
<th>'-'</th>
</tr>
</thead>
</table>
| $S := (Token-Category => Expression-String,
Expression-Value => To_A(" - ")); |
<table>
<thead>
<tr>
<th>'6'</th>
</tr>
</thead>
</table>
| \$
\$
\$
| (Token Category  => Expression_String,  
Expression_Value  => To_A(" & "));  
| ;  

bin_mul_op  
: 'a'
| \$
\$
\$
| (Token Category  => Expression_String,  
Expression_Value  => To_A(" + "));  
| ;  

| '/' 
| \$
\$
\$
| (Token Category  => Expression_String,  
Expression_Value  => To_A(" - "));  
| ;  

| MOD_TOKEN  
| \$
\$
\$
| (Token Category  => Expression_String,  
Expression_Value  => To_A(" MOD "));  
| ;  

| REM_TOKEN  
| \$
\$
\$
| (Token Category  => Expression_String,  
Expression_Value  => To_A(" REM "));  
| ;  

| time  
| time_number MICROSEC_TOKEN  
| \$
\$
\$
| (Token Category  => Integer_Literal,  
Integer_Value  => ($1.Integer_Value + 999)/1000);  
The_Time_String :=  
To_A(Integer'Image($1.Integer_Value) & " microsec");  
| }  

| time_number MS_TOKEN  
| \$
\$
\$
| (Token Category  => Integer_Literal,  
Integer_Value  => $1.Integer_Value);  
The_Time_String :=  
To_A(Integer'Image($1.Integer_Value) & " ms");  
| }  

| time_number SEC_TOKEN  
| \$
\$
\$
| (Token Category  => Integer_Literal,  
Integer_Value  => $1.Integer_Value);  
The_Time_String :=  
To_A(Integer'Image($1.Integer_Value) & " ms");  
| }  

102
\[ S$ := (\text{Token Category} \Rightarrow \text{Integer Literal}, \]
\[ \text{Integer Value} \Rightarrow \$1.\text{Integer Value} \times 1000); \]
\[ \text{The Time String} := \]
\[ \text{To}_A(\text{Integer'Image}(\$1.\text{Integer Value}) \text{" sec"}); \]

| time_number MIN_TOKEN
| ---
| \[ S$ := (\text{Token Category} \Rightarrow \text{Integer Literal}, \]
| \[ \text{Integer Value} \Rightarrow \$1.\text{Integer Value} \times 60000); \]
| \[ \text{The Time String} := \]
| \[ \text{To}_A(\text{Integer'Image}(\$1.\text{Integer Value}) \text{" min"}); \]

| time_number HOURS_TOKEN
| ---
| \[ S$ := (\text{Token Category} \Rightarrow \text{Integer Literal}, \]
| \[ \text{Integer Value} \Rightarrow \$1.\text{Integer Value} \times 3600000); \]
| \[ \text{The Time String} := \]
| \[ \text{To}_A(\text{Integer'Image}(\$1.\text{Integer Value}) \text{" hrs"}); \]

\[ \text{time number} \]
\[ \text{INTEGER LITERAL} \]
\[ \{ \]
\[ S$ := (\text{Token Category} \Rightarrow \text{Integer Literal}, \]
\[ \text{Integer Value} \Rightarrow \text{Convert To Digit(The Integer Token.$))}); \]
\[ \} \]

\[ --/* \text{Initialization of The Expression String should */} \]
\[ --/* \text{should be done by the parent rules */} \]
\[ \text{expression list} \]
\[ : \text{expression list } ,,' \]
\[ \{ \]
\[ \text{The Time String} := \text{Expression(A Strings.Empty)}); \]
\[ \} \]
\[ \text{expression} \]
\[ \{ \]
\[ \text{The Time String} := \text{Expression(A Strings.Empty)}); \]
\[ \} \]
\[ \text{expression} \]

;
Expressions Can Appear In Guards Appearing In Control Constraints.
These Guards Can Be Associated With Triggering Conditions, Or
Conditional Outputs, Conditional Exceptions, Or Conditional Timer
Operations. Similar To Initial Expression, Except That Time Value,
and References To Timers And Data Streams Are Allowed.

expression

: TRUE
{
   The_Expression_String := The_Expression_String & " TRUE ";
}

| FALSE
{
   The_Expression_String := The_Expression_String & " FALSE ";
}

| INTEGER_LITERAL
{
   The_Expression_String := The_Expression_String & " " &
                        Expression(The_Integer-Token);
}

| time
{
   The_Expression_String := The_Expression_String & " " &
                        The_Time_String;
}

| REAL_LITERAL
{
   The_Expression_String := The_Expression_String & " " &
                        The_Real-Token;
}

| STRING_LITERAL
{
   The_Expression_String := The_Expression_String & " " &
                        The_String-Token;
}

| IDENTIFIER
{
   The_Expression_String := The_Expression_String & " " &
                        Expression(The_Id-Token);
}

| type_name '.' IDENTIFIER
{
The_Expression_String := The_Expression_String & "." & Expression(The_Id_Token);
}

| type_name '.' IDENTIFIER
| The_Expression_String := The_Expression_String & "." & Expression(The_Id_Token);
|
| '{'
| The_Expression_String := The_Expression_String & ";"; }
expression_list '{'
| The_Expression_String := The_Expression_String & "}";
Exp_Seq_Pkg.Add( The_Expression_String, The_Exp_Seq);
|
| '}'
| The_Expression_String := The_Expression_String & ";"; }
expression ')'
| The_Expression_String := The_Expression_String & ")."; }

| expression log_op
| The_Expression_String :=
| The_Expression_String & $2.Expression_Value;
|
expression        %prec logical_operator

| expression rel_op
| The_Expression_String :=
| The_Expression_String & $2.Expression_Value;

| expression        %prec relational_operator
| '
| The_Expression_String := The_Expression_String & ";\"; }
expression        %prec unary_adding_operator

| '+'
| The_Expression_String := The_Expression_String & "+"; }
expression        %prec unary_adding_operator

| expression bin_add_op
|
The_Expression_String :=
  The_Expression_String & $2.Expression_Value;
)
expression
  %prec binary_adding_operator
| expression bin_mul_op
  {
    The_Expression_String :=
      The_Expression_String & $2.Expression_Value;
  }
expression
  %prec multiplying_operator
| expression EXP_TOKEN
  {
    The_Expression_String :=
      The_Expression_String & " EXP ";
  }
expression
  %prec highest_precedence_operator
| NOT_TOKEN
  { The_Expression_String := To_A(" NOT "); }
expression
  %prec highest_precedence_operator
| ABS_TOKEN
  { The_Expression_String := To_A(" ABS "); }
expression
  %prec highest_precedence_operator
;
package Parser is

-- Global Variable Which Is A Map From PSDL_Component Names To PSDL
-- Component Definitions
The_Program : PSDL_Program; -- Implemented

-- Global Variable For A PSDL_Component (Type Or Operator)
The_Component : PSDL_Component; -- Implemented

-- Global Variable Which Points To The PSDL_Component (Type Or Operator)
The_Component_Ptr : Component_Ptr; -- Implemented

-- Global Variable Which Points To The PSDL_Operator (Type Or Operator)
The_Op_Ptr : Op_Ptr; -- Implemented

-- used to construct the operation map
The_Operator : Operator;

-- Global Variable For An Atomic Type -- Implemented
The_Atom_Type : Atomic_Type;

-- Global Variable For An Atomic Operator
The_Atom_Operator : Atomic_Operator; -- Implemented

-- Global Variable For A Composite PSDL Type
The_Composite_Type : Composite_Type; -- Implemented
-- Global Variable For A Composite Psdl Type

The_Composite_Operator : Composite_Operator;

-- /* Global Variables For All Psdl Components: */

-- Global Variable Which Holds The Name Of The Component

The_Psdl_Name : Psdl_Id;

-- Global Variable Which Holds The Ada_Id Variable Of Component Record

The_Ada_Name : Ada_Id;

-- Global Variable Which Holds The Generic Parameters

The_Gen_Par : Type_Declaration;

-- used for psdl_type part (for not to mix with operation map)

The_Type_Gen_Par : Type_Declaration;

-- Global Variable Which Holds The Keywords

The_Keywords : Id_Set;

The_Description : Text;

The_Axioms : Text;

-- A Temporary Variable To Hold Output_Id To Construct Out_Guard Map

The_Output_Id : Output_Id;

-- A Temporary Variable To Hold Excep_Id To Construct Excep_Trigger Map

The_Excep_Id : Excep_Id;

-- Global Variables For All Psdl Types:

-- Used For Creating All Types

The_Model : Type_Declaration;
The Operation Map
: Operation_Map;

-- Implemented

-- Used For Creating Composite Types

The Data Structure
: Type_Name;

-- Implemented

-- Global Variables For All Operators:

The Input
: Type_Declaration;

-- Implemented

The Output
: Type_Declaration;

-- Implemented

The State
: Type_Declaration;

-- Implemented

The Initial Expression
: Init_Map;

-- Implemented

The Exceptions
: Id_Set;

-- Implemented

The Specified Met
: Millisec;

-- Implemented

-- Global Variables For Composite Operators:

The Graph
: Psdl_Graph;

-- Implemented

The Streams
: Type_Declaration;

-- Implemented

The Timers
: Id_Set;

-- Implemented

The Trigger
: Trigger_Map;

-- Implemented

The Exec Guard
: Exec_Guard_Map;

-- Implemented

The Out Guard
: Out_Guard_Map;

-- Implemented

The Excep Trigger
: Excep_Trigger_Map;

-- Implemented

The Timer Op
-- Implemented
The Per : Timing_Map; -- Implemented
The_Fw : Timing_Map; -- Implemented
The_Ncp : Timing_Map; -- Implemented
The_Nrt : Timing_Map; -- Implemented

The_Impl_Desc : Text := Empty_Text;

-- Is Used For Storing The Operator Names In Control Constraints Part

The_Operator_Name : PSDL_Id;

-- A Place Holder To For Time Values

The_Time : Millisec;

-- True If The PSDL_Component Is An Atomic One

Is_Atomc_Type : Boolean; -- Implemented

Is_Atomc_Operator: Boolean;

-- Holds The Name Of The Edge (I.E Stream Name)

The_Edge_Name : PSDL_Id; -- Implemented

-- Renames The Procedure Bind In Generic Map Package
-- PSDL Program Is A Mapping From PSDL Component Names ..
-- .. To PSDL Component Definitions

Procedure Bind_Program
( Name : In PSDL_Id;
  Component : In Component_Ptr;
  Program : In Out
  PSDL_Program )
Renames Bind;
-- Renames The Procedure Bind In Generic Map Package
-- Psdl Program Is A Mapping From Psdl Id'S To Psdl Type Names

Procedure Bind_Type_Decl_Map
( Key : In Psdl_Id;
  Result : In Type_Name;
  Map : In Out
  Type_Declaration )
Renames Type_Declaration_Pkg.
  Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Operation Map Is A Mapping From Psdl Operator Names To Psdl ...
-- .. Operator Definitions.

Procedure Bind_Operation
( Key : In Psdl_Id;
  Result : In Op_Ptr;
  Map : In Out Operation_Map )
Renames Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Trigger Map Is A Mapping From Psdl Operator Names To Trigger ..
-- .. Types (By Some, By All, None ..

Procedure Bind_Trigger
( Key : In Psdl_Id;
  Result : In Trigger_Record;
  Map : In Out Trigger_Map )
Renames Trigger_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Timing Map Is A Mapping From Psdl Operator Names To ..
-- .. Some Timing Parameters (Per, Mrt, Fw, Mop, ...)

Procedure Bind_Timing
( Key : In Psdl_Id;
  Result : In Millisec;
  Map : In Out Timing_Map )
Renames Timing_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Out_Guard Map Is A Mapping From Output Stream Id'S To ...
-- .. Expression Strings
Procedure Bind_Out_Guard
( Key : In Output_Id;
 Result : In Expression;
 Map : In Out_Out_Guard_Map )
Renames Out_Guard_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Init_Map Is A Mapping From Psdl Id'S To ..
-- .. Expression Strings
Procedure Bind_Init_Map
( Key : In Psdl_Id;
 Result : In Expression;
 Map : In Out_Init_Map )
Renames Init_Map_Pkg.Bind;

- Renames The Procedure Bind In Generic Map Package
-- Timer_Op_Map Is A Mapping From Psdl Id'S To ..
-- .. Timer_Op_Set
Procedure Bind_Timer_Op
( Key : In Psdl_Id;
 Result : In Timer_Op_Set;
 Map : In Out Timer_Op_Map )
Renames Timer_Op_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Exception Trigger Map Is A Mapping From Psdl Id'S To ..
-- .. Expression Strings
Procedure Bind_Excep_Trigger
( Key : In Excep_Id;
 Result : In Expression;
 Map : In Out
 Excep_Trigger_Map )
Renames Excep_Trigger_Map_Pkg.
 Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Exec_Guard Map Is A Mapping From Psdl Id'S To ..
-- .. Expression Strings
Procedure Bind_Exec_Guard

112
(Key: in PdId_Id;
Result: in Expression;
Map: in out Exec_Guard_Map)
Renames Exec_Guard_Map_Pkg.Bind;

-- Implements A Temporary Storage For Type Declaration.

Package Type_Decl_Stack_Pkg Is
  New Stack_Pkg (Type_Declaration);
Use Type_Decl_Stack_Pkg;

Subtype Type_Decl_Stack Is
  Type_Decl_Stack_Pkg.Stack;

-- A Stack Declaration And Initialization For Type_Declaration

The_Type_Decl_Stack
  := Type_Decl_Stack :=
      Type_Decl_Stack_Pkg.Create;

Package Id_Set_Stack_Pkg Is
  New Stack_Pkg (Id_Set);

Subtype Id_Set_Stack Is
  Id_Set_Stack_Pkg.Stack;

-- A Stack Declaration And Initialization For Id

The_Id_Set_Stack
  := Id_Set_Stack :=
      Id_Set_Stack_Pkg.Create;

-- Global Declaration For Type_Id_Set

The_Id_Set
  := Id_Set;

The_Id_Set_Size
  := Natural;

Package Expression_Stack_Pkg Is
  New Stack_Pkg (Expression);

Subtype Expression_Stack Is
  Expression_Stack_Pkg.Stack;
-- A Stack Declaration And Initialization For Id

The Expression Stack
: Expression Stack :=
Expression Stack Pkg.Create;

Package Exp Seq Pkg Is
New Generic Sequence Pkg (T =>
Expression, Block Size => 24)
);

Subtype Exp Seq Is
Exp Seq Pkg.Sequence;

-- returns an empty expression sequence
function Empty Exp Seq return Exp Seq;

The Exp Seq
: Exp Seq;

The Init Expr Seq : Exp Seq; -- Used For Constructing Init Map
Temp Init Expr Seq : Exp Seq;

package Init Exp Seq Stack Pkg is
new Stack Pkg (Exp Seq);

subtype Init Exp Seq Stack is Init Exp Seq Stack Pkg.Stack;

The Init Exp Seq Stack :
Init Exp Seq Stack := Init Exp Seq Stack Pkg.Create;

Procedure Remove Expr From Seq Is
New Exp Seq Pkg.Generic Remove (Eq => "=");

Package Id Seq Pkg Is
New Generic Sequence Pkg (T => Psdl Id,
Block Size => 24);

Subtype Id Seq Is
Id Seq Pkg.Sequence;

The Id Seq
: Id Seq;

The Init Map Id Seq: Id Seq; -- to hold the id's to construct init map
-- these are the same id's used in state map.

-- Holds The Name Of The Types;
The_Type_Name : Type_Name;

-- Used For The Type Decl Part Of Type_Name
The_Type_Name_Decl : Type_Declaration;

-- A Temporary Type_Decl
Temp_Type_Decl : Type_Declaration;

-- A Temporary Variable For Holding The Identifiers
The_String : Psdl_Id;

-- A Temporary Variable For Trigger_Record
The_Trigger_Record : Trigger_Record;

-- A Temp Variable For Holding The Value Of Timer_Op
The_Timer_Op_Record : Timer_Op;

The_Timer_Op_Set : Timer_Op_Set;

-- A Temp Variable For Producing The Expression String
The_Expression_String : Expression := Expression( A_Strings.Empty);

-- A Temp Variable For Producing The Time String
The_Time_String : Expression := Expression( A_Strings.Empty);

Echo : Boolean := False;

Number_Of_Errors : Natural := 0;

Semantic_Error : Exception;

Procedure Yyparse;

procedure GET(Item : out PSDL_PROGRAM);
procedure GET(Input_File_N : in String;
       Output_File_N : in String := "");

end Parser;

with Psdl_Tokens, Psdl_Goto,
    Psdl_Shift_Reduce, Psdl_Lex,
    Text_Io, Psdl_Lex_Dfa,
    Psdl_Lex_Io, A_Strings,
    Psdl_Concrete_Type_Pkg,
    Psdl_Graph_Pkg,
    Generic_Sequence_Pkg;
use Psdl_Tokens, Psdl_Goto,
    Psdl_Shift_Reduce, Psdl_Lex,
    Text_Io,
    Psdl_Concrete_Type_Pkg,
    Psdl_Graph_Pkg;

package body Parser is

    Type_Spec_Gen_Par : Boolean := FALSE;

    function Empty_Exp_Seq return Exp_Seq is
        S : Exp_Seq;
        begin
            Exp_Seq_Pkg.Empty(S);
            return S;
        end Empty_Exp_Seq;

    procedure Yyerror
        procedure Yyerror
( S : In String :=
   "Syntax Error" ) is
Space
   : Integer;

begin -- Yyerror
Number_Of_Errors :=
   Number_Of_Errors + 1;
Text_Io.New_Line;
Text_Io.Put("Line" & Integer'
   Image(Lines - 1) & " ");
Text_Io.Put_Line(Psdl_Lex_Dfa.
   Yytext);
Space := Integer(Psdl_Lex_Dfa.
   Yytext'Length) + Integer'
   Image(Lines)'Length + 5;
for I In 1 .. Space loop
   Put("- ");
   end loop;
end Yyerror;

-----------------------------------------------

-- function Convert_To_Digit
-- Given A String Of Characters Corresponding To A Natural Number,
-- Returns The Natural Value
-----------------------------------------------

function Convert_To_Digit
   ( String_Digit : String )
   Return Integer Is
Multiplier
   : Integer := 1;
Digit, Nat_Value
   : Integer := 0;

Begin -- Convert_To_Digit
For I In Reverse 1 ..
   String_Digit'Length Loop
   Case String_Digit(I) Is
      When '0' =>
         Digit := 0;
      When '1' =>
         Digit := 1;
      When '2' =>
         Digit := 2;
      When '3' =>
         Digit := 3;
      When '4' =>
         Digit := 4;
      When '5' =>
         Digit := 5;
      When '6' =>
         Digit := 6;
Digit := 6;
When '7' =>
  Digit := 7;
When '8' =>
  Digit := 8;
When '9' =>
  Digit := 9;
When Others =>
  Null;
End Case;
Nat_Value := Nat_Value + (Multiplier * Digit);
Multiplier := Multiplier * 10;
End Loop;
Return Nat_Value;
end Convert_To_Digit;

-- procedure GET
--
-- Reads the psdl source file, parses it and creates the PSDL ADT
-- Input file is line numbered and saved into a file
-- input file name .1st in the current directory. So if
-- there is no write permission for that directory, exception
-- Use_Error is raised and program aborts. if the second argument
-- is passed psdl file resulted from PSDL ADT is written into a
-- file with that name.

procedure GET(Input_File_N : in String;
              Output_File_N : in String := "";
              Item out PSDL_PROGRAM ) is

begin
  Psdl_LexIo.OpenInput(Input_File_N);
  if Output_File_N /= "" then
    Psdl_LexIo.CreateOutput(Output_File_N);
  else
    Psdl_LexIo.CreateOutput;
  end if;
  Psdl_Lex.Linenum;
  YYParse;
  Psdl_LexIo.CloseInput;
  Psdl_LexIo.CloseOutput;
  Item := The_Program;
  TextIo.Close(Psdl_Lex.List_File);
end Get;
procedure GET

-- Reads the standard input, parses it and creates the
-- PSDL ADT. Input file is line numbered and saved into a
-- file input file name .lst in the current directory. So if
-- there is no write permission for that directory, exception
-- Use_Error is raised and program aborts.

procedure GET(Item : out PSDL_PROGRAM) is
begin
  Text_Io.Create(Psdl_Lex.List_File, Out_File, "stdin.psdl.lst");
  Psdl_Lex.Linenum;
  YYParse;
  Psdl_Lex_Io.Close_Input;
  Psdl_Lex_Io.Close_Output;
  Item := The_Program;
  Text_Io.Close(Psdl_Lex.List_File);
end Get;

procedure Bind_Type_Declaration

--/* Bind Each Id In Id The Id */
--/* Set To The Type Name */
--/* Return Temp_Type_Dccl */

Procedure Bind_Type_Declaration(IS: In IdSet; Tn In Type_Name; Td : In out Type_Declaration) is

begin
  --/* m4 code
  --/* foreach([Id: Psdl_Id], [Id_Set_Pkg.Generic_Scan],
  --/*   [I_s],
  --/*   [B_Type_Decl_Map(Id, Tn, Td);
  --/* ])

  --/* Begin expansion of FOREACH loop macro.
  declare
    procedure Loop_Body(Id: Psdl_Id) is
      begin
        Bind_Type_Decl_Map(Id, Tn, Td);
      end Loop_Body;

end Loop_Body;
procedure Execute_Loop is
    new IdSet_Pkg.Generic_Scan(Loop_Body);
begin
    execute_loop(I_s);
end;
--/* end of expansion of FOREACH loop macro.

end Bind_Type_Declaration;

-- procedure Bind_Initial_State
-- /* Bind Each Id In the State map domain
-- /* Set To The Type Name initial expression

procedure Bind_Initial_State:
    State : in Type_Declaration;
    Init_Seq : in Exp_Seq;
    Init_Exp_Map : out Init_Map) is
i : Natural := 1;

-- M4 macro code for binding each initial expression in
-- the_init_expr_seq to the id's in state declaration map
-- foreach([Id: in Psdl_Id; Tn: in Type_Name],
-- [Type_Declaration_Pkg.Generic_Scan],
-- [State],
-- [TheInitial_Expression])
-- i := i + 1;

begin
    -- Begin expansion of FOREACH loop macro.
    declare
        procedure Loop_Body(Id: Sn PsdlId; Tn: in Type_Name) is
            begin
                if i > Exp_Seq_Pkg.Length(The_Init.Expr_Seq) then
                    Yyerror("SEMANTIC ERROR - Some states are not initialized.");
                    Raise SEMANTIC_ERROR;
                else
                    Bind_Init_Map(Id, Exp_Seq_Pkg.Fetch(The_Init.Expr_Seq, i),
                        The_Initial_Expression);
                    i := i + 1;
                end if;
        end Loop_Body;
    procedure execute_loop is new Type_Declaration_Pkg.Generic_Scan(Loop_Body);
    begin
        execute_loop(State);
    end;
    -- LIMITATIONS: Square brackets are used as macro quoting characters,
    -- so you must write [[x]] in the m4 source file
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "DEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.

-- if number of initial states > number of states, raise exception
-- and abort parsing
if (i-1) < Exp_Seq_Pkg.Length(The_Init_Expr_Seq) then
  Yyerror("SEMANTIC ERROR - There are more initializations than the states");
  raise SEMANTIC_ERROR;
end if;
end Bind_Initial_State;

begin

procedure Build_PSdl_Type
(C_Name : in Psdl_Name;
 C_a_Name : in Ada_Name;
 Mdl : in Type_Declaration;
 D_Str : in Type_Name;
 Ops : in Operation_Map;
 G_Par : in out Type_Declaration;
 Kwr : in out Id_Set;
 I_Desc : in out Text;
 F_Desc : in out Text;
 Is.Atomic : in Boolean;
 The_Type : in out Data_Type) is
begin

if IS_ATOMIC then
  The_Type := Make_Atom_Type
    ( Psdl_Name => C_Name,
      Ada_Name => C_a_Name,
      Model => Mdl,
      Gen_Par => G_Par,
      Operations => Ops,
      Keywords => Kwr,
      Informal_Description => I_Desc,
      Axioms => F_Desc );

else
  The_Type := Make_Composite_Type
    ( Name => C_Name,
      Psdl_Name => C_Name,
      Ada_Name => C_a_Name,
      Model => Mdl,
      Gen_Par => G_Par,
      Operations => Ops,
      Keywords => Kwr,
      Informal_Description => I_Desc,
      Axioms => F_Desc );

end if;
Model => Mdl,
Data_Structure => D_Str,
Operations => Ops,
Gen_Par => G_Par,
Keywords => Kwr,
Informal_Description => I_Desc,
Axioms => F_Desc);
begin

if IS_ATOMIC then
   The_Opr := Make_Atomic_Operator
   ( Psdl_Name => C_Name,
     Ada_Name  => C_A_Name,
     Gen_Par   => G_Par,
     Keywords  => Kwr,
     Informal_Description
                => I_Desc,
     Axioms    => F_Desc,
     Input     => Inp,
     Output    => Otp,
     State     => St,
     Initialization_Map
                => I_Exp_Map,
     Exceptions => Excps,
     Specified_Met => S_Met);
else
   The_Opr := Make_Composite_Operator
   ( Name      => C_Name,
     Gen_Par   => G_Par,
     Keywords  => Kwr,
     Informal_Description
                => I_Desc,
     Axioms    => F_Desc,
     Input     => Inp,
     Output    => Otp,
     State     => St,
     Initialization_Map
                => I_Exp_Map,
     Exceptions => Excps,
     Specified_Met => S_Met,
     Graph     => Gr,
     Streams   => D_Stream,
     Timers    => Tmrs,
     Trigger   => Trigs,
     Exec_Guard => E_Guard,
     Out_Guard => O_Guard,
     Excep_Trigger => E_Trigger,
     Timer_Op  => T_Op,
     Per       => Per,
     Fw        => Fw,
     Mct       => Mct,
     Mrt       => Mrt,
     Impl_Desc => Im_Desc);
end if;

/* After constructing the component */
/* initialized the global variables for */
/* optional attributes */
end Build_Psdl_Operator;

procedure Add_Op_Impl_To_Op_Map(Op_Name in Psdl_Id;
A_Name : in Ada_Id;
Is_Atomic : in Boolean;
O_Map : in out Operation_Map;
Gr : in out Psdl_Graph;
D_Stream : in out Type_Declaration;
Tmrs : in out Id_Set;
Trigs : in out Trigger_Map;
E_Guard : in out Exec_Guard_Map;
O_Guard : in out Out_Guard_Map;
E_Trigger : in out Excep_Trigger_Map;
T_Op : in out Timer_Op_Map;
Per : in out Timing_Map;
Fw : in out Timing_Map;
Mcp : in out Timing_Map;
Mrt : in out Timing_Map;
Im_Desc := Emptt_Text;
end Add_Op_Impl_To_Op_Map;
Mcp : in out Timing_Map;
Mrt : in out Timing_Map;
Im_Desc : in out Text ) is

Temp_Op : Operator;
Temp_Op_Ptr : Op_Ptr;

begin
  if Operation_Map_Pkg.Member(Op_Name, Operation_Map_Pkg.Map(O_Map)) then
    Temp_Op := Operation_Map_Pkg.Fetch(Operation_Map_Pkg.Map(O_Map),


Op_Name).all;
Operation_Map_Pkg.Remove(Op_Name, Operation_Map_Pkg.Map(O_Map));
if IsAtomic then
  Temp_Op := MakeAtomicOperator
(Psdl_Name => Op_Name,
Ada_Name => A_Name,
Gen_Par => Generic_Parameters(Temp_Op),
Keywords => Keywords(Temp_Op),
Informal_Description
  => Informal_Description(Temp_Op),
Axioms => Axioms(Temp_Op),
Input => Inputs(Temp_Op),
Output => Outputs(Temp_Op),
State => States(Temp_Op),
Initialization_Map
  => Get_Init_Map(Temp_Op),
Exceptions=> Exceptions(Temp_Op),
Specified_Met =>
  Specified_Maximum_Execution_Time(Temp_Op));
Temp_Op_Ptr := new Operator (Category => Psdl.Operator,
Granularity => Atomic);
Temp_Op_Ptr.all := Temp_Op;
else
  Temp_Op := MakeCompositeOperator
(Name => Op_Name,
Gen_Par => Generic_Parameters(Temp_Op),
Keywords => Keywords(Temp_Op),
Informal_Description
  => Informal_Description(Temp_Op),
Axioms => Axioms(Temp_Op),
Input => Inputs(Temp_Op),
Output => Outputs(Temp_Op),
State => States(Temp_Op),
Initialization_Map
  => Get_Init_Map(Temp_Op),
Exceptions=> Exceptions(Temp_Op),
Specified_Met =>
  Specified_Maximum_Execution_Time(Temp_Op),
Graph => Gr,
Streams => D_Stream,
Timers => Tmrs,
Trigger => Trigs,
Exec_Guard => E_Guard,
Out_Guard => O_Guard,
Excep_TRIGGER => E_TRIGGER,
Timer_OP => T_OP,
Per => Per,
Fw => Fw,
Mcp => Mcp,
Mrt => Mrt,
Impl_Desc => Im_Desc);
Temp_Op_Ptr := new Operator (Category => Psdl.Operator,
Granularity => Composite);

    Temp_Op_Ptr.all := Temp_Op;
end if;
Bind_Operation(Op_Name, Temp_Op_Ptr, O_Map);

-- reset everything after you are done.(the variables that have default values)
Gr := Empty_Psdl_Graph;
D_Stream := Empty_Type_Declaration;
Tmr := Empty_Id_Set;
Trigs := Empty_Trigger_Map;
E_Guard := EmptyExec_Guard_Map;
O_Guard := Empty_Out_Guard_Map;
E_Trigger := Empty_Excep_Trigger_Map;
T_Op := Empty_Timer_Op_Map;
Per := Empty_Timing_Map;
Fw := Empty_Timing_Map;
Mcp := Empty_Timing_Map;
Mrt := Empty_Timing_Map;
Im_Desc := EMpty_Text;
else
    Put("Warning: The specification of operator "");
    Put_line(Op_Name.s & " I was not given, implementation ignored.");
end if;
end Add_Op_Impl_To_Op_Map;

##procedure_parse
end Parser;
APPENDIX D. MAIN PROGRAM FOR THE EXPANDER

--:----------:
-- expander.a
--:----------:

--------------------------------------------------------------------------
--
-- Unit name : Main procedure for the PSDL Expander
-- File name : expander.a
-- Author : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Date Created : July 1991
-- Last Update : (Mon Sep 23 23:16:31 1991 - bayram)
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1.1,
                          Verdix Ada ver. 6.0(c)
--
--------------------------------------------------------------------------
--
-- Keywords : PSDL expander, multi-level to two-level
--
-- Abstract : This file contains main driver procedure for the expander
-- Uses command Unix command line interface, non-standard package U_ENV
--

--------------------------------------------------------------------------
-- Revision history -----------------------------------------------
--
--$Source: /n/gemini/work/bayram/AYACC/parser/RCS/expander.a,v $
--$Revision: 1.2 $
--$Date: 1991/09/24 06:26:50 $
--$Author: bayram $
--

--------------------------------------------------------------------------

with U_Env, Psdl_Component_Pkg,
    Psdl_Tokens, Parser,
    Text_Io, Psdl_Io;

use Text_Io, Psdl_Component_Pkg;

128
procedure Expander is

The_Psdl_Component : Psdl_Component_Pkg.Psdl_Program := Empty_Psdl_Program;

begin

-- Command: "expander" or "<command> | expander",
-- reads the standard input, outputs to standard output
if U_Env.Argc = 1 then
  Put_Line("Parsing stdin, terminate with "'D"');
  Psdl_Io.Get(The_Psdl_Component);
  Put_Line("Psdl ADT created for stdin,");
  Put_Line(" Input listing file is left in file 'stdin.lst'");
  -- Expand();
  Psdl_Io.Put(The_Psdl_Component);
-- Put_Line("Expanded Psdl source code is generated form Psdl ADT,");
elsif U_Env.Argc = 2 then
  if U_Env.Argv(1).S = "-help" or U_Env.Argv(1).S = "-h" then
    Put_Line("Usage: expander [input_file] [-o output_file]");
  else
    Psdl_Io.Get(F_Name => U_Env.Argv(1).S,
                Item => The_Psdl_Component);

    -- Expand();
    Psdl_Io.Put(The_Psdl_Component); -- output the expanded PSDL file
  end if;
elsif U_Env.Argc = 4 then
  if U_Env.Argv(2).S = "-o" then
    Put_Line("Parsing "'" & U_Env.Argv(1).S & "' .......");
    Psdl_Io.Get(U_Env.Argv(1).S, U_Env.Argv(3).S, The_Psdl_Component);
    Put_Line("Psdl ADT created for "'" & U_Env.Argv(1).S);
    Put_Line(" Input listing file is left in file "'" &
              U_Env.Argv(1).S & ".lst'"');
    -- Expand();
    Psdl_Io.Put(The_Psdl_Component);
    Put("Expanded Psdl source code is generated form Psdl ADT and left");
    Put_Line("in file "'" & U_Env.Argv(3).S & "'"');
  else

129
Put_line("unknown option; Usage: expander [input_file] [-o output_file]");
end if;
else
Put_line("Usage: expander [input_file] [-o output_file]");
end if;

exception
when Name_Error =>
  Put_line("Error: can't open " & U_Env.Argv(1).S &""");
when Use_Error =>
  Put_line("Error: can't create output file. Permission denied.");
when Psdl_Tokens_Syntax_Error =>
  Put_line("Parsing aborted due to Syntax Error");
when Parser_Semantic_Error =>
  Put_line("Semantic Error, parsing aborted");
end Expander;
APPENDIX E. PACKAGE PSDL_IO

--:............:
-- psdlio.a
--:............:

-------------------------------------------------------------------------------------------------
--
-- Unit name : Aflex specification file for PSDL parser
-- File name : psdli lex.l
-- Author : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Date Created : April 1991
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1.1,
-- Verdist Ada version 6.0 (c)
--

-------------------------------------------------------------------------------------------------
-- Keywords : input/output PSDL program
--
-- Abstract :
-- This file is the package that provides a standard I/O for
-- PSDL programs (This was an easy start to parser business!)

--------------------------------------------------------------------- Revision history ---------------------
--
--$Source: /n/gemini/work/bayram/AYACC/parser/RCS/psdlio.a,v $
--$Revision: 1.4 $
--$Date: 1991/09/24 06:46:48 $
--$Author: bayram $
--

---------------------------------------------------------------------

with Parser, PSDL_Component_Pkg, A_Strings;

package PSDL_IO is
-- procedure GET
--
-- Reads the psdl source file, parses it and creates the PSDL ADT
-- Input file is line numbered and saved into a file
-- input file name .1st in the current directory. So if
-- there is no write permission for that directory, exception
-- Use_Error is raised and program aborts. if the second argument
-- is passed psdl file resulted form PSDL ADT is written into a
-- file with that name.

procedure Get
( F_Name : in String; O_F_Name : in String := "";
 Item : out Psdl_Component_Pkg.Psdl Program )
renames Parser.Get;

-- procedure PUT
--
-- Extract the text representation of PSDL program from
-- the PSDL ADT and outputs as a legal PSDL source file
-- The output is always to standard output, but command line
-- switch when invoking the expander, directs renames the
-- renames the standard output to as the given UNIX file
-- A modification can be done to this procedure in package
-- Psdl_Component_Pkg, (separate procedure put_psdl)
-- to use a file instead of standard output for flexibity
-- The best thing to provide two procedures one for stdout
-- the other for file out, and it is fairly easy to do.
procedure Put
    ( P : in Psdl_Component_Pkg.Psdl_Program )
    renames Psdl_Component_Pkg.Put_Psdl;

end Psdl_IO;
APPENDIX F. SPECIFICATION OF PSDL ADT

---------------
-- psdl_types.a
--:--------------:

-- Keywords : abstract data type, PSDL program
--
-- Abstract :
-- This package is the specification for the PSDL ADT

--------------- Revision history ---------------
--
--$Source: /n/gemini/work/bayram/AYACC/parser/RCS/psdl_types.a,v $
--$Revision: 1.13 $
--$Date: 1991/09/24 04:51:13 $
--$Author: bayram $
--

---------------
--

with PSDL_CONCRETE_TYPE_PKG;
use PSDL_CONCRETE_TYPE_PKG;
with PSDL_GRAPH_PKG;
use PSDL_GRAPH_PKG;
with GENERIC_MAP_PKG;  --defines a generic map type
package PSDL_COMPONENT_PKG is
  -- BY REQUIREMENTS clauses are ignored in this version.
  -- The substructure of expressions is not represented in this version.

  -- Discriminant types.
type COMPONENT_TYPE is (PSDL_OPERATOR, PSDL_TYPE);

type IMPLEMENTATION_TYPE is (ATOMIC, COMPOSITE);

  -- Main types.
type PSDL_COMPONENT
    (CATEGORY : COMPONENT_TYPE := PSDL_OPERATOR;
     GRANULARITY : IMPLEMENTATION_TYPE := COMPOSITE) is private;

-- The initializations make c: psdl_component a l
-- egal variable declaration
-- even though psdl_component is an unconstrained type.
type COMPONENT_PTR is access PSDL_COMPONENT;
subtype OPERATOR is PSDL_COMPONENT; -- (category => psdl_operator).
type OP_PTR is access OPERATOR;

subtype DATA_TYPE is PSDL_COMPONENT; -- (category => psdl_type).
subtype ATOMIC_COMPONENT is PSDL_COMPONENT; -- (granularity => atomic).

subtype ATOMIC_OPERATOR is OPERATOR(CATEGORY => PSDL_OPERATOR,
                                     GRANULARITY => ATOMIC);

subtype COMPOSITE_OPERATOR is OPERATOR(CATEGORY => PSDL_OPERATOR,
                                         GRANULARITY => COMPOSITE);

subtype ATOMIC_TYPE is DATA_TYPE (CATEGORY => PSDL_TYPE,
                                     GRANULARITY => ATOMIC);

subtype COMPOSITE_TYPE is DATA_TYPE (CATEGORY => PSDL_TYPE,
                                     GRANULARITY => COMPOSITE);

  -- needed for generic map package
function Eq(x, y: Psdl_Id) return BOOLEAN;

function Eq(x, y: Component_Ptr) return BOOLEAN;

function Eq(x, y: Op_Ptr) return BOOLEAN;

135
package PSDL_PROGRAM_PKG is
  new GENERIC_MAP_PKG(K KEY => PSDL_ID,
                      RESULT => COMPONENT_PTR,
                      Eq_Key => Eq,
                      Eq_Res => Eq);

type PSDL_PROGRAM is new PSDL_PROGRAM_PKG.MAP;
-- A psdl program is an environment that binds
-- psdl component names
-- to psdl component definitions.
-- The operations on psdl_programs are the same as
-- the operations on maps.

function EMPTY_PSDL_PROGRAM return PSDL_PROGRAM;
-- returns an empty psdl_program.

package OPERATION_MAP_PKG is
  new GENERIC_MAP_PKG(K KEY => PSDL_ID,
                      RESULT => OP_PTR,
                      Eq_Key => Eq,
                      Eq_Res => Eq);

type OPERATION_MAP is new OPERATION_MAP_PKG.MAP;
-- A operation map is an environment that binds
-- psdl operator names
-- to psdl operator definitions.

function EMPTY_OPERATION_MAP return OPERATION_MAP;
-- returns an empty operation_map.

-- exception declarations
INITIAL_STATE_UNDEFINED : exception;
NO_DATA_STRUCTURE : exception;
INPUT_REDECLARED : exception;
OUTPUT_REDECLARED : exception;
STATE_REDECLARED : exception;
INITIAL_VALUE_REDECLARED : exception;
EXCEPTION_REDECLARED : exception;
SPECIFIED_MET_REDEFINED : exception;
NOT_A_SUBCOMPONENT : exception;
PERIOD_REDEFINED : exception;
FINISH_WITHIN_REDEFINED : exception;
MINIMUM_CALLING_PERIOD_REDEFINED : exception;
MAXIMUM_RESPONSE_TIME_REDEFINED : exception;

-- The following exceptions signal failures of explicit runtime checks for violations of subtype constraints. This is needed because Ada does not allow partially constrained types:
-- if any discriminants are constrained, then all must be constrained.

NOT_AN_OPERATOR : exception;
-- Raised by operations on psdl operators that have an actual parameter of type operator with category = psdl_type.

NOT_A_TYPE : exception;
-- Raised by operations on psdl data types that have an actual parameter of type data-type with category = psdloperator.

NOT_AN_ATOMIC_COMPONENT : exception;
-- Raised by operations on atomic components that have an actual parameter of type atomic_component with granularity = composite.

-- operations on all psdl components

function COMPONENTCATEGORY(C : PSDLCOMPONENT) return COMPONENT_TYPE;
-- Indicates whether c is an operator or a type.

function COMPONENTGRANULARITY(C : PSDLCOMPONENT) return IMPLEMENTATION_TYPE;
-- Indicates whether c is atomic or composite.

function NAME(C : PSDLCOMPONENT) return PSDL_ID;
-- Returns the psdl name of the component.
function GENERIC_PARAMETERS(C : PSDL_COMPONENT)  
    return TYPE_DECLARATION; 
-- Returns an empty type_declaration 
-- if no generic parameters are declared.

function KEYWORDS(C : PSDL_COMPONENT) 
    return ID_SET; 
-- Returns an empty set if no keywords are given.

function INFORMAL_DESCRIPTION(C : PSDL_COMPONENT) 
    return TEXT; 
-- Returns an empty string 
-- if no informal description is given.

function AXIOMS(C : PSDL_COMPONENT) 
    return TEXT; 
-- Returns an empty string 
-- if no formal description is given.

-----------------------------------------------
-- operations on psdl operators                --
-----------------------------------------------

function INPUTS(O : OPERATOR) 
    return TYPE_DECLARATION; 
-- Returns an empty type_declaration 
-- if no inputs are declared.

function OUTPUTS(O : OPERATOR) 
    return TYPE_DECLARATION; 
-- Returns an empty type_declaration 
-- if no outputs are declared.

function STATES(O : OPERATOR) 
    return TYPE_DECLARATION; 
-- Returns an empty type_declaration 
-- if no state variables are declared.

function INITIAL_STATE(O : OPERATOR; 
    V : VARIABLE) 
    return EXPRESSION; 
-- Raises initial_state_undefined 
-- if v is not initialized.

function GET_INIT_MAP(O : OPERATOR) 
    return INIT_MAP;
function EXCEPTIONS(O : OPERATOR)
    return ID_SET;
-- Returns an empty set if no exceptions are declared.

function SPECIFIED_MAXIMUM_EXECUTION_TIME(O : OPERATOR)
    return MILLISEC;
-- The maximum execution time given in the specification of o.
-- See also required_maximum_execution_time.
-- Returns zero if no maximum execution time is declared.

procedure ADD_INPUT(STREAM : in PSDL_ID;
    T : in TYPE_NAME;
    O : in out OPERATOR);
-- Adds a binding to the inputs map.
-- Raises input_redeclared if stream is already in inputs(o).

procedure ADD_OUTPUT(STREAM : in PSDL_ID;
    T : in TYPE_NAME;
    O : in out OPERATOR);
-- Adds a binding to the outputs map.
-- Raises output_redeclared if stream is already in outputs(o).

procedure ADD_STATE(STREAM : in PSDL_ID;
    T : in TYPE_NAME;
    O : in out OPERATOR);
-- Adds a binding to the states map.
-- Raises state_redeclared if stream is already in states(o).

procedure ADD_INITIALIZATION(STREAM : in PSDL_ID;
    E : in EXPRESSION;
    O : in out OPERATOR);
-- Adds a binding to the init map.
-- Raises initial_value_redeclared if stream is
-- already bound in the init map.

procedure ADD_EXCEPTION(E : PSDL_ID;
    O : in out OPERATOR);
-- Raises exception_redeclared if stream is
-- already in exceptions(o).

procedure SET_SPECIFIED_MET(MET : MILLISEC;
    O : in out OPERATOR);
-- Raises specified_met_redefined if specified_met
-- is already non-zero.
Operations on all atomic PSDL components.

-- Create an atomic operator
function ADA_NAME(A : ATOMIC_COMPONENT) return ADA_ID;

function MAKE_ATOMIC_OPERATOR
    (PSDL_NAME : PSDL_ID;
     ADA_NAME : ADA_ID;
     GEN_PAR : TYPEDECLARATION := EMPTY_TYPE_DECLARATION;
     KEYWORDS : IDSET := EMPTY_IDSET;
     INFORMAL_DESCRIPTION, AXIOMS : TEXT := EMPTY_TEXT;
     INPUT, OUTPUT, STATE : TYPEDECLARATION := EMPTYTYPEDECLARATION;
     INITIALIZATION_MAP : INITMAP := EMPTYINITMAP;
     EXCEPTIONS : ID_SET := EMPTY_ID_SET;
     SPECIFIED_MET : MILLISEC := 0)
return ATOMIC_OPERATOR;

-- Create an atomic type
function MAKE_ATOMIC_TYPE
    (PSDL_NAME : PSDL_ID;
     ADA_NAME : ADA_ID;
     MODEL : TYPEDECLARATION;
     OPERATIONS : OPERATION_MAP;
     GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
     KEYWORDS : ID_SET := EMPTY_ID_SET;
     INFORMAL_DESCRIPTION, AXIOMS : TEXT := EMPTY_TEXT)
return ATOMIC_TYPE;

Operations on composite operators.

function GRAPH(CO : COMPOSITE_OPERATOR)
    return PSDL_GRAPH;

function STREAMS(CO : COMPOSITE_OPERATOR)
    return TYPE_DECLARATION;
function TIMERS(CO : COMPOSITE_OPERATOR)
  return ID_SET;
-- Returns an empty set if no timers are declared.

function GET_TRIGGERTYPE
  (COMPONENT_OP : PSDL_ID;
   CO : COMPOSITE_OPERATOR)
  return TRIGGERTYPE;
-- Returns the type of triggering condition for
-- the given component operator.
-- Derived from the control constraints,
-- result is "none" if no trigger.
-- Raises not_a_subcomponent if component_op
-- is not a vertex in graph(co).

function EXECUTION_GUARD
  (COMPONENT_OP : PSDL_ID;
   CO : COMPOSITE_OPERATOR)
  return EXPRESSION;
-- Returns the IF part of the triggering condition for the
-- component operator, "true" if no triggering
-- condition is given.
-- Raises not_a_subcomponent if component_op is
-- not a vertex in graph(co).

function OUTPUT_GUARD
  (COMPONENT_OP,
   OUTPUT_STREAM : PSDL_ID;
   CO : COMPOSITE_OPERATOR)
  return EXPRESSION;
-- Returns the IF part of the output constraint
-- for the component operator
-- for each output stream mentioned in the constraint,
-- "true" if no output constraint with the stream is given.
-- Raises not_a_subcomponent if component_op is not a
-- vertex in graph(co).

function EXCEPTION_TRIGGER
  (COMPONENT_OP,
   EXCEPTION_NAME : PSDL_ID;
   CO : COMPOSITE_OPERATOR)
  return EXPRESSION;
-- Returns the IF part of the exception trigger for
-- the component operator
-- and exception name, "true" if there is an unconditional
-- exception trigger
-- in the control constraints, "false" if no exception
-- trigger is given
-- for component_op in the control constraints.
-- Raises not_a_subcomponent if component_op
-- is not a vertex in graph(co).

function TIMEROPERATION
  (COMPONENT_OP : PSDL_ID;
   CO      : COMPOSITE_OPERATOR)
  return TIMER_OP_SET;

-- Returns the timer_op part of the control
-- constraint for the
-- component operator, "none" if no timer
-- operation is given.
-- Raises not_a_subcomponent if component_op
-- is not a vertex in graph(co).

function PERIOD
  (COMPONENT_OP : PSDL_ID;
   CO      : COMPOSITE_OPERATOR)
  return MILLISEC;

-- Returns the period part of the control constraint for the
-- component operator, zero if no period is given.
-- Raises not_a_subcomponent if component_op is not
-- a vertex in graph(co).

function FINISH_WITHIN
  (COMPONENT_OP : PSDL_ID;
   CO      : COMPOSITE_OPERATOR)
  return MILLISEC;

-- Returns the finish_within part of the control
-- constraint for the
-- component operator, zero if no finish_within is given.
-- Raises not_a_subcomponent if component_op is
-- not a vertex in graph(co).

function MINIMUM_CALLING_PERIOD
  (COMPONENT_OP : PSDL_ID;
   CO      : COMPOSITE_OPERATOR)
  return MILLISEC;

-- Returns the minimum calling period part of the
-- control constraint for the
-- component operator, zero if no minimum calling
function MAXIMUM_RESPONSE_TIME
    (COMPONENT_OP : PSDL_ID;
     CO : COMPOSITE_OPERATOR)
    return MILLISEC;

-- Returns the maximum_response_time part of the
-- control constraint for the
-- component operator, zero if no
-- maximum_response_time is given.
-- Raises not_a_subcomponent if component_op
-- is not a vertex in graph(co).

function REQUIRED_MAXIMUM_EXECUTION_TIME
    (COMPONENT_OP : PSDL_ID;
     CO : COMPOSITE_OPERATOR)
    return MILLISEC;

-- Returns the maximum execution time part of the
-- control constraint for the
-- component operator, zero if no maximum execution time is given
-- in the graph. This includes time used by the implementations
-- of the control constraints and stream operations, and should be
-- greater than or equal to the specified_maximum_execution_time for
-- the component operator if it is defined (greater than zero).
-- Raises not_a_subcomponent if component_op is not a vertex in
-- graph(co).

function LATENCY
    (PRODUCER_OP,
     CONSUMER_OP,
     STREAM_NAME : PSDL_ID;
     CO : COMPOSITE_OPERATOR)
    return MILLISEC;

-- Returns the timing label on the edge from the producer operator
-- to the consumer operator in the graph, zero if none.
-- Represents the maximum data transmission delay allowed for
-- the data stream, for modeling network delay in
-- distributed systems.
-- Raises not_a_subcomponent if component_op is not a vertex
-- in graph(co).
-- Creates a composite operator
function MAKE_COMPOSITE_OPERATOR
(NAME : PSDL_ID;
GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
KEYWORDS : ID_SET := EMPTY_ID_SET;
INFORMAL_DESCRIPTION, AXIOMS : TEXT := EMPTY_TEXT;
INPUT, OUTPUT, STATE : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
INITIALIZATION_MAP : INIT_MAP := EMPTY_INIT_MAP;
EXCEPTIONS : ID_SET := EMPTY_ID_SET;
SPECIFIED_MET : MILLISEC := 0;
GRAPH : PSDL_GRAPH := EMPTY_PSDL_GRAPH;
STREAMS : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
TIMERS : ID_SET := EMPTY_ID_SET;
TRIGGER : TRIGGER_MAP := EMPTY_TRIGGER_MAP;
EXEC_GUARD : EXEC_GUARD_MAP := EMPTY_EXEC_GUARD_MAP;
OUT_GUARD : OUT_GUARD_MAP := EMPTY_OUT_GUARD_MAP;
EXCEPTRIGGER : EXCEPTRIGGER_MAP := EMPTY_EXCEPTRIGGER_MAP;
TIMEROP : TIMEROP_MAP := EMPTY_TIMEROP_MAP;
PER, FW, MCP, MRT : TIMING_MAP := EMPTY_TIMING_MAP;
impl_desc := emptytext)
return COMPOSITE_OPERATOR;

procedure ADD_VERTEX(OPNAME in PSDL_ID;
      CO : in out COMPOSITE_OPERATOR;
      MET : in MILLISEC := 0);

procedure ADD_EDGE(X, Y in PSDL_ID;
      STREAM : in PSDL_ID;
      CO : in out COMPOSITE_OPERATOR;
      LATENCY : in MILLISEC := 0);

procedure ADD_STREAM(S in PSDL_ID;
      T : in TYPE_NAME;
      CO : in out COMPOSITE_OPERATOR);
procedure ADD_TIMER(T : in PSDL_ID;
                      CO : in out COMPOSITE_OPERATOR);

procedure SET_TRIGGER_TYPE(OP_ID : in PSDL_ID;
                           T : in TRIGGER_TYPE;
                           CO : in out COMPOSITE_OPERATOR);

procedure SET_EXECUTION_GUARD(OP_ID : in PSDL_ID;
                               E : in EXPRESSION;
                               CO : in out COMPOSITE_OPERATOR);

procedure SET_OUTPUT_GUARD(OP_ID : in PSDL_ID;
                           STREAM : in PSDL_ID;
                           E : in EXPRESSION;
                           CO : in out COMPOSITE_OPERATOR);

procedure SET_EXCEPTION_TRIGGER(OP_ID : in PSDL_ID;
                                EXCEP : in PSDL_ID;
                                E : in EXPRESSION;
                                CO : in out COMPOSITE_OPERATOR);

procedure ADD_TIMER_OP(OP_ID,
                       TIMER_ID : in PSDL_ID;
                       TOP : in TIMER_OP_ID;
                       E : in EXPRESSION;
                       CO : in out COMPOSITE_OPERATOR);

procedure SET_PERIOD(OP_ID : in PSDL_ID;
                    P : in MILLISEC;
                    CO : in out COMPOSITE_OPERATOR);

procedure SET_FINISH_WITHIN(OP_ID : in PSDL_ID;
                           FW : in MILLISEC;
                           CO : in out COMPOSITE_OPERATOR);

procedure SET_MINIMUM_CALLING_PERIOD
          (OP_ID : in PSDL_ID;
           MCP : in MILLISEC;
           CO : in out COMPOSITE_OPERATOR);

-- Raises period_redefined if the period is non-zero.

-- Raises finish_within_redefined if the finish_within
-- is non-zero.

-- Raises minimum_calling_period_redefined if the
-- minimum_calling_period is non-zero.
procedure SET_MAXIMUM_RESPONSE_TIME
  (OP_ID : in PSDL_ID;
   MRT : in MILLISEC;
   CO  : in out COMPOSITE_OPERATOR);
-- Raises maximum_response_time_redefined if the
-- maximum_response_time is non-zero.

-- Operations on all psdl types.
function MODEL(T : DATA_TYPE)
    return TYPE_DECLARATION;
-- Returns the conceptual representation declared in
-- the specification part,
-- empty if not given.

function OPERATIONS(T : DATA_TYPE)
    return OPERATION_MAP;
-- Returns an environment binding operation names
-- to operation definitions,
-- an empty map if the type does not define any operations.

-- Operations on composite psdl data types.

function DATA_STRUCTURE(T : COMPOSITE_TYPE)
    return TYPE_NAME;
-- Returns the data structure declared in the
-- psdl implementation part,
-- raises no_data_structure if the type is
-- implemented in Ada.

-- Create a composite type
function MAKE_COMPOSITE_TYPE
    (NAME : PSDL_ID;
    MODEL : TYPE_DECLARATION;
    DATA_STRUCTURE : TYPE_NAME;
    OPERATIONS : OPERATION_MAP;
    GEN_PAR : TYPE_DECLARATION
        := EMPTY_TYPE_DECLARATION;
    KEYWORDS : ID_SET := EMPTY_ID_SET;
    INFORMAL_DESCRIPTION,
    AXIOMS : TEXT := EMPTY_TEXT)
    return COMPOSITE_TYPE;

-- print out the psdl program
procedure PUT_PSDL(P: IN PSDL_PROGRAM);

private
    type PSDL_COMPONENT
    (CATEGORY : COMPONENT_TYPE := PSDL_OPERATOR;

147
GRANULARITY : IMPLEMENTATION_TYPE := COMPOSITE) is

record
  NAME : PSDL_ID;
  GEN_PAR : TYPE_DECLARATION;
  KEYW : ID_SET;
  INF_DESC, AX : TEXT;
  case CATEGORY is
    when PSDLOPERATOR =>
      INPUT, OUTPUT, STATE
      INIT
      EXCEP
      SMET
    case GRANULARITY is
      when ATOMIC =>
        O_ADA_NAME
      when COMPOSITE =>
        G
        STR
        TIM
        TRIG
        EG
        OG
        ET
        TIM_OP
        PER, FW, MCP, MRT
        IMPL_DESC
    end case;
  when PSDL_TYPE =>
    MDL
    OPS
    case GRANULARITY is
      when ATOMIC =>
        T_ADA_NAME
      when COMPOSITE =>
        DATA_STR
    end case;
  end case;
end record;
end PSDL_COMPONENT_PKG;
APPENDIX G. IMPLEMENTATION OF PSDL ADT

---:------------------:
-- psdl_typeb.a
---:------------------:

-- Unit name : Implementation of PSDL ADT
-- File name : psdl_typeb.a
-- Author : Valdis Berzins (berzins@taurus.cs.nps.navy.mil)
-- Date Created : December 1990
-- Modified by : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Last Update : [Tue Sep 24 00:04:52 1991 - bayram]
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1.1,
--       Verdix Ada version 6.0 (c)
--
-- Keywords : abstract data type, PSDL program
--
-- Abstract
--   This package is the implementation for the PSDL ADT

----------------------------- Revision history -----------------------------
--
--$Source:
--/n/gemini/work/bayram/AYACC/parser/psdl_ada.lib/RCS/psdl_typeb.a,v $
--$Revision: 1.15 $
--$Date: 1991/09/24 08:02:15 $
--$Author: bayram $
--

with text_io, a_strings;
use text_io;

package body PSDL_COMPONENT_PKG is

-- the following functions are provided for`
-- instations of generic packages (map, set, sequence)
function Eq(x, y: Psdl_Id) return BOOLEAN is
begin
  return (X.S = Y.S);
end Eq;

function Eq(x, y: Component_Ptr) return BOOLEAN is
begin
  return (X.Name.s = Y.Name.s);
end Eq;

function Eq(x, y: Op_Ptr) return BOOLEAN is
begin
  return (X.Name.s = Y.Name.s);
end Eq;

-- returns an empty operation_map.
function EMPTY_OPERATION_MAP return OPERATION_MAP is

  M : OPERATION_MAP;

begin
  CREATE(null, M); -- default value of the map is the null pointer
  return M;
end EMPTY_OPERATION_MAP;

-- returns an empty psdl_program.
function EMPTY_PSDL_PROGRAM return PSDL_PROGRAM is

P : PSDL_PROGRAM;

begin
  CREATE(null, P); -- default value is the null pointer
  return P;
end EMPTY_PSDL_PROGRAM;

--************** FOR REFERENCE ONLY ******************
--************** EXCEPTION LISTING ******************
--* initial_state_undefined: exception;
--* no_data_structure: exception;
--* input_redeclared: exception;
--* output_redeclared: exception;
--* state_redeclared: exception;
--* initial_value_redeclared: exception;
exception_redeclared: exception;
specified_met_redefined: exception;
not_a_subcomponent: exception;
period_redefined: exception;
finish_within_redefined: exception;
minimum_calling_period_redefined: exception;
maximum_response_time_redefined: exception;
-- The following exceptions signal failures
-- of explicit runtime
-- checks for violations of subtype constraints.
-- This is needed because Ada does not allow
-- partially constrained types:
-- if any discriminants are constrained,
-- then all must be constrained.
not_an_operator: exception;
-- Raised by operations on psdl operators that
-- have an actual parameter
-- of type operator with category = psdl_type.
not_a_type: exception;
-- Raised by operations on psdl data types that
-- have an actual parameter
-- of type data_type with category = psdl_operator.
not_an_atomic_component: exception;
-- Raised by operations on atomic components that
-- have an actual parameter
-- of type atomic_component with granularity = composite.
--------------------------- END EXCEPTIONS ---------------------------

-- operations on all psdl components

-- Indicates whether c is an operator or a type.
function COMPONENT_CATEGORY(C : PSDL_COMPONENT)
    return COMPONENT_TYPE is
begin
    return C.CATEGORY;
end COMPONENT_CATEGORY;

-- Indicates whether c is atomic or composite.
function COMPONENT_GRANULARITY(C : PSDL_COMPONENT)
    return IMPLEMENTATION_TYPE is
begin
    return C.GRANULARITY;
end COMPONENT_GRANULARITY;

-- Returns the psdl name of the component.
function NAME(C : PSDL_COMPONENT)
    return PSDL_ID is
begin
    return C.NAME;
end NAME;

-- Returns an empty type declaration if no generic parameters are declared
function GENERIC_PARAMETERS(C : PSDL_COMPONENT)
    return TYPE_DECLARATION is
begin
    return C.GENPAR;
end GENERIC_PARAMETERS;

-- Returns an empty set if no keywords are given.
function KEYWORDS(C : PSDL_COMPONENT)
    return ID_SET is
begin
    return C.KEYW;
end KEYWORDS;

-- Returns an empty string if no informal description is given.
function INFORMAL_DESCRIPTION(C : PSDL_COMPONENT)
    return TEXT is
begin
    return C.INF_DESC;
end INFORMAL_DESCRIPTION;

-- Returns an empty string if no formal description is given.
function AXIOMS(C : PSDL_COMPONENT)
    return TEXT is
begin
  return C.AX;
end AXIOMS;

-- /* operations on psdl operators */

-- Returns an empty type_declaration if no inputs are declared.
function INPUTS(O : OPERATOR)
  return TYPE_DECLARATION is
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.INPUT;
  end if;
end INPUTS;

function OUTPUTS(O : OPERATOR)
  return TYPE_DECLARATION is
-- Returns an empty type_declaration if no outputs are declared.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.OUTPUT;
  end if;
end OUTPUTS;

function STATES(O : OPERATOR)
  return TYPE_DECLARATION is
-- Returns an empty type_declaration if no state variables are declared.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.STATE;
  end if;
end STATES;
end if;
end STATES;

function INITIAL_STATE(O : OPERATOR; V : VARIABLE)
return EXPRESSION is
  -- Raises initial_state_undefined if v is not initialized.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  elsif not INIT_MAP_PKG.MEMBER(V, O.INIT) then
    raise INITIAL_STATE_UNDEFINED;
  else
    return INIT_MAP_PKG.FETCH(O.INIT, V);
  end if;
end INITIAL_STATE;

function GET_INIT_MAP(O : OPERATOR) return INIT_MAP is
  -- Returns an empty init_map if no
  -- initializations are declared.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.Init;
  end if;
end GET_INIT_MAP;

function EXCEPTIONS(O : OPERATOR)
return IDSET is
  -- Returns an empty set if no exceptions are declared.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.EXCEP;
  end if;
end EXCEPTIONS;

function SPECIFIED_MAXIMUM_EXECUTION_TIME(O : OPERATOR)
return MILISEC is
-- The maximum execution time given in the specification of o.
-- See also required_maximum_execution_time.
-- Returns zero if no maximum execution time is declared.
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  else
    return O.SMET;
  end if;
end SPECIFIED_MAXIMUM_EXECUTION_TIME;

procedure ADD_INPUT
  (STREAM: in PSDL_ID;
   T   : in TYPE_NAME;
   O   : in out OPERATOR) is
-- Adds a binding to the inputs map.
-- Raises input_redeclared if stream is already in inputs(o).
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  elsif TYPE_DECLARATION_PKG.MEMBER(STREAM, O.INPUT) then
    raise INPUT_REDECLARED;
  else
    TYPE_DECLARATION_PKG.BIND(STREAM, T, O.INPUT);
  end if;
end ADD_INPUT;

procedure ADD_OUTPUT(STREAM: in PSDL_ID;
                      T   : in TYPE_NAME;
                      O   : in out OPERATOR) is
-- Adds a binding to the outputs map.
-- Raises output_redeclared if stream is already in outputs(o).
begin
  if O.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  elsif TYPE_DECLARATION_PKG.MEMBER(STREAM, O.OUTPUT) then
    raise OUTPUT_REDECLARED;
  else
    TYPE_DECLARATION_PKG.BIND(STREAM, T, O.OUTPUT);
  end if;
end ADD_OUTPUT;
procedure ADD_STATE(STREAM in PSDL_ID;
    T   in TYPE_NAME;
    O   in out OPERATOR) is

-- Adds a binding to the states map.
-- Raises state_redeclared if stream is already in states(o).
begin
    if O.CATEGORY /= PSDL_OPERATOR then
        raise NOT_AN_OPERATOR;
    elsif TYPE_DECLARATION_PKG.MEMBER(STREAM, O.STATE) then
        raise STATE_REDECLARED;
    else
        TYPE_DECLARATION_PKG.BIND(STREAM, T, O.STATE);
    end if;
end ADD_STATE;

procedure ADD_INITIALIZATION(STREAM in PSDL_ID;
                            E   in EXPRESSION;
                            O   in out OPERATOR) is

-- Adds a binding to the init map.
-- Raises initial_value_redeclared if stream is
-- already bound in the init map.
begin
    if O.CATEGORY /= PSDL_OPERATOR then
        raise NOT_AN_OPERATOR;
    elsif INIT_MAP_PKG.MEMBER(STREAM, O.INIT) then
        raise INITIAL_VALUE_REDECLARED;
    else
        INIT_MAP_PKG.BIND(STREAM, E, O.INIT);
    end if;
end ADD_INITIALIZATION;

procedure ADD_EXCEPTION(E  in PSDL_ID;
                        O  in out OPERATOR) is

-- Raises exception_redeclared if stream is already in
-- exceptions(o).
begin
    if O.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
elsif ID_SET_PKG.MEMBER(E, O.EXCEP) then
    raise EXCEPTION_REDECLARED;
else
    ID_SET_PKG.ADD(E, O.EXCEP);
end if;
end ADD_EXCEPTION;

procedure SET_SPECIFIED_MET(MET : MILLISEC;
    O : in out OPERATOR) is
-- Raises specified_met_redefined if
-- specified_met is already non-zero.
begin
    if O.CATEGORY /= PSDL OPERATOR then
        raise NOT_AN_OPERATOR;
    elsif O.SMET /= 0 then
        raise INPUT_REDECLARED;
    else
        O.SMET := MET;
    end if;
end SET_SPECIFIED_MET;

-- Operations on all atomic psdl components.

function ADA_NAME(A : ATOMIC_COMPONENT)
    return ADA_ID is
begin
    case A.GRANULARITY is
    when ATOMIC =>
        case A.CATEGORY is
        when PSDL_OPERATOR =>
            return A.O_ADANAME;
        when PSDL_TYPE =>
            return A.T_ADANAME;
        end case;
    when COMPOSITE =>
        raise NOT_AN_ATOMIC_COMPONENT;
    end case;
end ADA_NAME;

function MAKE_ATOMIC_OPERATOR
    (PSDL_NAME : PSDL_ID;
ADANAME : ADA_ID;
GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
KEYWORDS : ID_SET := EMPTY_ID_SET;
INFORMAL_DESCRIPTION : TEXT := EMPTY_TEXT;
AXIOMS : TEXT := EMPTY_TEXT;
INPUT, OUTPUT, STATE : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
INITIALIZATION_MAP : INIT_MAP := EMPTY_INIT_MAP;
EXCEPTIONS : ID_SET := EMPTY_ID_SET;
SPECIFIED_MET : MILLISEC := 0)
return ATOMIC_OPERATOR is

-- Create an atomic operator.

X : ATOMIC_OPERATOR;

begin
  X.NAME := PSDL_NAME;
  X.O_ADANAME := ADANAME;
  X.GEN_PAR := GEN_PAR;
  X.KEYW := KEYWORDS;
  X.INF_DESC := INFORMAL_DESCRIPTION;
  X.AX := AXIOMS;
  X.INPUT := INPUT;
  X.OUTPUT := OUTPUT;
  X.STATE := STATE;
  X.INIT := INITIALIZATION_MAP;
  X.EXCEP := EXCEPTIONS;
  X.SMET := SPECIFIED_MET;

  return X;
end MAKE_ATOMIC_OPERATOR;

function MAKE_ATOMIC_TYPE
(PSDL_NAME : PSDL_ID;
 ADA_NAME : ADA_ID;
 MODEL : TYPE_DECLARATION;
 OPERATIONS : OPERATION_MAP;
 GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
 KEYWORDS : ID_SET := EMPTY_ID_SET;
 INFORMAL_DESCRIPTION, AXIOMS : TEXT := EMPTY_TEXT)
return ATOMIC_TYPE is

-- Create an atomic type.
X : ATOMIC_TYPE;

begin
X.NAME := PSDL_NAME;
X.T_ADA_NAME := ADA_NAME;
X.MDL := MODEL;
X.OPS := OPERATIONS;
X.GEN_PAR := GEN_PAR;
X.KEYW := KEYWORDS;
X.INF_DESC := INFORMAL_DESCRIPTION;
X.AX := AXIOMS;

return X;
end MAKE_ATOMIC_TYPE;

-- Operations on composite operators.

function GRAPH(CO : COMPOSITE_OPERATOR) return PSDL_GRAPH is
begin
if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;
return CO.G;
end GRAPH;

function STREAMS(CO : COMPOSITE_OPERATOR) return TYPE_DECLARATION is
-- Returns an empty type_declaration if no local streams are declared.
begin
if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;
return CO.STR;
end STREAMS;

function TIMERS(CO : COMPOSITE_OPERATOR) return ID_SET is
-- Returns an empty set if no timers are declared.
begin
if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;

return CO.TIM;
end TIMERS;

function GET_TRIGGER_TYPE(COMPONENT_OP : PSDL_ID;
                          CO : COMPOSITE_OPERATOR)
    return TRIGGER_TYPE is
    -- Returns the type of triggering condition for the
    -- given component operator.
    -- Derived from the control constraints,
    -- result is "none" if no trigger.
    -- Raises not_a_subcomponent if component_op is
    -- not a vertex in graph(co).
    T_RECORD: TRIGGER_RECORD;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        if not HAS_VERTEX(COMPONENT_OP, CO.G) then
            raise NOT_A_SUBCOMPONENT;
        elsif (not TRIGGER_MAP_PKG.MEMBER(COMPONENT_OP, CO.TRIG)) then
            return NONE;
        else
            T_RECORD := TRIGGER_MAP_PKG.FETCH(CO.TRIG, COMPONENT_OP);
            return T_RECORD.TT;
        end if;
    end GET_TRIGGER_TYPE;

function EXECUTION_GUARD(COMPONENT_OP : PSDL_ID;
                          CO : COMPOSITE_OPERATOR)
    return EXPRESSION is
    -- Returns the IF part of the triggering condition for the
    -- component operator, "true" if no triggering
    -- condition is given.
    -- Raises not_a_subcomponent if component_op is
    -- not a vertex in graph(co).
    NO_TRIGGERING : EXPRESSION;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        NO_TRIGGERING.S := "true";
        if not HAS_VERTEX(COMPONENT_OP, CO.G) then
            raise NOT_A_SUBCOMPONENT;
        elsif (not EXEC_GUARD_MAP_PKG.MEMBER(COMPONENT_OP, CO.EG)) then
            return NO_TRIGGERING;
else
    return EXEC_GUARD_MAP_PKG.FETCH(CO.EG, COMPONENT_OP);
end if;
end EXECUTION_GUARD;

function OUTPUT_GUARD(COMPONENT_OP,
    OUTPUT_STREAM : PSDL_ID;
    CO : COMPOSITE_OPERATOR)
    return EXPRESSION is

    -- Returns the IF part of the output constraint
    -- for the component operator
    -- for each output stream mentioned in the constraint,
    -- "true" if no output constraint with the stream is given.
    -- Raises not_a_subcomponent if component_op is
    -- not a vertex in graph(co).
    TEMP_ID : OUTPUT_ID;
    NO_CONSTRAINT : EXPRESSION;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        NO_CONSTRAINT.S := "true";
        TEMP_ID.OP := COMPONENT_OP;
        TEMP_ID.STREAM := OUTPUT_STREAM;
        if not HAS_VERTEX(COMPONENT_OP, CO.G) then
            raise NOT_A_SUBCOMPONENT;
        elsif (not OUT_GUARD_PKG.MEMBER(TEMP_ID, CO.OG)) then
            return NO_CONSTRAINT;
        else
            return OUT_GUARD_PKG.FETCH(CO.OG, TEMP_ID);
        end if;
    end OUTPUT_GUARD;

function EXCEPTION_TRIGGER(COMPONENT_OP,
    EXCEPTION_NAME : PSDL_ID;
    CO : COMPOSITE_OPERATOR)
    return EXPRESSION is

    -- Returns the IF part of the exception trigger
    -- for the component operator
    -- and exception name, "true" if there is an
    -- unconditional exception trigger
    -- in the control constraints, "false" if no
    -- exception trigger is given
    -- for component_op in the control constraints.
    -- Raises not_a_subcomponent if component_op is
begin
  if CO.CATEGORY /= PSDLOPERATOR then
    raise NOTANOPERATOR;
  end if;

  UNCONDITIONAL_EXCEPTION := "true";
  NO_EXCEPTION := "false";

  TEMPID.OP := COMPONENTOP;
  TEMPID.EXCEPT := EXCEPTIONNAME;

  if not HAS_VERTEX(COMPONENTOP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  elsif (not EXCEPTRIGGERMAPPKG.MEMBER(TEMPID, CO.ET)) then
    return NOEXCEPTION;
  else
    return EXCEPTRIGGERMAPPKG.FETCH(CO.ET, TEMPID);
  end if;
end EXCEPTIONTRIGGER;

function TIMEROPERATION(COMPONENTOP : PSDLID; CO : COMPOSITEOPERATOR)
return TIMER_OP_SET is
begin
  if CO.CATEGORY /= PSDLOPERATOR then
    raise NOTANOPERATOR;
  elsif not HAS_VERTEX(COMPONENTOP, CO.G) then
    raise NOTASUBCOMPONENT;
  else
    return TIMER_OP_MAP_PKG.FETCH(CO.TIM_OP, COMPONENTOP);
  end if;
end TIMEROPEPkTION;

function PERIOD(COMPONENTOP : PSDLID; CO : COMPOSITEOPERATOR)
return MILLISEC is
begin
  return EXCEPTRIGGER_MAPPKG.FETCH(CO.ET, TEMP_ID);
end PERIOD;
-- Raises not_a_subcomponent if component_op is
-- not a vertex in graph(co).

begin
  if not HAS_VERTEX(COMPONENT_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return TIMING_MAP_PKG.FETCH(CO.PER, COMPONENT_OP);
  end if;
end PERIOD;

function FINISH_WITHIN(COMPONENT_OP : PSDL_ID;
                        CO : COMPOSITE_OPERATOR)
  return MILLISEC is
  -- Returns the finish_within part of the control
  -- constraint for the
  -- component operator, zero if no finish_within is given.
  -- Raises not_a_subcomponent if component_op is
  -- not a vertex in graph(co).

begin
  if not HAS_VERTEX(COMPONENT_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return TIMING_MAP_PKG.FETCH(CO.FW, COMPONENT_OP);
  end if;
end FINISH_WITHIN;

function MINIMUM_CALLING_PERIOD(COMPONENT_OP : PSDL_ID;
                                 CO : COMPOSITE_OPERATOR)
  return MILLISEC is
  -- Returns the minimum calling period
  -- part of the control constraint for the
  -- component operator, zero if no minimum calling
  -- period is given.
  -- Raises not_a_subcomponent if component_op
  -- is not a vertex in graph(co).

begin
  if not HAS_VERTEX(COMPONENT_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return TIMING_MAP_PKG.FETCH(CO.MCP, COMPONENT_OP);
  end if;
end MINIMUM_CALLING_PERIOD;
function MAXIMUM_RESPONSE_TIME(COMPONENT_OP : PSDL_ID; CO : COMPOSITEOPERATOR)
return MILLISEC is
-- Returns the maximum response time part of the
-- control constraint for the
-- component operator, zero if no
-- maximum response time is given.
-- Raises not a subcomponent if component_op is
-- not a vertex in graph(co),
begin
  if not HAS_VERTEX(COMPONENT_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return TIMING_MAP_PKG.FETCH(CO.MRT, COMPONENT_OP);
  end if;
end MAXIMUM_RESPONSE_TIME;

function REQUIRED_MAXIMUM_EXECUTION_TIME(COMPONENT_OP : PSDL_ID;
                                          CO : COMPOSITEOPERATOR)
return MILLISEC is
-- Returns the maximum execution time
-- part of the control constraint for the
-- component operator, zero if no maximum
-- execution time is given
-- in the graph. This includes time used by the implementations
-- of the control constraints and stream operations,
-- and should be
-- greater than or equal to the
-- specified maximum execution time for
-- the component operator if it is defined (greater than zero).
-- Raises not a subcomponent if component_op is
-- not a vertex in graph(co).
begin
  if not HAS_VERTEX(COMPONENT_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return 0;  -- just a stub
  end if;
end REQUIRED_MAXIMUM_EXECUTION_TIME;

function LATENCY(PRODUCER_OP,
CONSUMER_OP,  
STREAM_NAME : PSDL_ID;  
CO : COMPOSITE_OPERATOR)  
return MILLISEC is

-- Returns the timing label on the edge from the producer operator to the consumer operator in the graph, zero if none.
-- Represents the maximum data transmission delay allowed for the data stream, for modeling network delay in distributed systems.
-- Raises not_a_subcomponent if component_op is not a vertex in graph(co).

begin
  if not HAS_VERTEX(PRODUCER_OP, CO.G) or not HAS_VERTEX(CONSUMER_OP, CO.G) then
    raise NOT_A_SUBCOMPONENT;
  else
    return LATENCY(PRODUCER_OP, CONSUMER_OP, STREAM_NAME, CO.G);
  end if;
end LATENCY;

function MAKE_COMPOSITE_OPERATOR
(NAME : PSDL_ID;  
GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;  
KEYWORDS : ID_SET := EMPTY_ID_SET;  
INFORMAL_DESCRIPTION : TEXT := EMPTY_TEXT;  
AXIOMS : TEXT := EMPTY_TEXT;  
INPUT, OUTPUT, STATE : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;  
INITIALIZATION_MAP : INIT_MAP := EMPTY_INIT_MAP;  
EXCEPTIONS : ID_SET := EMPTY_ID_SET;  
SPECIFIED_MILLISEC : MILLISEC := 0;  
GRAPH : PSDL_GRAPH := EMPTY_PSDL_GRAPH;  
STREAMS : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;  
TIMERS : ID_SET := EMPTY_ID_SET;  
TRIGGER : TRIGGER_MAP := EMPTY_TRIGGER_MAP;  
EXEC_GUARD : EXEC_GUARD_MAP := EMPTY_EXEC_GUARD_MAP;  
OUT_GUARD : OUT GUARD_MAP;

165
EXCEPTRIGGER := EXCEPTRIGGER_MAP;

EXCEPTRIGGER := EXCEPTRIGGER_MAP;

TIMER_OP := TIMER_OP_MAP;

PER, FW, MCP, MRT := TIMING_MAP;

IMPL_DESC := TEXT := EMPTY_TEXT);

return COMPOSITE_OPERATOR is

-- Create a composite operator.

X : COMPOSITE_OPERATOR;

begin

X.name := name;
X.gen_par := gen_par;
X.keyw := keywords;
X.inf_desc := informal_description;
X.ax := axioms;
X.input := input;
X.output := output;
X.state := state;
X.init := initialization_map;
X.excep := exceptions;
X.smet := specified_met;
X.g := graph;
X.str := streams;
X.tim := timers;
X.trig := trigger;
X.eg := exec_guard;
X.og := out_guard;
X.et := exceptrigger;
X.tim_op := timer_op;
X.per := per;
X.fw := fw;
X.mcp := mcp;
X.mrt := mrt;
X.impl_desc := impl_desc;
return X;

end MAKE_COMPOSITE_OPERATOR;

procedure ADD_VERTEX(OPNAME in PSDL_ID;
CO in out COMPOSITE_OPERATOR;
MET in MILLISEC := 0) is
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  end if;
  CO.G := PSDL_GRAPH_PKG.ADD_VERTEX(OPNAME, CO.G, MET);
end ADD_VERTEX;

procedure ADD_EDGE(X, Y : in PSDL_ID;
  STREAM : in PSDL_ID;
  CO : in out COMPOSITE_OPERATOR;
  LATENCY : in MILLISEC := 0) is
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  end if;
  CO.G := PSDL_GRAPH_PKG.ADD_EDGE(X, Y, STREAM, CO.G, LATENCY);
end ADD_EDGE;

procedure ADD_STREAM(S : in PSDL_ID;
  T : in TYPE_NAME;
  CO : in out COMPOSITE_OPERATOR) is
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  end if;
  TYPE_DECLARATION_PKG BIND(S, T, CO.STR);
end ADD_STREAM;

procedure ADD_TIMER(T : in PSDL_ID;
  CO : in out COMPOSITE_OPERATOR) is
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;
ID_SET_PKG.ADD(T, CO.TIM);
end ADD_TIMER;

procedure SET_TRIGGER_TYPE(OP_ID : in PSDL_ID;
T : in TRIGGER_TYPE;
CO : in out COMPOSITE_OPERATOR) is
T_RECORD : TRIGGER_RECORD;
begin
if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;
T_RECORD.TT := T;
T_RECORD STREAMS := EMPT"_ID_SET;
TRIGGER_MAP_PKG.BIND(OP_ID, T_RECORD, CO.TRIG);
end SET_TRIGGER_TYPE;

procedure SET_EXECUTION_GUARD(OP_ID : in PSDL_ID;
E : in EXPRESSION;
CO : in out COMPOSITE_OPERATOR) is
begin
if CO.CATEGORY /= PSDL_OPERATOR then
raise NOT_AN_OPERATOR;
end if;
EXEC_GUARD_MAP_PKG.BIND(OP_ID, E, CO.EG);
end SET_EXECUTION_GUARD;

procedure SET_OUTPUT_GUARD(OP_ID : in PSDL_ID;
STREAM : in PSDL_ID;
E : in EXPRESSION;
procedure SET_OUTPUT_GUARD(
    OPID in PSDL_ID;
    E in EXPRESSION;
    CO in out COMPOSITE_OPERATOR) is

    TEMP_ID : OUTPUT_ID;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        TEMP_ID.OP := OPID;
        TEMP_ID.STREAM := STREAM;
        OUT_GUARD_MAP_PKG.BIND(TEMP_ID, E, CO.OG);
    end SET_OUTPUT_GUARD;

procedure SET_EXCEPTION_TRIGGER(
    OP_ID : in PSDL_ID;
    EXCEP : in PSDL_ID;
    E : in EXPRESSION;
    CO in out COMPOSITE_OPERATOR) is

    TEMP_ID : EXCEP_ID;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        TEMP_ID.OP := OPID;
        TEMP_ID.EXCEP := EXCEP;
        EXCEPTRIGGERMAP_PKG.BIND(TEMP_ID, E, CO.ET);
    end SET_EXCEPTION_TRIGGER;

procedure ADD_TIMER_OP(
    OP_ID, 
    TIMERID in PSDL_ID;
    TOP in TIMER_OP_ID;
    E : in EXPRESSION;
    CO : in out COMPOSITE_OPERATOR) is

    TEMP_ID : TIMER_OP;
    TEMP_SET : TIMER_OP_SET;
    begin
        if CO.CATEGORY /= PSDL_OPERATOR then
            raise NOT_AN_OPERATOR;
        end if;
        TEMP_ID.OP_ID := OP_ID;
        TEMP_ID.TIMER_ID := TIMER_ID;
TEMP_ID.GUARD := E;
TIMER_OP_SET_PKG.EMPTY(TEMP_SET);
TIMER_OP_SET_PKG.ADD(TEMP_ID, TEMP_SET);
TIMER_OP_MAP_PKG.BIND(OP_ID, TEMP_SET, CO.TIM_OP);
end ADD_TIMER_OP;

procedure SET_PERIOD(OP_ID : in PSDL_ID;
P : in MILLISEC;
CO : in out COMPOSITE_OPERATOR) is
-- Raises period_redefined if the period is non-zero.
begin
if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
end if;
if (TIMING_MAPPKG.FETCH(CO.PER, OP_ID)) /= 0 then
    raise PERIOD_REDEFINED;
end if;
TIMING_MAPPKG.BIND(OP_ID, P, CO.PER);
end SET_PERIOD;

procedure SET_FINISH_WITHIN(OP_ID: in PSDL_ID;
FW : in MILLISEC;
CO : in out COMPOSITE_OPERATOR) is
-- Raises finish_within_redefined if
-- the finish_within is non-zero.
begin
if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
end if;
if (TIMING_MAPPKG.FETCH(CO.FW, OP_ID)) /= 0 then
    raise FINISH_WITHIN_REDEFINED;
end if;
TIMING_MAPPKG.BIND(OP_ID, FW, CO.FW);
end SET_FINISH_WITHIN;

procedure SET_MINIMUM_CALLING_PERIOD
  (OP_ID : in PSDL_ID;
-- Raises minimum_calling_period redefined if the 
-- minimum_calling_period is non-zero.
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  end if;
  if (TIMING_MAPPKG.FETCH(CO.MCP, OP_ID)) /= 0 then
    raise MINIMUM_CALLING_PERIOD_REDEFINED;
  end if;
  TIMING_MAPPKG.BIND(OPID, MCP, CO.MCP);
end SET_MINIMUM_CALLING_PERIOD;

procedure SET_MAXIMUM_RESPONSE_TIME
  (OP_ID : in PSDL_ID;
   MRT : in MILLISEC;
   CO : in out COMPOSITE_OPERATOR) is
-- Raises maximum_response_time redefine if the 
-- maximum_response_time is non-zero.
begin
  if CO.CATEGORY /= PSDL_OPERATOR then
    raise NOT_AN_OPERATOR;
  end if;
  if (TIMING_MAPPKG.FETCH(CO.MRT, OP_ID)) /= 0 then
    raise MAXIMUM_RESPONSE_TIME_REDEFINED;
  end if;
  TIMING_MAPPKG.BIND(OP_ID, MRT, CO.MRT);
end SET_MAXIMUM_RESPONSE_TIME;

-- Operations on all psdl types.

function MODEL(T : DATA_TYPE) return TYPE_DECLARATION is
-- Returns the conceptual representation declared 
-- in the specification part,
-- empty if not given.
begin
case T.CATEGORY is
  when PSDL_OPERATOR =>
    raise NOT_A_TYPE;
  when PSDL_TYPE =>
    return T.MDL;
end case;
end MODEL;

function OPERATIONS(T : DATA_TYPE)
  return OPERATION_MAP is
-- Returns an environment binding operation
-- names to operation definitions,
-- an empty map if the type does not define any operations.
begin
  case T.CATEGORY is
    when PSDL_OPERATOR =>
      raise NOT_A_TYPE;
    when PSDL_TYPE =>
      return T.OPS;
  end case;
end OPERATIONS;

-- Operations on composite psdl data types.

function DATA_STRUCTURE(T : COMPOSITE_TYPE) return TYPE_NAME is
-- Returns the data structure declared in the
-- psdl implementation part,
-- raises no_data_structure if the type is implemented in Ada.
begin
  case T.CATEGORY is
    when PSDL_OPERATOR =>
      raise NOT_A_TYPE;
    when PSDL_TYPE =>
      case T.GRANULARITY is
        when ATOMIC =>
          raise NO_DATA_STRUCTURE;
        when COMPOSITE =>
          return T.DATA_STR;
      end case;
  end case;
end case;
end DATA_STRUCTURE;

function MAKE_COMPOSITE_TYPE
 (NAME : PSDL_ID;
  MODEL : TYPE_DECLARATION;
  DATA_STRUCTURE : TYPE_NAME;
  OPERATIONS : OPERATION_MAP;
  GEN_PAR : TYPE_DECLARATION := EMPTY_TYPE_DECLARATION;
  KEYWORDS : ID_SET := EMPTY_ID_SET;
  INFORMAL_DESCRIPTION,
  AXIOMS : TEXT := EMPTY_TEXT)

return COMPOSITE_TYPE is
  -- Create a new composite type.

  X : COMPOSITE_TYPE;

begin
  X.NAME := NAME;
  X.GEN_PAR := GEN_PAR;
  X.KEYW := KEYWORDS;
  X.INF_DESC := INFORMAL_DESCRIPTION;
  X.AX := AXIOMS;
  X.OPS := OPERATIONS;
  X.MDL := MODEL;
  X.DATA_STR := DATA_STRUCTURE;

  return
  X;}
end MAKE_COMPOSITE_TYPE;

-- outputs the psdl program
procedure PUT_PSDL (P: IN PSDL_PROGRAM) is separate;

--**************** FORREFERENCE_ONLY ****************
--**************** FORREFERENCE ONLY
private
-- type psdl_component(category: component_type := psdl_operator;
--                 granularity: implementation_type := composite) is
-- record
--   name: psdl_id;
--   gen_par: type_declaration;
--   keyw: id_set;
--   inf_desc, ax: text;
--   case category is
--     when psdl_operator =>
--       input, output, state: type_declaration;
--       init: init_map;
--       excep: id_set;
--       smet: millisec;
--     case granularity is
--       when atomic => o_ada_name: psdl_id;
--       when composite =>
--         g: psdl_graph;
--         str: type_declaration;
--         tim: id_set;
--         trig: trigger_map;
--         eg: exec_guard_map;
--         og: out_guard_map;
--         et: excep_trigger_map;
--         tim_op: timer_op_map;
--         per, fw, mcp, mrt, rmet: timing_map;
--       end case;
--     end case;
--     when psdl_type =>
--         mdl: type_declaration;
--         ops: operation_map;
--     case granularity is
--       when atomic => t_ada_name: psdl_id;
--       when composite => data_str: type_name;
--       end case;
end private;
end case;
end record;
end PSDL_COMPONENT_PKG;

APPENDIX H. IMPLEMENTATION OF PUT OPERATION

--:-----------:
-- psdl_put.a
--:-----------:

------------------------------------------------------------------------
--
-- Unit name : Output operation for PSDL ADT
-- File name : psdl_put.a
-- Author : Suleyman Bayramoglu
-- Date Created : December 1990
-- Address : bayram@taurus.cs.nps.navy.mii
-- Last Update : [Tue Sep 24 01:14:17 1991 - bayram]
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1.1,
--                                      Verdix Ada version 6.0 (c)
--
------------------------------------------------------------------------

-- Keywords : abstract data type, PSDL program
--
-- Abstract :
--    This package is the implementation for the PSDL ADT

------------------------------------------------------------------------

separate(Psd1_Component_Pkg)
------------------------------------------------------------------------

-- procedure PUT_PSDL
--
-- Extract the text representation of PSDL program from
-- the PSDL ADT and outputs as a legal PSDL source file
-- The output is always to standard output, but command line
-- switch when invoking the expander, directs renames the
-- renames the standard output to as the given UNIX file
-- A modification can be done to this procedure in package

176
procedure Put_Psdl (P: in Psdl_Program) is

Cp : Component_Ptr;
C : Psdl_Component;
O : Operator;
T : Data_Type;
A : Atomic_Component;
Ao : Atomic_Operator;
Co : Composite_Operator;
Ct : Composite_Type;

function Size_Of(S: Psdl_Program_Pkg.Res_Set) return NATURAL
renames Psdl_Program_Pkg.Res_Set_Pkg.Size;

function Size_Of(S: Id_Set) return NATURAL
renames Id_Set_Pkg.Size;

-- function fetch_id(s: id_set; n: natural) return psdl_id
-- renames id_set_pkg.fetch;

Pp_Domain_Set: Psdl_Program_Pkg.Key_Set;
Pp_Range_Set : Psdl_Program_Pkg.Res_Set;

Htab : constant STRING := "     "; -- horizontal tabulation

-- print component category and name of the component
procedure Put_Component_Name(C : in Psdl_Component) is
begin
if Component_Category(C) = Psdl_Operator then
  Put("OPERATOR ");
else

procedure Put_Id List (Id List : in Id Set;
                      Message : in String) is
    I : NATURAL := 1;
    begin
        if not Id_Set_Pkg.Equal(Id List, Empty Id Set) then
            Put_Line(Htab & Htab & Message);
            Put(Htab & Htab & Htab);
            -- Begin expansion of FOREACH loop macro.
            declare
                procedure Loop_Body(Id : Psl_Id) is
                    begin
                    if I > 1 then
                        Put(",");
                    end if;
                    Put(Id.S);
                    I := I + 1;
                end Loop_Body;
            procedure Execute_Loop is new Id_Set_Pkg.Generic_Scan(Loop_Body);
            begin
                Execute_Loop(Id List);
            end;
            -- LIMITATIONS: Square brackets are used as macro
            -- quoting characters,
            -- so you must write [[x]] in the m4 source file
            -- to get [x] in the generated Ada code.
            -- Ada programs using FOREACH loops must avoid the
            -- lower case spellings of
            -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
            -- or must quote them like this: [define].
            -- The implementation requires each package to be generated by
            -- a separate call to m4: put each package in a separate file.
            -- Exit and return statements inside the body of a FOREACH loop
            -- may not work correctly if FOREACH loops are nested.
            -- An expression returned from within a loop body must not
            -- mention any index variables of the loop.
            -- End expansion of FOREACH loop macro.
            New_Line(2);
end if;
end Put_Id_List;

procedure Put_Id_List (Id_List : in Id_Set) is

  I : NATURAL := 1;
begin
  if not Id_Set_Pkg.Equal(Id_List, Empty_Id_Set) then
    -- Begin expansion of FOREACH loop macro.
    declare
      procedure Loop_Body(Id : PSDL_Id) is
      begin
        if I > 1 then
          Put (", ");
        end if;
        Put (Id.S);
        I := I + 1;
      end Loop_Body;
    procedure Execute_Loop is
      new Id_Set_Pkg.Generic_Scan (Loop_Body);
    begin
      Execute_Loop (Id_List);
    end;
  end if;
end Put_Id_List;

procedure Put_Smet (O : in Operator) is
begin
  if O.Smet > 0 then
    Put (Htab & Htab & "MAXIMUM EXECUTION TIME ");
    Put_Line (INTEGER'Image (O.Smet) & " ms");
    New_Line;
  end if;
end Put_Smet;

-- output Informal_Description, Formal_Description
procedure Put_Text (T : in Text; Message : in String) is
begin
  if not A_Strings.IsNull(A_Strings.A_String(T))
    and T /= Empty_Text then
    Put(Htab & Htab & Message & " ");
    Put_Line(T.S);
    New_Line;
  end if;
end Put_Text;

-- Output the Type Name in a recursive manner
procedure Put_Type_Name(Tname: in Type_Name) is
  i : Natural := 1;
begin
  Put(Tname.name.s);
  if not Type_Declaration_Pkg.Equal(Empty_Type_Declaration,
        Tname.Gen_Par) then
    Put("[");
    -- Begin expansion of FOREACH loop macro.
    declare
      procedure loop_body(id: in Psdl_Id; Tn: in Type_Name) is
        begin
          if i > 1 then
            Put(",");
          end if;
          Put(Id.s & ": ");
          Put_Type_Name(Tn);  -- print out the rest
          i := i + 1;
        end loop_body;
    procedure executeloop is
      new Type_Declaration_Pkg.Generic_Scan(loop_body);
    begin
      execute_loop(Tname.Gen_par);
    end;
    -- LIMITATIONS: Square brackets are used as macro
    -- quoting characters,
    -- so you must write "[[x]]" in the m4 source file
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the
    -- lower case spellings of
    -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
    -- or must quote them like this: [define].
    -- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.

    Put("]]");
end if;
end Put_Type_Name;

procedure Put_Type_Decl(Td in TypeDeclaration;
    Message: in String:="") is

    i : natural := 1;
begin
    if not Type_Declaration_Pkg.Equal(Empty_Type_Declaration, Td) then
        if not Type_Declaration_Pkg.Equal(Empty_Type_Declaration, Td) then
            put_line(htab & htab & Message);

            -- Begin expansion of FOREACH loop macro.
            declare
                procedure loopbody(id: in PsdlId; Tn: in Type-Name) is
                    begin
                        if i > 1 then
                            Put(""," & Ascii.lf);
                        end if;
                    Put(Htab & Htab & Htab & Id.S & Ascii.HT &": ");
                    Put_Type_Name(Tn);
                    i := i + 1;

                    end loopbody;
            procedure execute_loop is
                new Type_Declaration_Pkg.Generic_Scan(loop_body);
                begin
                    execute_loop(Td);
                end;
            end if;
        end if;
    end if;
end Put_Type_Decl;
procedure Put_State(State: in Type_Declaration;
    Init : in Init_Map) is
  i, j : Natural := 1;
  Prev_Tn : Type_Name := null;
Begin
  if not Type_Declaration_Pkg.Equal(Empty_Type_Declaration, State) then
    Put_Line(Htab & Htab & "STATES");

    -- Begin expansion of FOREACH loop macro.
    declare
      procedure loop_body(Id: In Psdl_Id; Tn: TypeName) is
        begin
          if i > 1 then
            if Prev_Tn = Tn then
              put(""," & AsciiLf);
            else
              put(" : ");
              Put_Type_Name(Prev_Tn);
              Put_Line("","");
            end if;
          end if;
          put(Htab & Htab & Htab & Id.S);
          Prev_Tn := Tn;
          i := i + 1;
        end loop_body;
      procedure execute_loop is
        new Type_Declaration_Pkg.Generic_Scan(loop_body);
      begin
        execute_loop(State);
      end;
    -- LIMITATIONS: Square brackets are used as macro quoting
    -- characters,
    -- so you must write [[x]] in the m4 source file
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the lower
    -- case spellings of
    -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
    -- or must quote them like this: [define].
    -- The implementation requires each package to be generated by
    -- a separate call to m4: put each package in a separate file.
    -- Exit and return statements inside the body of a FOREACH loop
    -- may not work correctly if FOREACH loops are nested.
    -- An expression returned from within a loop body must not
    -- mention any index variables of the loop.
    -- End expansion of FOREACH loop macro.
    Put(" : ");
  end;
Put_Type_Name(Prev_Tn);
put(" INITIALLY ");

-- Begin expansion of FOREACH loop macro.
declare
    procedure loop_body(Id: In Psdl_Id; E: Expression) is
      begin
        if j > 1 then
          Put(" ");
        end if;
        Put(E.S);
        j := j + 1;
      end loop_body;
      procedure execute_loop is
        new Init_Map_Pkg.Generic_Scan(loop_body);
      begin
        execute_loop(Init);
      end;
      -- LIMITATIONS: Square brackets are used as macro quoting characters,
      -- so you must write [([x]) in the m4 source file
      -- to get [x] in the generated Ada code.
      -- Ada programs using FOREACH loops must avoid the lower case
      -- spellings of
      -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
      -- or must quote them like this: [define].
      -- The implementation requires each package to be generated by
      -- a separate call to m4: put each package in a separate file.
      -- Exit and return statements inside the body of a FOREACH loop
      -- may not work correctly if FOREACH loops are nested.
      -- An expression returned from within a loop body must not
      -- mention any index variables of the loop.
      -- End expansion of FOREACH loop macro.
        new_line(2);
      end if;
    end;
end Put_State;

-- Output operator spec
----------------------------------------
procedure Put_Operator_Spec(O: in Operator) is
begin
  Put_Line(Htab & "SPECIFICATION");
  Put_Type_Decl(O.Gen_Par, "GENERIC");  -- put generic parameters
  Put_Type_Decl(O.Input, "INPUT");  -- put inputs
  Put_Type_Decl(O.Output, "OUTPUT");  -- put outputs

----------------------------------------
Put_State(O.State, O.Init);  -- put states
Put_Id_List(O.Excep, "EXCEPTIONS");  -- put exceptions
Put_Smet(O);  -- put specified MET
-- put reqmts_trace  --not implemented in this version of ADT
Put_Id_List(O.Keyw, "KEYWORDS");  -- put keywords
Put_Text(O.Inf_Desc, "DESCRIPTION");  -- put inf. description
Put_Text(O.Ax, "AXIOMS");  -- put formal description
Put_Line(Htab & "END");
end Put_Operator_Spec;

-- Output psdl type spec

procedure Put_Type_Spec(T: in Data_Type) is
-- Output operator spec for a psdl type
-- the only difference is the format, an elegant
-- way can be easily
-- found to use the procedure Put_Operator_Spec by
-- setting a flag, but this is a quick and dirty fix.

procedure Put_Op_Spec(O: in Operator) is

begin
    Put_Line(Htab & "SPECIFICATION");
    Put_TypeDecl(O.Gen_Par, "GENERIC");  -- put generic parameters
    Put_TypeDecl(O.Input, "INPUT");       -- put inputs
    Put_TypeDecl(O.Output, "OUTPUT");    -- put outputs
    Put_State(O.State, O.Init);          -- put states
    Put_Id_List(O.Excep, "EXCEPTIONS");  -- put exceptions
    Put_Smet(O);                         -- put specified MET
    -- put reqmts_trace -- not implemented in this version of ADT
    Put_Id_List(O.Keyw, "KEYWORDS");    -- put keywords
    Put_Text(O.Inf.Desc, "DESCRIPTION"); -- put inf. description
    -- put formal description
    Put_Line(Htab & "END");
end Put_Op_Spec;

procedure Put_Op_Spec_List(Op_Map: in Operation_Map) is

begin

    declare
        procedure Loop_Body(Id: in Psdl_d; Op in Op_Ptr) is
            begin
                O := Op.all;
                Put(Htab);  -- indent a little bit
                Put_Component_Name(O);
                Put_Op_Spec(O);
                New_Line;
            end Loop_Body;

        procedure Execute_Loop is
            new Operation_Map_Pkg.Generic_Scan(Loop_Body);

            begin
                Execute_Loop(Operation_Map_Pkg.Map(Op_MAP));
            end;

    end Loop_Body;

    procedure Execute_Loop is
        new Operation_Map_Pkg.Generic_Scan(Loop_Body);

        begin
            Execute_Loop(Operation_Map_Pkg.Map(Op_MAP));
        end;

    end Put_Op_Spec_List;
begin  -- Put_Type_Spec
  Put_Line("SPECIFICATION");  -- put generic parameters
  Put_Type_Decl(T.Gen_Par, "GENERIC");  -- Put Model
  Put_Type_Decl(T.Mdl);
  Put_Op_Spec_List(T.Ops);
  Put_Id_List(O.Keyw, "KEYWORDS");  -- put keywords
  Put_Text(O.Inf_Desc, "DESCRIPTION");  -- put inf. description
  Put_Text(O.Ax, "AXIOMS");  -- put formal description
  Put_Line("END");
  New_Line;
end Put_Type_Spec;

-----------------------------------------------
--Output operator implementation

186
procedure Put_Operator_Implmentation(O: in Operator) is
    Co: Composite_Operator;

-- output the graph
-----------------------------------------------
procedure Put_Graph(G: in Psdl_Graph) is

-- output the vertices
-----------------------------------------------
procedure Put_Vertices(G: in Psdl_Graph) is
    Vertex_List: Id_Set;
    Met: MilliSec;
begin
    Id_Set_PKG.Assign(Vertex_List, Psdl_Graph_PKG.Vertices(G));

    --/*foreach([Id: Psdl_Id],[Id_set_PKG.Generic_Scan],[Vertex_List],
    --/* [Value],
    --/*)
    --/* Put(Htab & Htab & Htab & "VERTEX ' & Id.s);
    --/* Met := Psdl_Graph_PKG.Maximum_Execution_Time(Id,G);
    --/* if Met /= 0 then
    --/* Put.Line(" : " & Integer'Image(Met) & " ms");
    --/* else
    --/* New_Line;
    --/* end if;
    --/*

-- Begin expansion of FORFACH loop macro.
declare:
procedure loop_body(Id: Psdl_Id) is
begin
    Put("Htab & Htab & Htab ("VERTEX " & Id.s);
    Met := Psdl_Graph_PKG.Maximum_Execution_Time(Id, G);
    if Met /= 0 then
        Put.Line(" : " & Integer'Image(Met) & " ms");
    else
        New_Line;
    end if;
end loop_body;

procedure execute_loop is
new Id_Set_Pkg.Generic_Scan(loop_body);

begin
  execute_loop(Vertex_List);
end;
New_Line;
end PutVertices;

-----------------------------------------------

-- output the edges
-----------------------------------------------

procedure Put_Edges (G: in Psdl_Graph) is
  Edge_List : Edge_Set;
  Latency_time: Millisec;

begin
  Edge_Set_Pkg.Assign(Edge_List, Psdl_Graph_Pkg.Edges(G));

  --/*foreach([E : EDGE],
  --/*  [Edge_Set_Pkg.Generic_Scan],
  --/*  [Edge_List],
  --/*  [
  --/*  Put(Htab & Htab & Htab & "EDGE  " &
  --/*    E.Stream_Name.s & " ");
  --/*  Latency_Time :=
  --/*  Psdl_Graph_pkg.Latency(E.X, E.Y, E.Stream_Name,G);
  --/*  if Latency_Time /= 0 then
  --/*    Put(": " & Integer'Image(Latency_Time) &" ms ");
  --/*  end if;
  --/*  PutLine (E.X.s " -> " & E.Y.s);
  --/* ])
  -- Begin expansion of FOREACH loop macro.
decclare
  procedure loop_body(E : EDGE) is
  begin
    Put(Htab & Htab & Htab & "EDGE  " & E.Stream_Name.s &" ");
    Latency_Time :=
      Psdl_Graph_pkg.Latency(E.X, E.Y, E.Stream_Name,G);
    if Latency_Time /= 0 then
      Put(": " & Integer'Image(Latency_Time) &" ms ");
    end if;
    PutLine (E.X.s " -> " & E.Y.s);
  end loop_body;

  procedure execute_loop is
    new Edge_Set_Pkg.Generic_Scan(loop_body);
  begin

  end execute_loop;
execute_loop(Edge_List);
end;
New_Line;
end Put_Edges;

begin  -- Put_Graph
  New_Line;
  Put_Line(Htab & Htab & "GRAPH");
  Put_Vertices(G);
  Put_Edges(G);
end Put_Graph;

-- output the control constraints

procedure Put_Control_Constraint(Co :in Composite_OPERATOR) is

  The_Op_Id_Set : Id_Set := Empty_Id_Set;
  Local_Id : Psl_Id; -- to get around Verdict bug

function Vertices (G: Psl_Graph) return Id_Set
renames Psl_Graph_Pkg.Vertices;

--package Tim_IO is new Enumeration_IO(TRIGGER_TYPE);
package Tim_Op_IO is new Enumeration_IO(TIMING_OP_ID);

-- output trigger map

procedure Put_Triggers(O_Name : Psl_Id;
                        T_Map : Trigger_Map) is

  The_Trigger_Record : Trigger_Record;

begin
  -- /* Put the trigger for each operator if they exist */
  if Trigger_Map_Pkg.Member(O_Name, T_Map) then
    Put(Htab & Htab & "TRIGGERED");
    The_Trigger_Record := Trigger_Map_Pkg.Fetch(T_Map, O_Name);
    if The_Trigger_Record.TT = BY_ALL then
      Put("BY ALL ");
Put_Id_List(The_Trigger_Rec.Streams);
elsif The_Trigger_Rec.TT = BY_SOME then
  Put(" BY SOME ");
  Put_Id_List(The_Trigger_Rec.Streams); -- if none
  -- then do nothing
end if;
if not Exec_Guard_Map_Pkg.Member(O_name, O.Eg) then
  Put(Ascii.Lf);
end if;
end if;
end Put_Triggers;

-- output execution guard for each trigger if exists

procedure Put_Execute_Guard(OName Psdl_Id;
    Egjap Exec_GuardMap) is

  The_Execute_Guard_Expr : Expression;

begin
  if Exec_Guard_Pkg.Member(O_name, Eg_Map) then
    The_Execute_Guard_Expr :=
      Exec_Guard_Map_Pkg.Fetch(Eg_Map, O_name);
    Put_Line(" IF ", The_Execute_Guard_Expr.s);
  end if;
end Put_Execute_Guard;

-- output timings for each operator if exists

procedure Put_Timing(Key : in Psdl_Id;
                      Tin_Map : in Timing_Map;
                      Timing_Message: in String) is

  Time_Val: Milliseconds:=0;

begin
  -- Check if timing exists for each operator
  -- if exists print them out.
  if Timing_Map_Pkg.Member(Key, Tim_Map) then
    Time_Val := Timing_Map_Pkg.Fetch(Tim_Map, Key);
    Put(Htab & Htab & Htab & " " & Timing_Message);
  end if;
end Put_Timing;
Put_Line(integer'image(Time_Val) & " ms");
end if;
end Put_Timing;

-- output out guard for each trigger if exists

procedure Put_Output_Guard(O_Name : PSDL_Id;
                          Og_Map : Out_Guard_Map) is
begin
  -- m4 macro code
  -- foreach[Id: Output_Id; E: Expression],
  -- [Out_Guard_Map_Pkg.Generic_Scan],
  -- [Og_Map],
  -- []
  -- if Eq(O_Name, O_Id.Op) then
  --     Put(Htab & Htab & Htab);
  --     Put(" OUTPUT ");
  --     Puc(O_Id.Stream.s);
  --     Put_Line(" IF " & E.s);
  -- end if;
end)

-- Begin expansion of FOREACH loop macro.
declare
  procedure loop_body(O_Id: Output_Id; E: Expression) is
begin
  if Eq(O_Name, O_Id.Op) then
    Put(Htab & Htab & Htab);
    Put(" OUTPUT ");
    Puc(O_Id.Stream.s);
    Put_Line(" IF " & E.s);
  end if;
end loop_body;

procedure execute_loop is
  new Out_Guard_Map_Pkg.Generic_Scan(loop_body);
begin
  execute_loop(Og_Map);
end;
end Put_Output_Guard;

-- output timer op for each operator if exists


procedure Put_Timer_Op(O_Name : Pidl_Id; 
            T_Op_Map : Time_Op_Map) is

   -- The_Timer_Op_Rec : Timer_Op;
   The_Timer_Op_List : Timer_Op_Set;

begin
   -- /* Check if timer op exists for each operator */
   if Timer_Op_Map_Pkg.Member(O_name, T_Op_Map) then
      The_Timer_Op_List :=
         Timer_Op_Map_Pkg.Fetch(T_Op_Map, O_name);

      -- foreach([The_Timer_Op_Rec : Timer_Op],
      --          [Timer_Op_Set_Pkg.Generic_Scan],
      --          [The_Timer_Op_List],
      --          [
      --             Put(Htab & Htab & Htab & " ");
      --             Tim_Op_io.Put(The_Timer_Op_Rec.Op_Id);
      --             Put(" TIMER ");
      --             Put(The_Timer_Op_Rec.Timer_Id.s);
      --             Put_Line(" IF " & The_Timer_Op_Rec.Guard.s);
      --          ])

   -- Begin expansion of FOREACH loop macro.
   declare
      procedure loop_body(The_Timer_Op_Rec : Timer_Op) is
      begin
         Put(Htab & Htab & Htab & " ");
         Tim_Op_io.Put(The_Timer_Op_Rec.Op_Id);
         Put(" TIMER ");
         Put(The_Timer_Op_Rec.Timer_Id.s);
         Put_Line(" IF " & The_Timer_Op_Rec.Guard.s);
      end loop_body;

   procedure execute_loop is
      new Timer_Op_Set_Pkg.Generic_Scan(loop_body);
   begin
      execute_loop(The_Timer_Op_List);
   end;
   end if;
end Put_Timer_Op;
-- output exception triggers for each operator if exists
------------------------------------------------------------------------
procedure Put_Excep_Trigger(O_Name : PSDL_Id;
            Et_Map = Excep_Trigger_Map) is
begin
    -- foreach([E_Id: Excep_Id; E: Expression],
    --       [Excep_Trigger_Map_Pkg.Generic_Scan],
    --       [Et_Map],
    --       [ ]
    -- if Eq(O_name, E_Id.Op) then
    --     Put(Htab & Htab & Htab);
    --     Put(" EXCEPTION ");
    --     Put(E_Id.Excep.s );
    --     Put_Line(" IF " & E.s);
    --     end if;
    -- )
    
    -- Begin expansion of FOREACH loop macro.
    declare
        procedure loop_body(E_Id: Excep_Id; E: Expression) is
            begin
                if Eq(O_name, E_Id.Op) then
                    Put(Htab & Htab & Htab);
                    Put(" EXCEPTION ");
                    Put(E_Id.Excep.s );
                    Put_Line(" IF " & E.s);
                end if;
            end loop_body;

        procedure execute_loop is
            new Excep_Trigger_Map_Pkg.Generic_Scan(loop_body);
        begin
            execute_loop(Et_Map);
        end;
    end Put_Excep_Trigger;
begin
    -- Put_Control_Constraints
    Id_Set_Pkg.Assign(The_Op_Id_Set, Vertices(Co.G));
PutLine(Htab & Htab & "CONTROL CONSTRAINTS");

-- foreach([Id : Psdl_Id], [Id_Set_Pkg.Generic_Scan],
-- [The_Op_Id_Set],
-- {
-- Local_Id := Id;
-- PutLine(Htab & Htab & Htab & "OPERATOR " &Local_Id.s);
-- Put_Triggers (Local_Id, Co.Trig);
-- PutExec_Guard(Local_Id, Co.Eg,);
-- /* Put the timings if exist */
-- Put_Timing(Local_Id, Co.Per, "PERIOD " );
-- Put_Timing(Local_Id, Co.Fw, "FINISH WITHIN ");
-- Put_Timing(Local_Id, Co.Mcp,"MINIMUM CALLING PERIOD ");
-- Put_Timing(Local_Id, Co.Mrt,"MAXIMUM RESPONSE TIME ");
-- Put_ Output_Guard (Local_Id, Co.Og);
-- Put_TimerOp (Local_Id, Co.Tim_Op);
-- Put_Excep_Trigger(Local_Id, Co.Et);
-- NewLine;
-- }
--
-- Begin expansion of FOREACH loop macro.
declare
procedure loop_body(Id : Psdl_Id) is
begin
 Local_Id := Id;
 PutLine(Htab & Htab & Htab & "OPERATOR " &Local_Id.s);
 Put_Triggers (Local_Id, Co.Trig);
 PutExec_Guard(Local_Id, Co.Eg);
/* Put the timings if exist */
 Put_Timing(Local_Id, Co.Per, "PERIOD ");
 Put_Timing(Local_Id, Co.Fw, "FINISH WITHIN ");
 Put_Timing(Local_Id, Co.Mcp, "MINIMUM CALLING PERIOD ");
 Put_Timing(Local_Id, Co.Mrt, "MAXIMUM RESPONSE TIME ");
 Put_Output_Guard (Local_Id, Co.Og);
 Put_TimerOp (Local_Id, Co.Tim_Op);
 Put_Excep_Trigger(Local_Id, Co.Et);
 NewLine;
end loop_body;
procedure execute_loop is
 new Id_Set_Pkg.Generic_Scan(loop_body);
begin
 execute_loop(The_Op_Id_Set);
end;
-- LIMITATIONS: Square brackets are used as
-- macro quoting characters,
-- so you must write [[x]] in the m4 source file
to get \[ x \] in the generated Ada code.

Ada programs using FOREACH loops must avoid the lower case
spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
end Put_Control_Constraints;

begin
Put(Htab & "IMPLEMENTATION ");
if Component_Granularity(O) = Composite then
  Co := O;
  Put_Graph(CO.G); -- put graph
  Put_Type_Decl(Co.Str, "DATA STREAM"); -- put data streams
  Put_Id_List (Co.Tim, "TIMERS"); -- put timers
  Put_Control_Constraints(Co); -- put control constraints
  Put_Text(Co.Impl_Desc, "DESCRIPTION"); -- put inf. description
else
  Put_Line("ADA " & O.0_Ada_Name.S); -- put ada_name
end if;
Put_Line(Htab & "END");
New_Line;
end Put_Operator_Implementation;

-----------------------------------------------
--Output psdl_type implementation
-----------------------------------------------

procedure Put_Type_Implementation(T: in Data_Type) is
  O: Operator;
begin
  Put("IMPLEMENTATION ");
  if Component_Granularity(T) = Composite then
    Put_Type_Name(T.Data_Str);
    New_Line;
    declare
      procedure Loop_Body(Id : in PSDL_Id; Op : in Op_Ptr) is
        begin
          O := Op.all;
          Put_Line(Htab & "OPERATOR " & Id.s);
          Put_Operator_Implementation(O);
        end Loop_Body;
New_Line;

end Loop_Body;
procedure Execute_Loop is
  new Operation_Map_Pkg.Generic_Scan(Loop_Body);
begin
  Execute_Loop(Operation_Map_Pkg.Map(T.Ops));
end;
else
  Put_Line(" ADA " & T.T_Ada_Name.S);
end if;
Put_Line("END");
end Put_Type_Implementation;

begin

  declare
    procedure Loop_Body(Id : in PSDL_Id; Cp : in Component_Ptr) is
    begin
      C := Cp.all;
      Put_Component_Name(C);
      if Component_Category(C) = PSDL_Operator then
        O := C;
        Put_Operator_Spec(O);
        Put_Operator_Implementation(O);
      else
        T := C;
        Put_Type_Spec(T);
        Put_Type_Implementation(T);
      end if;
    end Loop_Body;
  procedure Execute_Loop is
    new PSDL_Program_Pkg.Generic_Scan(Loop_Body);
  begin
    Execute_Loop(PSDL_Program_Pkg.Map(P));
  end; end Put_PSDL;
APPENDIX I. PACKAGE PSDL CONCRETE TYPES

---:-------------:
-- psdl_cts.a
--:-------------:

-----------------------------------------------
--
-- Unit name : Specification of package psdl concrete types
-- File name : psdl_cts.a
-- Author : Valdis Berzins (berzins@taurus.cs.nps.navy.mil)
-- Date Created : December 1990
-- Modified by : Suleyman BAyramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Last Update : {Tue Sep 24 02:00:10 1991 - bayram}
-- Machine/System Compiled/Run on : Sun4, SunOs 4.1.1,
-- VerdiX Ada version 6.0 (c)
--
-----------------------------------------------
-- Keywords : abstract data types
--
-- Abstract :
-- Provides the supporting types to PSDL ADT

----------------------------------------------- Revision history -----------------------------------------------
--
--$Source: 
--/n/gemini/work/bayram/AYACC/parser/psdlAda.lib/RCS/psdlcts.a,v $
--$Revision: 1.7 $
--$Date: 1991/09/24 09:08:47 $
--$Author: bayram $
--
-----------------------------------------------

-- Modified (Fri Aug 30 17:27:59 1991 - bayram)
-- Provided function Eq for generic map and set instatations.

with A_Strings;  -- See verdiX library "standard".
with Generic_Set_Pkg;
    -- Defines a generic set data type.
with Generic_Map_Pkg;
    -- Defines a generic map data type.
package PSDL_CONCRETE_TYPE_PKG is
  subtype MILLISEC is NATURAL;

  type TEXT is new A_Strings.A_string;
  type ADA_ID is new A_Strings.A_string;
  type PSDL_ID is new A_Strings.A_string;
  subtype VARIABLE is PSDL_ID;
  type EXPRESSION is new A_Strings.A_string;

  Empty_Text : constant TEXT := TEXT(A_Strings.Empty);

  function Eq(x, y: PSDL_ID) return BOOLEAN;
  function Eq(x, y: EXPRESSION) return BOOLEAN;

package Id_Set_Pkg is
  new GenericSetPkg(T => PSDL_ID,
                     Block_Size => 48,
                     Eq => Eq);

  subtype IDSET is IdSetPkg.Set;
  function EmptyIdSet return IDSET;
-- Returns an empty set.

package Init_Map_Pkg is
  new GenericMapPkg(Key => PSDL_ID,
                     Result => EXPRESSION,
                     Eq Key => Eq,
                     Eq Res => Eq);

  subtype INIT_MAP is Init_Map_Pkg.Map;
  function Empty_Init_Map return INIT_MAP;
-- Returns an empty init_map;

package Exec_Guard_Map_Pkg is
  new GenericMapPkg(Key => PSDL_ID,
                     Result => EXPRESSION,
                     Eq Key => Eq,
                     Eq_Res => Eq);

  subtype EXEC_GUARD_MAP is Exec_Guard_Map_Pkg.Map;
  function Empty_Exec_Guard_Map return EXEC_GUARD_MAP;
-- Returns an empty exec_guard_map;

  type OUTPUT_ID is
record
    Op, Stream : PSDL_ID;
end record;

function Eq(X, Y: Output_Id) return Boolean;

package Out_Guard_Map_Pkg is
    new Generic_Map_Pkg(Key => OUTPUT_ID,
    Result => EXPRESSION,
    Eq_Key => Eq,
    Eq_Res => Eq);

    subtype OUT_GUARD_MAP is Out_Guard_Map_Pkg.Map;

    function Empty_Out_Guard_Map return OUT_GUARD_MAP;
    -- Returns an empty out_guard_map;

type EXCEP_ID is
    record
        Op, Excep : PSDL_ID;
    end record;

    function Eq(X, Y: Excep_Id) return Boolean;

package Excep_Trigger_Map_Pkg is
    new Generic_Map_Pkg(Key => EXCEP_ID,
    Result => EXPRESSION,
    Eq_Key => Eq,
    Eq_Res => Eq);

    subtype Excep_Trigger_Map is Excep_Trigger_Map_Pkg.Map;
    function Empty_Excep_Trigger_Map return Excep_Trigger_Map;
    -- Returns an empty exceptrigger_map;

type TRIGGER_TYPE is (BY_ALL, BY_SOME, NONE);

type TRIGGER_RECORD is
    record
        Tt : TRIGGER_TYPE;
        Streams : ID_SET;
    end record;

package Trigger_Map_Pkg is new
    Generic_Map_Pkg(Key => PSDL_ID,
    Result => TRIGGER_RECORD,
    Eq_Key => Eq);
subtype TRIGGER_MAP is Trigger_Map_Pkg.Map;

function Empty_Trigger_Map return TRIGGER_MAP;
-- Returns an empty trigger_map;

type TIMER_OP_ID is (START, STOP, RESET, NONE);
type TIMER_OP is
record
  Op_Id : TIMER_OP_ID;
  Timer_Id : PSDL_ID;
  Guard : EXPRESSION;
end record;
package Timer_OP_Set_Pkg is
  new Generic_Set_Pkg(T => TIMER_OP, Block_Size => 1);
subtype Timer_OP_Set is Timer_OP_Set_Pkg.Set;
package Timer_OP_Map_Pkg is
  new Generic_Map_Pkg(Key => PSDL_ID,
                      Result => TIMER_OP_SET,
                      Eq_Key => Eq);

subtype Timer_OP_Map is Timer_OP_Map_Pkg.Map;

function Empty_Timer_OP_Map return Timer_OP_Map;
-- Returns an empty timer_op_map;
package Timing_Map_Pkg is
  new Generic_Map_Pkg(Key => PSDL_ID,
                      Result => MILLISEC,
                      Eq_Key => Eq);

subtype TIMING_MAP is Timing_Map_Pkg.Map;

function Empty_Timing_Map return TIMING_MAP;
-- Returns an empty timing_map;

type TYPE_NAME_RECORD;
   -- Forward declaration.

type TYPE_NAME is access TYPE_NAME_RECORD;
   -- The name of a psdl type, with optional generic parameters.
package Type_Declaration_Pkg is
  new Generic_Map_Pkg(Key => PSDL_ID,
                      Result => TYPE_NAME,
                      Eq_Key => Eq);
subtype Type_Declaration is Type_Declaration_Pkg.Map;
-- A psdl type declaration is a map from psdl identifiers
-- to psdl type names.
-- The default value of a type declaration map is
-- the null pointer.

type TYPE_NAME_RECORD is
  record
    Name : PSDL_ID;
    Gen Par : Type_Declaration;
  end record;
-- The generic parameter map is empty if
-- the named type is not generic.

function Empty_Type_Declaration return Type_Declaration;
-- Returns an empty type declaration map.

end PSDL_CONCRETE_TYPE_PKG;

-- ------------------------
-- Keywords : abstract data types
-- Abstract
-- Provides the supporting types to PSDL ADT

------------------------ Revision history ------------------------
package body Psdl_Concrete_Type 2kg is

use !d_SetPkg, TimerOp_Set 2kg;
use InitMapPkg, Exec_Guard_Map_Pkg,
use OutGuardMapPkg;
use Excep_Trigger_Map_Pkg, Trigger_Map_Pkg;
use TimerOp_Map_Pkg, Timing_Map_Pkg,
use TypeDeclaration_2kg;

Empty_Expression: constant Expression := A_Strings.Empty;

function Empty_Id_Set return Id_Set is
  S: Id_Set;
begin
  Empty(S);
  return S;
end Empty_Id_Set;
-- Returns an empty set.

-- Overloaded functions for generic instantiations

function Eq(x, y: Psdl_Id)
return BOOLEAN is
begin
  return (X.S = Y.S);
end Eq;

function Eq(x, y: Expression)
return BOOLEAN is
begin
  return (X.S = Y.S);
end Eq;
function Eq(X, Y: Output_Id)
    return Boolean is
begin
    return (Eq(X.Op, Y.Op) and Eq(X.Stream, Y.Stream));
end Eq;

function Eq(X, Y: Excep_Id) return Boolean is
begin
    return (Eq(X.Op, Y.Op) and Eq(X.Excep, Y.Excep));
end Eq;

function Empty_Init_Map return Init_Map is
    M : Init_Map;
begin
    Create(Empty_Expression, M);
    return M;
end Empty_Init_Map;
-- Returns an empty init_map;

function Empty_Exec_Guard_Map
    return Exec_Guard_Map is
    M : Exec_Guard_Map;
begin
    Create(Empty_Expression, M);
    return M;
end Empty_Exec_Guard_Map;
-- Returns an empty exec_guard_map;

function Empty_Out_Guard_Map
    return Out_Guard_Map is
    M : Out_Guard_Map;
begin
    Create(Empty_Expression, M);
    return M;
end Empty_Out_Guard_Map;
-- Returns an empty out_guard_map;

function Empty_Excep_Trigger_Map
    return Excep_Trigger_Map is
    M : Excep_Trigger_Map;
begin
    
204
Create(EmptyExpression, M);
return M;
end Empty_Exception_Map;
-- Returns an empty exceptrigger_map;

function Empty_Tigger_Map
return Trigger_Map is
  X : Trigger_Record;
  M : Trigger_Map;
begin
  X.Tt := None;
  X.Streams := Empty_Id_Set;
  Create(X, M);
  return M;
end Empty_Tigger_Map;
-- Returns an empty trigger_map;

function Empty_Timer_Op_Map return Timer_Op_Map is
  X : Timer_Op_Set;
  M : Timer_Op_Map;
begin
  Empty(X);
  Create(X, M);
  return M;
end Empty_Timer_Op_Map;
-- Returns an empty timer_op_map;

function Empty_Timing_Map
return Timing_Map is
  M : Timing_Map;
begin
  Create(C, M);
  return M;
end Empty_Timing_Map;
-- Returns an empty timing_map;

function Empty_Type_Declaration
return Type_Declaration is
  X : Type_Name := null;
  M : Type_Declaration;
begin
  Create(X, M);
  return X;
end Empty_Type_Declaration;
end Empty_Type_Declaration;
-- Returns an empty type declaration map.
end Psdl_Concrete_Type_Pkg;
APPENDIX J. SPECIFICATION OF PSDL GRAPH ADT

--:----------:
-- psddl_graph_s.a
--:----------:

---------------------------------------------------------------------
--
-- Unit name : Specification of PSDL Graph ADT
-- File name : psddl_graph_s.a
-- Author : Valdis Berzins (berzins@taurus.cs.nps.navy.mil)
-- Date Created : December 1990
-- Modified by : Suleyman Bayramoglu
-- Address : bayram@taurus.cs.nps.navy.mil
-- Last Update : [Tue Sep 24 02:00:10 1991 - bayram]
-- Machine/System Compiled/Run on : Sun4, SunOS 4.1.1,
--                               Verdix Ada version 6.0 (c)
--
---------------------------------------------------------------------
-- Keywords : abstract data types, graphs, PSDL
--
-- Abstract :
--      Provides the supporting types to PSDL ADT

---------------------------------------------------------------------
-- Revision history --
--
-- $Source: /n/gemini/work/bayram/AYACC/parser//RCS/psddl_graph_s.a,v $
-- $Revision: 1.5 $
-- $Date: 1991/09/24 09:33:35 $
-- $Author: bayram $
--
---------------------------------------------------------------------
--
---------------------------------------------------------------------
--- REFERENCES ---
--

---
---

207
package PSDL_GRAPH_PKG is

  type PSDL_GRAPH is private;

  -- An EDGE represents a data stream from operator X to operator Y.
  -- Since there can exist more than one data stream between X and Y,
  -- the name STREAM_NAME identifies a unique data stream.
  -- In this way, the use of STREAM_NAME allows several streams
  -- with different names to connect the
  -- same pair of operators, X and Y.

  type EDGE is record
    X, Y,
    STREAM_NAME : PSDL_ID;
  end record;

package EDGE_SET_PKG is

  new GENERIC_SET_PKG(t => EDGE, block_size => 12);

  subtype EDGE_SET is EDGE_SET_PKG.SET;
iconstructor OPERATIONS

-- Returns the graph with no vertices and no edges.
function EMPTY_PSDL_GRAPH return PSDL_GRAPH;

function ADD_VERTEX(
   OP_ID : PSDL_ID;
   G    : PSDL_GRAPH;
   MAXIMUM_EXECUTION_TIME : MILLISEC := 0)
return PSDL_GRAPH;

function ADD_EDGE(X, Y, STREAM_NAME : PSDL_ID;
   G    : PSDL_GRAPH;
   LATENCY : MILLISEC := 0)
return PSDL_GRAPH;

ATTRIBUTE OPERATIONS

-- HAS_VERTEX() returns TRUE if
-- and only if OP_ID is a vertex in G.
function HAS_VERTEX(OP_ID : PSDL_ID;
   G    : PSDL_GRAPH)
return BOOLEAN;

-- HAS_EDGE() returns TRUE if and only if
-- there exists an edge from vertex
-- X to vertex Y in G.
function HAS_EDGE(X, Y : PSDL_ID;
   G    : PSDL_GRAPH)
return boolean;

-- STREAM_NAMES() accepts arguments for vertices and the graph.
-- The function returns the names of the data streams
-- connecting operator X and operator Y.

209
-- The result can be empty if there are no streams
-- between X and Y, and it can have more than one element
-- if several streams connect X and Y.
function STREAM_NAMES(X,
    Y : PSDL_ID;
    G : PSDL_GRAPH)
return id_set;

-- The maximum execution time allowed for the operator V.
function MAXIMUM_EXECUTION_TIME(V : PSDL_ID;
    G : PSDL_GRAPH)
return MILLISEC;

-- The maximum data transmission delay between
-- a write operation by
-- operator X on the given stream and the
-- corresponding read operation by
-- operator Y.
function LATENCY(X,
    Y,
    STREAM_NAME : PSDL_ID;
    G : PSDL_GRAPH)
return MILLISEC;

-- The maximum data transmission delay between
-- the last write operation
-- by operator X and the first read operation
-- by operator Y. Zero if
-- there are no edges between X and Y,
-- the largest latency of the edges if
-- several edges connect X and Y.
function LATENCY(X,
    Y : PSDL_ID;
    G : PSDL_GRAPH)
return MILLISEC;

-- The set of all vertices in G.
function VERTICES(G : PSDL_GRAPH)
return ID_SET;

-- The set of all edges in G.
function EDGES(G : PSDL_GRAPH)
return EDGE_SET;
The set of all vertices U with an EDGE from V to U in G.

function SUCCESSORS (V: PSDL_ID; G: PSDL_GRAPH) return ID_SET;

The set of all vertices U with an EDGE from U to V in G.

function PREDECESSORS (V: PSDL_ID; G: PSDL_GRAPH) return ID_SET;

private

package MAXIMUM_EXECUTION_TIME_MAP_PKG is
   new GENERIC_MAP (KEY => PSDL_ID,
                   RESULT => MILLISEC);

   type MAXIMUM_EXECUTION_TIME_MAP is
      new MAXIMUM_EXECUTION_TIME_MAP_PKG.MAP;

package LATENCY_PKG is
   new GENERIC_MAP_PKG (KEY => EDGE,
                         RESULT => MILLISEC);

   type LATENCY_MAP is new LATENCY_PKG.MAP;

   type PSDL_GRAPH is record
      VERTICES : ID_SET;
      EDGES : EDGE_SET;
      MAXIMUM_EXECUTION_TIME : MAXIMUM_EXECUTION_TIME_MAP;
      LATENCY : LATENCY_MAP;
   end record;

end PSDL_GRAPH_PKG;
APPENDIX K. IMPLEMENTATION OF PSDL GRAPH ADT

--:-----------:
-- psdl_graph_b.a
--:-----------:

-----------------------------------------------
--
-- Unit name : Implementation of Psdl Graph ADT
-- File name : psdl_graph_b.a
-- Modified by : Suleyman BAYRAMOGLU
-- Address : bayram@taurus.cs.nps.navy.mil
-- Last Update : (Tue Sep 24 02:00:10 1991 - bayram)
-- Machine/System Compiled/Run on : Sun4, SunOS 4.1.1,
-- SMALL Ada version 6.0 (c)
--
-----------------------------------------------
-- Keywords : abstract data types, graphs, PSDL
--
-- Abstract :
-- Provides the supporting types to PSDL ADT

---------------------------------------------- Revision history ----------------------------------------------
--
--$Source: /n/gemini/work/bayram/AYACC/parser//RCS/psdl_graph_b.a,v $
--$Revision: 1.3 $
--$Date: 1991/09/24 09:52:09 $
--$Author: bayram $
--
----------------------------------------------

package body PSDL_GRAPH_PKG is

-- +-------------------------------------------------------
-- |
-- |
-- CONSTRUCTOR OPERATIONS
-- |
-- +-------------------------------------------------------

212
function EMPTY_PSDL_GRAPH return PSDL_GRAPH is

begin

  G.VERTICES := PSDL_CONCRETE_TYPE_PKG.EMPTY_ID_SET;
  EDGE_SET_PKG.EMPTY(G.EDGES);
  CREATE(0, G.MAXIMUM_EXECUTION_TIME);
  CREATE(0, G.LATENCY);

  return G;

end EMPTY_PSDL_GRAPH;

function ADD_VERTEX(OPID : PSDL_ID;
                      G : PSDL_GRAPH;
                      MAXIMUM_EXECUTION_TIME MILLISEC := 0)

return PSDL_GRAPH is

begin

  -- Add OP_ID to the vertex set and then use
  -- the GENERIC_MAP_PKG procedure
  -- BIND() to bind the OP_ID to its MAXIMUM_EXECUTION_TIME and
  -- updates the new graph's map accordingly.

  PSDL_CONCRETE_TYPE_PKG.IDSETPKG.ADD(OP_ID, H.VERTICES);
  BIND(OP_ID, MAXIMUM_EXECUTION_TIME, H.MAXIMUM_EXECUTION_TIME);

return H;

end ADD_VERTEX;
return H;
end ADD_VERTEX;

-- ADD_EDGE: Adds a directed edge from X to Y in G.
--
-- The edge takes on the name STREAM_NAME, supplied by the caller.
-- The caller may also specify a LATENCY for the edge (or accept the
-- default of zero. H is the new PSDL_GRAPH. That is,
--
-- H = G + (the new edge).
--
function ADD_EDGE(X, Y, STREAM_NAME : PSDL_ID;
    G : PSDL_GRAPH; LATENCY : MILLISEC := 0)
return PSDL_GRAPH is

    E : EDGE;
    H : PSDL_GRAPH := G;

begin

    -- Assign to components of the edge E...ADD() the edge E to H...and
    -- finally, update the LATENCY map of H with the (argument) LATENCY
    -- for the edge E.
    E.X := X;
    E.Y := Y;
    E.STREAM_NAME := STREAM_NAME;
    EDGE_SET_PKG.ADD(E, H.EDGES);
    BIND(E, LATENCY, H.LATENCY);

    return H;

end ADD_EDGE;

--
-- ATTRIBUTE OPERATIONS
--
--
-- HAS_VERTEX() returns TRUE if and only if OP_ID is a vertex in G.
--
function HAS_VERTEX(OP_ID : PSDL_ID;
                      G : PSDL_GRAPH)
    return BOOLEAN is
begin
    return
        PSDL_CONCRETE_TYPE_PKG.ID_SET_PKG.MEMBER(OP_ID, G.VERTICES);
end HAS_VERTEX;

-- HAS_EDGE() returns TRUE if and only if there exists
-- an edge from vertex
-- X to vertex Y in G.
-- First we find the LAST_INDEX for the EDGES of G,
-- the first to the last ELEMENT and compare X and Y.
-- If we obtain a
-- match at any time, we return TRUE.
-- If we search the entire list with
-- no success, FALSE is returned.
function HAS_EDGE(X, Y : PSDL_ID;
                  G : PSDL_GRAPH)
    return BOOLEAN is
    e : EDGE;
    local_x : psdl_id:=x;
    local_y : psdl_id:=y;
begin
    -- Begin expansion of FOREACH loop macro.
    declare
        procedure loop_body(e : edge) is
            begin
                if (e.X = local_X) and then (e.y = local_y) then
                    raise edge_set_pkg.return_from_foreach;
                end if;
            end loop_body;
    procedure execute_loop is
new edge_set_pkg.generic_scan(loop_body);
begin
execute_loop(g.edges);
except
when edge_set_pkg.return_from_for => return true;
end;

-- LIMITATIONS: Square brackets are used as macro
-- quoting characters,
-- so you must write [[x]] in the m4 source file
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the
-- lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
end HAS_EDGE;

-- STREAM_NAMES() accepts arguments for vertices and the graph. The
-- function returns the name(s) of the data stream(s)
-- connecting operator
-- X and operator Y. The result can be empty if there is no stream
-- between X and Y, and it can have more than one element if several
-- streams connect X and Y.

-- The function starts by assigning the size of the edge set of G to
-- LAST_INDEX and making S an empty ID_SET.
-- Next, we loop from 1 until
-- the LAST_INDEX, looking at the EDGES in G. When we find an EDGE
-- from X to Y, the corresponding STREAM_NAME is added to S.

function STREAM_NAME(X, Y : PSDL_ID;
                     G : PSDL_GRAPH)
  return ID.Set is

  e : EDGE;
  s : ID_SET := PSDL_CONCRETE_TYPE_PKG.EMPTY_ID_SET;
  local_x : psdl_id := x;
  local_y : psdl_id := y;

  begin
-- Begin expansion of FOREACH loop macro.

declare
 procedure loop_body( e : edge) is
 begin
   if (e.x = local_x) and then (e.y = local_y) then
     id_set_pkg.add(e.stream_name, s);
   end if;
 end loop_body;
 procedure execute_loop is new edge_set_pkg.generic_scan(loop_body);
 begin
   execute_loop(g.edges);
 end;

 return S;
 end STREAM_NAMES;

-- The maximum execution time allowed for the operator V.

function MAXIMUM_EXECUTION_TIME(V : PSDL_ID;
                                  G : PSDL_GRAPH)
 return MILLISEC is

-- Value to flag no such vertex in G?
 MET : MILLISEC := 0;

begin

-- Search the MAXIMUM_EXECUTION_TIME mapping of G
-- for the (key) vertex
-- V. If the vertex is found, the corresponding
-- time is returned;
-- else, zero is returned.

if HAS_VERTEX(V, G) then
  return FETCH(G.MAXIMUM_EXECUTION_TIME, V);
else
  return MET;
end if;

end MAXIMUM_EXECUTION_TIME;

-- The maximum data transmission delay between a write operation by
-- operator X on the given stream and the corresponding
-- read operation by
operator Y.

function LATENCY(X, Y, STREAM_NAME : PSDL_ID;
G : PSDL_GRAPH)
    return MILLISEC is

    E : EDGE;
    T : MILLISEC := 0;

    begin

        E.X := X;
        E.Y := Y;
        E.STREAM_NAME := STREAM_NAME;

        if HAS_EDGE(X, Y, G) then
            return FETCH(G.LATENCY, E);
        else
            return T;
        end if;

    end LATENCY;

-- The maximum data transmission delay between the last write operation
-- by operator X and the first read operation by operator Y.
-- Zero if
-- there are no edges between X and Y,
-- the largest latency of the edges if
-- several edges connect X and Y.
--
function LATENCY(X, Y : PSDL_ID;
G : PSDL_GRAPH)
    return MILLISEC is

    E     : EDGE;
    L     : MILLISEC;
    T     : MILLISEC := 0;
    local_x : psdl_id := x;
    local_y : psdl_id := y;

    begin

        if HAS_EDGE(X, Y, G) then
            -- Begin expansion of FOREACH loop macro.
            declare
                procedure loop_body(e : edge) is

                begin

                    declare
                        procedure loop_body(e : edge) is

                    end loop_body;

                    begin

                        -- End expansion of FOREACH loop macro.

                end if;

        end LATENCY;
begin
  if (E.X = local_X and E.Y = local_Y) then
    L := FETCH(G.LATENCY, E);
    if (L > T) then
      T := L;
      end if;
    end if;
  end loop body;
procedure execute_loop is
  new edge_set_pkg.generic_scan(loop_body);
begin
  execute_loop(g.edges);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters, -- so you must write [[x]] in the m4 source file -- to get [x] in the generated Ada code. -- Ada programs using FOREACH loops must avoid the lower -- case spellings of -- the identifier names "DEFINE", "UNDEFINE", and "DNL", -- or must quote them like this: [define]. -- The implementation requires each package to be generated by -- a separate call to m4: put each package in a separate file. -- Exit and return statements inside the body of a FOREACH loop -- may not work correctly if FOREACH loops are nested. -- An expression returned from within a loop body must not -- mention any index variables of the loop. -- End expansion of FOREACH loop macro.
end if;
return T;
end LATENCY;
-- The set of all vertices in G.
-- function VERTICES(G : PSDL_GRAPH) return ID_SET is
begin
  return G.VERTICES;
end VERTICES;

-- The set of all edges in G.
-- function EDGES(G : PSDL_GRAPH) return EDGE_SET is
begin
   return G.EDGES;
end EDGES;

-- The set of all vertices U with an EDGE from V to U in G.
--
function SUCCESSORS(V: PSDL_ID; G: PSDL_GRAPH) return ID_SET is

   E   : EDGE;
   S   : ID_SET := PSDL_CONCRETE_TYPE_PKG.EMPTY_ID_SET;

begin
   -- Begin expansion of FOREACH loop macro.
   declare
      procedure loop_body(e: edge) is
         begin
            if (E.X = V) then
               ID_SET_PKG.ADD(E.Y, S);
            end if;
         end loop_body;
   procedure execute_loop is
      new edge_set_pkg.generic_scan(loop_body);
   begin
      execute_loop(g.edges);
   end;
   -- LIMITATIONS: Square brackets are used as macro quoting characters,
   -- so you must write [[x]] in the m4 source file
   -- to get [x] in the generated Ada code.
   -- Ada programs using FOREACH loops must avoid the
   -- lower case spellings of
   -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
   -- or must quote them like this: [define].
   -- The implementation requires each package to be generated by
   -- a separate call to m4: put each package in a separate file.
   -- Exit and return statements inside the body of a FOREACH loop
   -- may not work correctly if FOREACH loops are nested.
   -- An expression returned from within a loop body must not
   -- mention any index variables of the loop.
   -- End expansion of FOREACH loop macro.

   return S;
function PREDECESSORS(V: PSDL_ID;
   G: PSDL_GRAPH)
   return ID_SET is
begin
   -- Begin expansion of FOREACH loop macro.
   declare
      procedure loopbody(e: edge) is
      begin
         if (E.y = V) then
            ID_SET_PKG.ADD(E.x, S);
         end if;
      end loopbody;
   end;

   begin
      -- LIMITATIONS: Square brackets are used as macro quoting characters,
      -- so you must write "[[x]]" in the m4 source file
      -- to get [x] in the generated Ada code.
      -- Ada programs using FOREACH loops must avoid
      -- the lower case spellings of
      -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
      -- or must quote them like this: [define].
      -- The implementation requires each package to be generated by
      -- a separate call to m4: put each package in a separate file.
      -- Exit and return statements inside the body of a FOREACH loop
      -- may not work correctly if FOREACH loops are nested.
      -- An expression returned from within a loop body must not
      -- mention any index variables of the loop.
      -- End expansion of FOREACH loop macro.

      return S;
   end PREDECESSORS;
end PSDL_GRAPH_PKG;
APPENDIX L. GENERIC SET PACKAGE

-- This implementation is limited: the Ada ":=" and ":=" operations
-- are not safe or correct for sets.
-- Use the "assign" and "generic_equal" procedures instead.
-- An Ada limited private type could not used because of restrictions
-- on generic in parameters of limited private types
-- (see generic_reduce).

-- You should use the "recycle" procedure on block exit
-- or subprogram return
-- to reclaim storage for any local variables of
-- type set declared in the block

-- Sets are unbounded, but do not require heap storage unless
-- the size of the set exceeds the block_size.

with Text_IO; use Text_IO;
generic
  type T is private;
  Block_Size: in NATURAL := 128;
  with function Eq(X, Y: T) return BOOLEAN is ":=";
package Generic_Set_Pkg is
  type SET is private;

  procedure Empty(S: out SET);
  procedure Add(X: in T; S: in out SET);
  procedure Remove(X: in T; S: in out SET);
  function Member(X: T; S: SET) return BOOLEAN; -- x IN s.
  procedure Union(S1, S2: in SET; S3: out SET); -- s3 = s1 U s2.
  procedure Difference(S1, S2: in SET; S3: out SET); -- s3 = s1 - s2.
  procedure Intersection(S1, S2: in SET; S3: out SET);

  -- generic
type other_set_type is private; -- set(t1).
-- package generic_cross_product_pkg;

function Size(S: SET) return NATURAL;
function Equal(S1, S2: SET) return BOOLEAN;
function Subset(S1, S2: SET) return BOOLEAN;
-- function proper_subset(s1, s2: set) return boolean;

generic
    with function "<"(X, Y: T) return BOOLEAN is <>
    with function Successor(X: T) return T;
procedure Generic_Interval(X1, X2: in T; S: out SET); -- {x1 .. x2}.

generic
    type ET is private; -- Element type for result.
    type ST is private; -- Element set type for result.
    with function F(X: T) return ET is <>
    with procedure Empty(S: out ST) is <>
    with procedure Add(X: in ET; S: in out ST) is <>
    procedure Generic_Apply(S1: in SET; S2: out ST);

generic
    with function F(X, Y: T) return T;
    Identity: T;
function Generic_Reduce(S: SET) return T;

generic
    with function F(X, Y: T) return T;
    function Generic_Reduce1(S: SET) return T;

generic
    with procedure Generate(X: in T);
procedure Generic_Scan(S: SET);

Exit_From_Foreach, Return_From_Foreach: exception;

Empty_Reduction_Undefined : exception; -- Raised by reducel.

-- System functions.
procedure Assign(X: out SET; Y: in SET); -- x := y
procedure Recycle(S: in SET);
    -- Recycles any heap storage used by s.
    -- Call recycle(s) just before leaving any block where
    -- a variable s: set is declared.

-- Text I/O procedures
-- Package lookahead_stream_pkg and procedure input are
used instead of get
because text_io does not support examining a lookahead character
from an input file without moving past it.
One character lookahead is needed to parse Spec set syntax.
Format is \{ element, element, .., element \}

generic
  with procedure Input(Item: out T) is <>;
  procedure Generic_Input(Item: out SET);

procedure Generic_File_Input(File: in File_Type; Item: out SET);
  -- Read a set element from the lookahead stream, stream_machine_pkg.

-- The generic put procedures are designed to work with the standard
-- put procedures provided by the predefined Ada data types.

generic
  with procedure Put(Item: in T) is <>;
  procedure Generic_Put(Item: in SET);

generic
  with procedure Put(File: in File_Type; Item: in T) is <>;
  procedure Generic_File_Put(File: in File_Type; Item: in SET);

private
  type LINK is access SET;
  type ELEMENTS_TYPE is array(1 .. Block_Size) of T;

  type SET is
    record
      Size: NATURAL := 0;  -- The size of the set.
      Elements: ELEMENTS_TYPE;  -- The actual elements of the set.
      Next: LINK := null;  -- The next node in the list.
    end record;
    -- Elements[1 .. min(size, block_size)] contains data.
end Generic_Set_Pkg;
-- Warning: due to a bug in vedix Ada version 6.0,
-- it has been necessary to patch the definitions of
-- remove, member, difference, intersection, subset.
-- The compiler bug causes incorrect references to the
-- formal parameters of a
-- subprogram from within a locally declared subprogram (e.g. loop_body)
-- that is passed as a generic subprogram parameter in
-- a generic instantiation.
-- Patches introduce local copies of procedure parameters
-- (such as local_x)
-- to work around a case where variable references get confused.
-- If the compiler bug is fixed someday, these local copies can be
-- removed.

with unchecked_deallocation;
with lookahead_pkg; use lookahead_pkg;
with delimiter_pkg; use delimiter_pkg;

-- generic
-- type t is private;
-- block_size: in natural := 128;
-- with function eq(x, y: t) return boolean is "=";
package body generic_set_pkg is

recycle_list: link := null; -- The recycle list for recycling storage.

nodes_in_recycle_list: natural := 0; -- The length of the recycle list.

nodes_in_use: natural := 0; -- The number of set heap nodes in use;
-- Invariant: nodes_in_recycle_list
-- = length(recycle_list) <= nodes_in_use.

-- Local subprogram declarations.

function copy_list(l: link)
return link;

function create(sz: natural;
  e: elements_type;
  next: link)
return link;
function token return character;

-- End local subprogram declarations.

-- Constant declarations.
blank: constant delimiter_array := initialize_delimiter_array;
-- End constant declarations.

SET PACKAGE PROCEDURES & FUNCTIONS

-- note: called by details internal usage of functions and procedures.
-- by default all instantiating programs are potential users as well.

-- Procedure name: empty
-- Description: return an empty set
-- Called by: apply

procedure empty (s: out set) is
  sl: set;
begin
  s := sl;
  end empty;

-- Procedure name: add
-- Description: add an element to a set

procedure add (x: in t; s: in out set) is
begin
  if not(member(x, s)) then
    s.size := s.size + 1;
    if s.size <= block_size then
      s.elements(s.size) := x;
    elsif s.next = null then
      s.next := create(1, (others => x), null);
    else
      add(x, s.next.all);
    end if;
  end if;
end add;
procedure remove (x: in t; s: in out set) is
  ss: set;
local_x: t := x; -- patch to work around compiler bug, verdix 6.0.
begin
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
      begin if not(eq(local_x, y)) then add(y, ss); end if;
    end loop_body;
    procedure execute_loop is new generic_scan(loop_body);
  begin
    execute_loop(s);
  end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write \[x\] in the body of a FOREACH
  -- to get \(x\) in the generated Ada code.
  -- Ada programs using FOREACH loops must avoid the lower case spellings
of

-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
recycle(s);
s:= ss;
end remove;

---------------------------------MEMBER---------------------------------

-- Function name: member
-- Description: test if an element is a member in a set
-- Called by: subset, add, union, difference, intersection

---------------------------------MEMBER---------------------------------

function member (x: t; s: set) return boolean is
  local_x: t := x; -- patch to work around compiler bug, verdix 6.0.
begin
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
      begin if eq(local_x, y) then raise return_from_foreach ; end if;
      end loop_body;
    procedure execute_loop is new generic-scan(loop_body);
    begin
      execute_loop(s);
    exception
      when return_from_foreach => return true;
    end;
    -- LIMITATIONS: Square brackets are used as macro quoting characters,
    -- so you must write [[x]] in the body of a FOREACH
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the lower case
    -- spellings of
    -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
    -- or must quote them like this: [define].
    -- The implementation requires each package to be generated by
    -- a separate call to m4: put each package in a separate file.
    -- Exit and return statements inside the body of a FOREACH loop
    -- work correctly only if the FOREACH loops are not nested.
    -- End expansion of FOREACH loop macro.
  return(false);
end member;
procedure union (s1, s2: in set; s3: out set) is
  ss : set; -- Initialized to empty.
begin
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
      begin add(y, ss);
    end loop_body;
    procedure execute_loop is new generic_scan(loop_body);
  begin
    execute_loop(s1);
  end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write \[[x]\] in the body of a FOREACH
  -- to get [x] in the generated Ada code.
  -- Ada programs using FOREACH loops must avoid the lower case
  -- spellings of
  -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
  -- or must quote them like this: [define].
  -- The implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- Exit and return statements inside the body of a FOREACH loop
  -- work correctly only if the FOREACH loops are not nested.
  -- End expansion of FOREACH loop macro.
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
      begin add(y, ss);
    end loop_body;
    procedure execute_loop is new generic_scan(loop_body);
  begin
    execute_loop(s2);
  end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write \[[x]\] in the body of a FOREACH
  -- to get [x] in the generated Ada code.
  -- Ada programs using FOREACH loops must avoid the lower case
  -- spellings of
  -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
  -- or must quote them like this: [define].
The implementation requires each package to be generated by
a separate call to m4: put each package in a separate file.
Exit and return statements inside the body of a FOREACH loop
work correctly only if the FOREACH loops are not nested.
End expansion of FOREACH loop macro.

--- DIFFERENCE-----------------------------

-- Procedure name: difference
-- Description: return a set difference of two input sets
-- Called by:

procedure difference (s1, s2: in set; s3: out set) is
  ss : set;
  local_s2 : set := s2; -- patch to work around compiler bug, verdix 6.0.
begin
  -- Begin expansion cf FOR:.CH loop macro.
  declare
    procedure loop_body(y: t) is
      begin if not member(y, local_s2) then add(y, ss); end if;
    end loop_body;
    procedure execute_loop is new generic_scan(loop_body);
  begin
    execute_loop(s1);
  end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write [[x]] in the body of a FOREACH
  -- to get [x] in the generated Ada code.
  -- Ada programs using FOREACH loops must avoid the lower case
  -- spellings of
  -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
  -- or must quote them like this: [define].
  -- The implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- Exit and return statements inside the body of a FOREACH loop
  -- work correctly only if the FOREACH loops are not nested.
  -- End expansion of FOREACH loop macro.
  s3 := ss;
end difference;

--------------------------INTERSECTION--------------------------

-- Function name: intersection
-- Description: return a set intersection of two input sets
-- Called by:
procedure intersection (s1, s2: in set; s3: out set) is
ss : set;
local_s2: set := s2; -- patch to work around compiler bug, verdix 6.0.
begin
  -- Begin expansion of FOREACH loop macro.
declare
  procedure loop_body(y: t) is
  begin
    if member(y, local_s2) then add(y, ss); end if;
  end loop_body;
  procedure execute_loop is new generic_scat(loop_body);
begin
  execute_loop(s1);
end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write [[x]] in the body of a FOREACH
  -- to get [x] in the generated Ada code.
  -- Ada programs using FOREACH loops must avoid the lower
  -- case spellings of
  -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
  -- or must quote them like this: [define].
  -- The implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- Exit and return statements inside the body of a FOREACH loop
  -- work correctly only if the FOREACH loops are not nested.
  -- End expansion of FOREACH loop macro.
s3 := ss;
end intersection;

--------------------------- SIZE ---------------------------------------

-- Function name: size
-- Description: return the number of elements in a set, zero if empty
-- Called by:

function size (s: set) return natural is
begin
  return s.size;
end size;

------------------------- EQUAL ---------------------------------------

-- Function name: equal
-- Description: tests if two sets are equal
-- Called by:

function equal(s1, s2: set) return boolean is
begin
bl := subset(s1, s2);
b2 := subset(s2, s1);
return bl and b2;
end;

----------------------------- SUBSET -----------------------------

-- Function name: subset
-- Description: check if one set is a subset of another set
-- Called by: equal

function subset(s1, s2: set) return boolean is
i1: natural := 1;
result: boolean := true;
local_s2: set := s2; -- patch to work around compiler bug, verdix 6.0.
begi
if s1.size > s2.size then result := false;
else -- Begin expansion of FOREACH loop macro.
declare
    procedure loop_body(y: t) is
    begin if not(member(y, local_s2))
        then result := false; raise exit_from_foreach; end if;
    end loop_body;
    procedure execute_loop is new genericscan(loop_body);
begi
    execute_loop(s1);
exception
    when ex:t from foreach => null;
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the body of a FOREACH
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case
-- spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
end if;
return result;
end subset;
-- Procedure name: interval
-- Description: get the elements of a sel that are within the input
interval

-- generic
-- with function "<(x, y: t) return boolean is <>
-- with function successor(x: t) return t;
-- ALL(x: t) :: (x < y = successor(x) <= y)
procedure generic_interval(x1, x2: in t; s: out set) is
  ss: set; -- Initialized to empty.
y: t := x1;
begin
  while not (x2 < y) loop -- Invariant: x1 <= y.
    add(y, ss);
    y := successor(y);
  end loop;
s := ss;
end generic_interval;

------------------- APPLY ----------------------------------
-- Procedure name: apply
-- Description: apply function "f" on element of a set
-- Called by:

-- generic
-- type et is private; -- Element type for result.
-- type st is private; -- Element set type for result.
-- with function f(x: t) return et is <>
-- with procedure empty(s: out st) is <>
-- with procedure add(x: in et; s: in out st) is <>
procedure generic_apply(sl: in set; s2: out st) is
  ss: st;
begin
  empty(ss);
  -- Begin expansion of FOREACH loop macro.
declare
  procedure loop_body(y: t) is
    begin add(f(y), ss);
  end loop_body;
  procedure execute_loop is new generic_scan(loop_body);
begin
  execute_loop(sl);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the body of a FOREACH
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case
-- spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
s2 := ss;
end generic_apply;

------------------------------REDUCE--------------------------------
-- Function name: reduce
-- Description: reduce set to an element by applying function "f"
-- Called by:

-- generic
-- with function f(x, y: t) return t;
-- identity: t;
function generic_reduce(s: set) return t is
  x: t := identity;
begin
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
    begin x := f(y, x);
    end loop_body;
  procedure execute_loop is new genericscan(loop_body);
begin
  execute_loop(s);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the body of a FOREACH
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the
-- lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
return x;
end generic_reduce;
-- Function name: reducel
-- Description: same as reduce only without the identity element
-- Called by:

-- generic
-- with function f(x, y: t) return t;
function generic_reducel(s: set) return t is
  x: t;
  i: natural := 1;
begin
  if s.size = 0 then raise empty_reduction_undefined; end if;
  begin expansion of FOREACH loop macro.
  declare
    procedure loop_body(y: t) is
      begin if i = 1 then x := y; else x := f(y, x); end if;
        i := i + 1;
    end loop_body;
    procedure execute_loop is new generic_scan(loop_body);
  begin
    execute_loop(s);
  end;
  -- LIMITATIONS: Square brackets are used as macro quoting characters,
  -- so you must write [[x]] in the body of a FOREACH
  -- Ada programs using FOREACH loops must avoid the lower
  -- case spellings of
  -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
  -- or must quote them like this: [define].
  -- The implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- Exit and return statements inside the body of a FOREACH loop
  -- work correctly only if the FOREACH loops are not nested.
  -- End expansion of FOREACH loop macro.
  return x;
end generic_reducel;

-----------------------------------SCAN-----------------------------------
-- Procedure name: scan
-- Description: frame of loop structure
-- Called by:

-- generic
-- with procedure generate(x: in t);
procedure generic_scan(s: set) is
t: set := s;
begin
  while t.next /= null loop
    for i in 1..block_size loop
      generate(t.elements(i));
    end loop;
    t := t.next.all;
  end loop;
  for i in 1..t.size loop
    generate(t.elements(i));
  end loop;
end generic_scan;

-----------------ASSIGN-----------------------------
-- Function name: assign
-- Description: safe version of ":=".
-----------------RECYCLE-----------------------------
-- Procedure name: recycle
-- Description: destroys a set and reuses the associated storage
-- Called by: remove

procedure recycle (s: in set) is
  l: link := s.next;
  head, temp: link;
  procedure free is new unchecked_deallocation (set, link);
begin
  while l /= null loop
    head := l;
    l := l.next;
    nodes_in_use := nodes_in_use - 1;
    if nodes_in_recycle_list < nodes_in_use then
      temp := recycle_list;
      recycle_list := head;
      recycle_list.next := temp;
      nodes_in_recycle_list := nodes_in_recycle_list + 1;
    else
      free(head);
  end while;
end recycle;
end if;
end loop;
end recycle;

-- LOCAL SUBPROGRAMS

------------------------COPY_LIST------------------------

-- Function name: copy_list
-- Description: creates a distinct copy of a list representing a set.
-- Called by: assign

function copy_list(l: link) return link is
begin
if l = null then return l;
else return create(l.size, l.elements, copy_list(l.next));
end if;
end copy_list;

-----------------------CREATE-----------------------

-- Function name: create
-- Description: create a new block of set elements
-- Called by: add

function create (sz: natural; e: elements_type; next: link) return link is
l: link;
begin
nodes_in_use := nodes_in_use + l;
if recycle_list = null then
return new set'(sz, e, next);
else
l := recycle_list;
recycle_list := recycle_list.next;
nodes_in_recycle_list := nodes_in_recycle_list - l;
l.size := sz;
l.elements := e;
l.next := next;
return l;
end if;
end create;

----------------------------------------TOKEN----------------------------------------
function token return character is
  -- Blank is a constant array, see top of package body.
  begin
  -- Advance the lookahead stream to a non-blank character.
    while blank(peek) loop skip_char; end loop;
  -- Return the character without removing it from the stream.
    return peek;
  end token;

procedure generic_input
  -- Procedure name: generic input
  -- Description: input sets. Format is { element , element , .. , element }
  -- Called by: generic_file_input

generic
with procedure input(item: out t) is <>

procedure generic_input(item: out set) is
  x: t;
  s: set; /* Working copy of the result, initialized to empty.*/
begin
  empty(s);
  if token /= '{' then raise data_error; end if;
  skip_char; /* Pass over the opening left bracket.*/
  while token /= '}' loop
    input(x); /* Read and pass over the next element of the set.*/
    add(x, s); /* Add the element to the set.*/
    if token = ',' then
      skip_char;
    elsif token /= ')') then
      raise data_error;
      /* if there is no comma we should be at the end of the set.*/
      end if;
  end loop; /* Now the closing right brace is the lookahead character.*/
  skip_char;
  item := s;
exception
  when others => raise data_error;
end generic_input;

---- GENERIC-FILE-INPUT ----
-- Procedure name: generic file input
-- Description: sets input from files
-- Called by:
---------

-- generic
with procedure input(item: out t) is <>

procedure generic_file_input(file: in file_type; item: out set) is
  procedure get_set is new generic_input;
begin
  set_input(file); /* Connect the lookahead stream to the file.*/
  get_set(item);
  set_input(standard_input); /* Restore the standard input file.*/
end generic_file_input;
--- GENERIC-PUT ---
-- Procedure name: generic put
-- Description: output set. Format is \{ element, \ldots, element \}
-- Called by:

-- generic
-- with procedure put(item: in t) is <>;
procedure generic_put(item: in set) is
i: natural := 1;
begi
put(ascii.l_brace);
-- Begin expansion of FOREACH loop macro.
declare
procedure loopbody(y: t) is
begin if i > 1 then put(","); end if;
put(y); i := i + 1;
end loopbody;
procedure execute_loop is new generic_scan(loopbody);
begi
execute_loop(item);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write \[[x]\] in the body of a FOREACH
-- to get \[x\] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower
-- case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: \[define\].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
put(ascii.r_brace);
end generic_put;

--- GENERIC-FILE-PUT ---
-- Procedure name: Generic file put
-- Description: Output set to file
-- Called by:

-- generic
-- with procedure put(file: in file_type; item: in t) is <>;
procedure generic_file_put(file: in file_type; item: in set) is
i: natural := 1;

241
begin
put(file, ascii.l_brace);
-- Begin expansion of FOREACH loop macro.
declare
  procedure loop_body(y: t) is
  begin if i > 1 then put(file, ", "); end if;
    put(file, y); i := i + 1;
  end loop_body;
  procedure execute_loop is new generic_scan(loop_body);
begin
execute_loop(item);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the body of a FOREACH
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower
-- case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- work correctly only if the FOREACH loops are not nested.
-- End expansion of FOREACH loop macro.
put(file, ascii.r_brace);
end generic_file_put;

--------------------------------------------------------------------------------
end generic_set_pkg;
APPENDIX M. GENERIC MAP PACKAGE

-- this implementation is limited: the ada "=" and "#" operations
-- are not safe or correct for maps.
-- use the "assign" and "generic_equal" procedures instead.

-- you should use the "recycle" procedure on block exit or subprogram return
-- to reclaim storage for any local variables of type map declared in the block

-- maps are unbounded, but do not require heap storage unless
-- the size of the map exceeds the block_size.

with generic_set_pkg;
with text_io; use text_io;
generic
  type key is private;  -- type of the domain element
  type result is private;  -- type of the range element
  block_size: in natural := 12;  -- the memory allocation unit.
with function eq_key(k1, k2: key) return boolean is "=";
with function eq_res(r1, r2: result) return boolean is "=";
package generic_map_pkg is
  type pair is private;
  type map is private;

package key_set_pkg is
  new generic_set_pkg(t => key, eq => eq_key, block_size => block_size);
subtype key_set is key_set_pkg.set;
package res_set_pkg is
  new generic_set_pkg(t => result, eq => eq_res, block_size => block_size);
subtype res_set is res_set_pkg.set;

procedure create(r: in result; m: out map);
procedure bind(x: in key; y: in result; m: in out map);
procedure remove(x: in key; m: in out map);
procedure remove(s: in key_set; m: in out map);
function fetch(m: map; x: key) return result;
function member(x: key; m: map) return boolean;
function equal(m1, m2: map) return boolean;
function submap(m1, m2: map) return boolean;
function map_domain(m: map) return key_set;
function map_range(m: map) return res_set;
function map_default(m: in map) return result;

generic
    with procedure generate(k: in key; r: in result);
procedure generic_scan(m: in map);

exit_from_foreach, return_from_foreach: exception;

-- system functions.
procedure assign(x: out map; y: in map); -- x := y
procedure recycle(m: in map);
    -- recycles any heap storage used by m.
    -- call recycle(m) just before leaving any block where
    -- a variable m: map is declared.

-- text i/o procedures
-- this package supports generic input of map data in the following format:
-- [{key1,result1},{key2,result2}, ..., ; default]
-- the following generic procedures will read and write the map data.
-- package iookahead_stream_pkg and procedure input are used instead of get
-- because text_io does not support examining a lookahead character
-- from an input file without moving past it.
-- one character lookahead is needed to parse spec map syntax.

generic
    with procedure key_input(k: out key) is <>;
    with procedure res_input(r: out result) is <>;
procedure generic_input(m: out map);

generic
    with procedure key_put(k: in key) is <>;
    with procedure res_put(r: in result) is <>;
procedure generic_put(item: in map);

 generic
    with procedure key_put(file: in file_type; k: in key) is <>;
    with procedure resy_put(file: in file_type; r: in result) is <>;
procedure generic_file_put(file: in file_type; item: in map);

private
type pair is
    record
        key_val: key;
        res_val: result;
    end record;

function pair_eq(x, y: pair) return boolean;

package pair_set_pkg is
    new generic_set_pkg(t => pair, eq => pair_eq, block_size => block_size),
subtype pair_set is pair_set_pkg.set;

type map is
record
  def_val: result; -- default value supplied by user
  pairs: pair_set;
end record;
end generic_map_pkg;

-- empty log message
-- map_b.a
-- empty log message

-- warning: due to a bug in vedix ada version 6.0,
-- it has been necessary to patch the definitions of
-- fetch, member.
-- the compiler bug causes incorrect references to the formal parameters of a
-- subprogram from within a locally declared subprogram (e.g. loop_body)
-- that is passed as a generic subprogram parameter in a generic instantiation.
-- patches introduce local copies of procedure parameters (such as local_x)
-- to work around a case where variable references get confused.
-- if the compiler bug is fixed someday, these local copies can be removed.

with lookahead_pkg; use lookahead_pkg;
with delimiter_pkg; use delimiter_pkg;
with text_io; use text_io;
-- generic
-- type key is private; -- type of the domain element
-- type result is private; -- type of the range element
-- with function eq_key(k1, k2: key) return boolean;
-- with function eq_res(r1, r2: result) return boolean;
package body generic_map_pkg is

-- local subprogram declarations
  function token return character;
  -- constant declarations
blank: constant delimiter_array := initialize_delimiter_array;

------------------------ create ------------------------
-- procedure name: create
-- description: creates a map instance and sets the user supplied default
procedure create(r: in result; m: out map) is
  mm: map;
begin
  mm.def_val := r;
  pair_set_pkg.empty(mm.pairs);
  m := mm;
end create;

---------- bind ----------------------------------------
-- procedure name: bind
-- description: adds an element to an existing map
procedure bind(x: in key; y: in result; m: in out map) is
  p: pair;
begin
  remove(x, m);
  if y /= m.def_val then
    p.key_val := x;
    p.res_val := y;
    pair_set_pkg.add(p, m.pairs);
  end if;
end bind;

------------------ remove -----------------------------------
-- procedure name: remove
-- description: removes an element from a map
procedure remove(x: in key; m: in out map) is
  p: pair;
begin
  if member(x, m) then
    p.key_val := x;
    p.res_val := fetch(m, x);
    pair_set_pkg.remove(p, m.pairs);
  end if;
end remove;

------------------ remove -----------------------------------
-- procedure name: remove
-- description: removes a set of elements from a map
procedure remove(s: in key_set; m: in out map) is
  p: pair;
begin
  -- for k: key in generic_scan(s) loop
  --  remove(k, m);
  --  end loop;
  -- begin expansion of foreach loop macro.
  declare
    procedure loop_body(k: key) is
      begin remove(k, m);
    end loop_body;
    procedure execute_loop is new key_set_pkg.generic_scan(loop_body);
  begin
    execute_loop(s);
  end;
  -- limitations: square brackets are used as macro quoting characters,
  -- so you must write [[x]] in the m4 source file
  -- to get [x] in the generated ada code.
  -- ada programs using foreach loops must avoid the lower case spellings of
  -- the identifier names "define", "undefine", and "dnl",
  -- or must quote them like this: [define].
  -- the implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- exit and return statements inside the body of a foreach loop
  -- may not work correctly if foreach loops are nested.
  -- an expression returned from within a loop body must not
  -- mention any index variables of the loop.
  -- end expansion of foreach loop macro.
end remove;

------------------------------- fetch -------------------------------
-- function name: fetch
-- description: returns the range value of a map for a given domain value

function fetch(m: map; x: key) return result is
  y: result := m.def_val;
  local_x: key := x; -- patch to work around compiler bug, verdix 6.0.
begin
  -- begin expansion of foreach loop macro.
  declare
    procedure loop_body(p: pair) is
      begin if eq_key(p.key_val, local_x)
        then y := p.res_val; raise exit_from_foreach ; end if;
    end loop_body;
    procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
  begin
    execute_loop(m.pairs);
  exception
    when exit_from_foreach => null;
  end;
function member(x: key; m: map) return boolean is
   p: pair;
   found: boolean := false;
   local_x: key := x; -- patch to work around compiler bug, verdix 6.0.
begin
   -- begin expansion of foreach loop macro.
   declare
      procedure loop_body(p: pair) is
         begin
            if eqkey(p.key_val, local_x) then raise return_from_foreach; end if;
         end loop_body;
      procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
   begin
      execute_loop(m.pairs);
   exception
      when return_from_foreach => return true;
   end;
   -- limitations: square brackets are used as macro quoting characters,
   -- so you must write [[x]] in the m4 source file
   -- to get [x] in the generated ada code.
   -- ada programs using foreach loops must avoid the lower case spellings of
   -- the identifier names "define", "undefine", and "dnl",
   -- or must quote them like this: [define].
   -- the implementation requires each package to be generated by
   -- a separate call to m4: put each package in a separate file.
   -- exit and return statements inside the body of a foreach loop
   -- may not work correctly if foreach loops are nested.
   -- an expression returned from within a loop body must not
   -- mention any index variables of the loop.
   -- end expansion of foreach loop macro.
   return(false);
end member;

----------------------------- member -------------------------------------
-- function name: member
-- description: indicates whether an element is a member of a map

----------------------------------

-- limitations: square brackets are used as macro quoting characters,
-- so you must write [[x]] in the m4 source file
-- to get [x] in the generated ada code.
-- ada programs using foreach loops must avoid the lower case spellings of
-- the identifier names "define", "undefine", and "dnl",
-- or must quote them like this: [define].
-- the implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- exit and return statements inside the body of a foreach loop
-- may not work correctly if foreach loops are nested.
-- an expression returned from within a loop body must not
-- mention any index variables of the loop.
-- end expansion of foreach loop macro.
return(y);
end fetch;

------------------------------ member --------------------------------------
-- function name: member
-- description: indicates whether an element is a member of a map

function member(x: key; m: map) return boolean is
   p: pair;
   found: boolean := false;
   local_x: key := x; -- patch to work around compiler bug, verdix 6.0.
begin
   -- begin expansion of foreach loop macro.
   declare
      procedure loop_body(p: pair) is
         begin
            if eqkey(p.key_val, local_x) then raise return_from_foreach; end if;
         end loop_body;
      procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
   begin
      execute_loop(m.pairs);
   exception
      when return_from_foreach => return true;
   end;
   -- limitations: square brackets are used as macro quoting characters,
   -- so you must write [[x]] in the m4 source file
   -- to get [x] in the generated ada code.
   -- ada programs using foreach loops must avoid the lower case spellings of
   -- the identifier names "define", "undefine", and "dnl",
   -- or must quote them like this: [define].
   -- the implementation requires each package to be generated by
   -- a separate call to m4: put each package in a separate file.
   -- exit and return statements inside the body of a foreach loop
   -- may not work correctly if foreach loops are nested.
   -- an expression returned from within a loop body must not
   -- mention any index variables of the loop.
   -- end expansion of foreach loop macro.
   return(false);
end member;

------------------------------ member --------------------------------------
-- function name: member
-- description: indicates whether an element is a member of a map

function member(x: key; m: map) return boolean is
   p: pair;
   found: boolean := false;
   local_x: key := x; -- patch to work around compiler bug, verdix 6.0.
begin
   -- begin expansion of foreach loop macro.
   declare
      procedure loop_body(p: pair) is
         begin
            if eqkey(p.key_val, local_x) then raise return_from_foreach; end if;
         end loop_body;
      procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
   begin
      execute_loop(m.pairs);
   exception
      when return_from_foreach => return true;
   end;
   -- limitations: square brackets are used as macro quoting characters,
   -- so you must write [[x]] in the m4 source file
   -- to get [x] in the generated ada code.
   -- ada programs using foreach loops must avoid the lower case spellings of
   -- the identifier names "define", "undefine", and "dnl",
   -- or must quote them like this: [define].
   -- the implementation requires each package to be generated by
   -- a separate call to m4: put each package in a separate file.
   -- exit and return statements inside the body of a foreach loop
   -- may not work correctly if foreach loops are nested.
   -- an expression returned from within a loop body must not
   -- mention any index variables of the loop.
   -- end expansion of foreach loop macro.
   return(false);
end member;

--- function name: equal
--- description: indicates whether or not two maps are equal by determining
--- whether each map is a submap of the other.

function equal(m1, m2: map) return boolean is
  b1, b2: boolean;
begin
  return submap(m1, m2) and then submap(m2, m1);
end equal;

--- function name: submap
--- description: indicates whether one map is a subset of another map by
--- determining whether the set of domain and range values of
--- one map is a subset of the domain and range values of the
--- other.

function submap(ml, m2: map) return boolean is
begin
  return (map_default(ml) = map_default(m2)) and then
  (pair_set_pkg.subset(ml.pairs, m2.pairs));
end submap;

--- function name: map_domain
--- description: returns the set of domain values for a map

function map_domain(m: map) return key_set is
  k_set : key_set;
begin
  key_set_pkg.empty(k_set);
  -- begin expansion of foreach loop macro.
  declare
    procedure loop_body(p: pair) is
    begin
      key_set_pkg.add(p.key_val, k_set);
    end loop_body;
    procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
begin
  execute_loop(m.pairs);
  end;
  -- limitations: square brackets are used as macro quoting characters,
  -- so you must write {{x}} in the m4 source file
  -- to get [x] in the generated ada code.
  -- ada programs using foreach loops must avoid the lower case spellings of
  -- the identifier names "define", "undefined", and "dnl",
function map_range(m: map) return res_set is
  r_set : res_set;
begin
  res_set_pkg.empty(r_set);
  declare
    procedure loop_body(p: pair) is
      begin
        res_set_pkg.add(p.res_val, r_set);
      end loop_body;
    procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
    begin
      execute_loop(m.pairs);
    end;
  end;
end map_range;

-------------------------- map_default --------------------------

function name: map_default
-- description: returns the default value of a map

function map_default(m: in map) return result is
begin
    return m.def_val;
end map_default;

------------------------------- scan -------------------------------
procedure name: scan
--- description: generic procedure which provides the capability to move
--- through a map, one element at a time, performing a generic
--- procedure on each element.

--- generic
--- with procedure generate(k: in key; r: in result);
procedure generic_scan(m: in map) is
begin
    -- begin expansion of foreach loop macro.
    declare
        procedure loop_body(p: pair) is
            begin generate(p.key_val, p.res_val);
            end loop_body;
        procedure execute_loop is new pair_set_pkg.generic_scan(loop_body);
        begin
            execute_loop(m.pairs);
        end;
    -- limitations: square brackets are used as macro quoting characters,
    -- so you must write [[x]] in the m4 source file
    -- to get [x] in the generated ada code.
    -- ada programs using foreach loops must avoid the lower case spellings of
    -- the identifier names "define", "undefine", and "dnl",
    -- or must quote them like this: [define].
    -- the implementation requires each package to be generated by
    -- a separate call to m4: put each package in a separate file.
    -- exit and return statements inside the body of a foreach loop
    -- may not work correctly if foreach loops are nested.
    -- an expression returned from within a loop body must not
    -- mention any index variables of the loop.
    -- end expansion of foreach loop macro.
end generic_scan;

------------------------------- assign -------------------------------
procedure assign(x: out map; y: in map) is
begin
    x.def_val := y.def_val;
    pair_set_pkg.assign(x.pairs, y.pairs);
end assign;
procedure recycle (m: in map) is
begin
    pair_set_pkg.recycle(m.pairs);
end recycle;

procedure generic_input (m: out map) is
x: key;
y: result;
ml: map;
begin
    if token /= '{' then raise data_error; end if;
    skip_char;
    while token /= '}' loop
        if token /= '{' then raise data_error; end if;
        skip_char;
        key_input(x);
        if token /= ',' then raise data_error; end if;
        skip_char;
        res_input(y);
        if token /= '}' then raise data_error; end if;
        skip_char;
        bind(x, y, ml);
        if token = ',' then skip_char;
        elsif token = ';' then
            skip_char;
            res_input(ml.def_val);
            if token = ')' then skip_char; else raise data_error; end if;
        else raise data_error;
        end if;
    end loop;
    m := ml;
exception
    when others => raise data_error;
end generic_input;
--- generic_put -------------------------------------

-- procedure name: generic_put
-- description: outputs map data to the screen

---

-- generic
-- with procedure key_put(k: in key) is <>;
-- with procedure res_put(r: in result) is <>;

procedure generic_put(item: in map) is
  i: natural := 1;
begin
  put("(");
  -- begin expansion of foreach loop macro.
  declare
    procedure loop_body(k: in key; r: in result) is
      begin if i > 1 then put(" "); end if;
      put("["); key_put(k); put(" "); res_put(r); put("]" );
      i := i + 1;
    end loop_body;
  procedure execute_loop is new genericscan(loop_body);
  begin
    execute_loop(item);
  end;
  -- limitations: square brackets are used as macro quoting characters,
  -- so you must write [\{x\}] in the m4 source file
  -- to get [x] in the generated ada code.
  -- ada programs using foreach loops must avoid the lower case spellings of
  -- the identifier names "define", "undefine", and "dnl",
  -- or must quote them like this: [define].
  -- the implementation requires each package to be generated by
  -- a separate call to m4: put each package in a separate file.
  -- exit and return statements inside the body of a foreach loop
  -- may not work correctly if foreach loops are nested.
  -- an expression returned from within a loop body must not
  -- mention any index variables of the loop.
  -- end expansion of foreach loop macro.
  put(")"); res_put(map_default(item));
  put(")" );
end generic_put;

---

--- generic_file_put -------------------------------------

-- procedure name: generic_file_put
-- description: outputs map data to the screen

---

-- generic
-- with procedure key_put(file: in file_type; k: in key) is <>;
-- with procedure res_put(file: in file_type; r: in result) is <>;

procedure generic_file_put(file: in file_type; item: in map) is
  i: natural := 1;
begin
put(file, ");
-- begin expansion of foreach loop macro.
declare
procedure loop_body(k: in key; r: in result) is
begin if i > 1 then put(file, ", "); end if;
   put(file, ");
   key_put(file, k);
   put(file, ", ");
   res_put(file, r);
   put(file, "]");
i := i + 1;
end loop_body;
procedure execute_loop is new generic_scan(loop_body);
begin
   execute_loop(item);
end;
-- limitations: square brackets are used as macro quoting characters,
-- so you must write \[(x\] in the m4 source file
-- to get \(x\] in the generated ada code.
-- ada programs using foreach loops must avoid the lower case spellings of
-- the identifier names "define", "undefine", and "dnl",
-- or must quote them like this: [define].
-- the implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- exit and return statements inside the body of a foreach loop
-- may not work correctly if foreach loops are nested.
-- an expression returned from within a loop body must not
-- mention any index variables of the loop.
-- end expansion of foreach loop macro.
put(file, "; "); res_put(file, map_default(item));
p
put(file, "]");
end generic_fileput;

--------------------------------------------------------------------------------
-- local subprograms
--------------------------------------------------------------------------------

---------------------- pair_eq ----------------------
-- procedure name: pair_eq
-- description: used to check equality of pairs, for supporting pair sets.
--------------------------------------------------------------------------------

function pair_eq(x, y: pair) return boolean is
begin
   return eq_key(x.key_val, y.key_val) and then eq_res(x.res_val, y. res_val);
end pair_eq;

------------------------------ token -----------------------------
-- procedure name: token
-- description: used to parse input characters from input stream
-----------------------------------------------

function token return character is
   -- blank is a constant array, see local constants section of package body
begin
  -- advance the lookahead stream to a non-blank character
  while blank(peek) loop
    skip_char;
    end loop;
  -- return the character without removing it from the stream
  return peek;
end token;
end generic_map_pkg;
APPENDIX N. GENERIC SEQUENCE PACKAGE

-- seq_s.a
-- $Source: /n/gemini/work/bayram/AYACC/parser/psdl_ada.lib/RCS/seq_s.a,v $
-- $Date: 1991/09/24 10:42:27 $
-- $Revision: 1.5 $
-- This implementation is limited: the Ada ":=" and ":=" operations
-- are not safe or correct for sequences.
-- Use the "assign" and "generic_equal" procedures instead.
-- An Ada limited private type could not used because of restrictions
-- on generic in parameters of limited private types (see generic_reduce).
-- You should use the "recycle" procedure on block exit or subprogram return
-- to reclaim storage for any local variables of type sequence declared in
-- the block.
-- Sequences are unbounded, but do not require heap storage unless
-- the length of the sequence exceeds the block_size.

with generic_set_pkg;
-- with max;
-- with square_root_pkg; use square_root_pkg;
-- with text_io; use text_io;
generic
type t is private;
block_size : in natural := 8;
-- average_size: in natural := 8;
-- The average number of elements per sequence, for efficiency.
package generic_sequence_pkg is

  type sequence is private;
type index_array is array(natural range <>) of natural; -- used by fetch $2.
package natural_set_pkg is new generic_set_pkg(natural);
subtype natural_set is natural_set_pkg.set;

  procedure empty(s: out sequence);
  procedure add(x: in t; s: in out sequence);

generic
  with function eq(x, y: t) return boolean is <>;
  procedure generic_remove(x: in t; s: in out sequence);

procedure append(s1, s2: in sequence; s: out sequence); -- s := s1 || s2.

256
function fetch(s: sequence; n: natural) return t;  -- s[n].
procedure fetch(sl: sequence; ia: index_array; s: out sequence);  -- sl[s].
procedure fetch(sl: sequence; low, high: natural; s: out sequence);
  -- sl[low .. high]
function length(s: sequence) return natural;
function domain(s: sequence) return natural_set;

generic
  with function eq(x, y: t) return boolean is <>;
function generic_member(x: t; s: sequence) return boolean;  -- x IN s.

generic
  with function eq(x, y: t) return boolean is <>;
function generic_part_of(s1, s2: sequence) return boolean;  -- s1 IN s2.

generic
  with function eq(x, y: t) return boolean is <>;
function generic_equal(s1, s2: sequence) return boolean;

generic
  with function "<"(x, y: t) return boolean is <>;
function generic_less_than(s1, s2: sequence) return boolean;

generic
  with function "<"(x, y: t) return boolean is <>;
  with function eq(x, y: t) return boolean is <>;
function generic_less_than_or_equal(s1, s2: sequence) return boolean;

generic
  with function "<"(x, y: t) return boolean is <>;
function generic_greater_than(s1, s2: sequence) return boolean;

generic
  with function "<"(x, y: t) return boolean is <>;
  with function eq(x, y: t) return boolean is <>;
function generic_greater_or_equal(s1, s2: sequence) return boolean;

generic
  with function eq(x, y: t) return boolean is <>;
function generic_subsequence(s1, s2: sequence) return boolean;

generic
  with function "<"(x, y: t) return boolean is <>;
  with function successor(x: t) return t;
  -- ALL(x y: t :: x < y => successor(x) <= y)
procedure generic_interval(x1, x2: t; s: out sequence);  -- x1 .. x2.

generic
  type et is private;
  type st is private;  -- st = sequence(et)
  with function f(x: et) return t;

257
with function length(s: st) return natural is <>;
with function fetch(s: st; n: natural) return et is <>;
procedure generic_apply(sl: st; s2: out sequence);

generic
with function f(x, y: t) return t;
identity: t;
function generic_reduce(s: sequence) return t;

generic
with function f(x, y: t) return t;
function generic_reduce1(s: sequence) return t;

generic
with procedure generate(x: in t);
procedure generic_scan(s: sequence);

exit_from_foreach, return_from_foreach: exception;

-- System functions.
procedure assign(x: out sequence; y: in sequence); -- x := y
procedure recycle(s: in sequence);
-- Recycles any heap storage used by s.
-- Call recycle(s) just before leaving any block where
-- a variable s: sequence is declared.

-- Text I/O procedures
-- Package lookahead_pkg and procedure input are used instead of get
-- because text_io does not support examining a lookahead character
-- from an input file without moving past it.
-- One character lookahead is needed to parse Spec sequence syntax.

generic
with procedure input(item: out t) is <>;
-- Read a sequence element from the lookahead stream, stream_machine_pkg.
procedure generic_input(item: out sequence);
-- Read a sequence element from the lookahead stream, stream_machine_pkg.

generic
with procedure input(item: out t) is <>;
-- Read a sequence element from the lookahead stream, stream_machine_pkg.
procedure generic_file_input(item: out sequence; file: in file_type);
-- Read a sequence from the file, using lookahead from stream_machine_pkg.

-- The generic put procedures are designed to work with the standard
-- put procedures provided by the predefined Ada data types.

generic
with procedure put(item: in t) is <>;
procedure generic_put(item: in sequence);
generic
  with procedure put(file: in file_type; item: in t) is <>;
procedure generic_file_put(file: in file_type; item: in sequence);

bounds_error: exception; -- Raised by fetch.
empty_reduction_undefined: exception; -- Raised by reducel.

private
  -- A linked list containing up to block_size elements per node.
  -- The header node is contained directly in the variable.
  -- Distinct sequences are contained in distinct memory locations,
  -- so the representation data structures can be safely modified without
  -- risk of interference.

  type link is access sequence;
  -- Let a = average_size, b = block_size,
  -- p = #bits:pointer, e = #bits:element of type t
  -- Expected space overhead = o = (a/b)*p + (b/2)*e
  -- minimize o: do/db = 0 = -ap/b*b +e/2
  -- optimal b = sqrt(2*a*p/e)
  -- block_size : constant natural
  -- := max(1, natural(square_root(float(2 * average_size * link'size)
  --       / float(t'size))));

  type elements_type is array(1 .. block_size) of t;

type sequence is
  record
    length: natural := 0; -- The length of the sequence.
    elements: elements_type; -- A prefix of the sequence.
    next: link := null; -- The next node in the list.
  end record;
  -- Elements[1 .. min(length, block_size)] contains data.
end generic_sequence_pkg;

-- seq_b.a
-- $Source: /n/gemini/work/bayram/AYACC/parser/psdl_ada.lib/RCS/seq_b.a,v $
-- $Date: 1991/09/24 10:42:27 $
-- $Revision: 1.5 $

-- Warning: due to a bug in vedix Ada version 6.0.
-- it is necessary to patch the definitions of
-- generic_remove and generic_member,
-- to introduce local copies of procedure parameters (such as local_x)
-- to work around a case where variable references get confused.
with unchecked_deallocation;
with lookahead_pkg; use lookahead_pkg;
with delimiter_pkg; use delimiter_pkg;

-- generic
-- type t is private;
-- block size: in natural := 32;
package body generic_sequence pkg is
use natural_set_pkg; -- For the domain operation.

recycle_list: link := null; -- The recycle list for recycling storage.
nodes_in_recycle_list: natural := 0; -- The length of the recycle list.
nodes_in_use: natural := 0; -- The number of sequence heap nodes in use.
-- Invariant: nodes_in_recycle_list = length(recycle_list) <= nodes_in_use.

-- Local subprogram declarations.
function copy_list(l: link) return link;
function create(len: natural; e: elements_type; next: link) return link;
function token return character;
-- End local subprogram declarations.

-- Constant declarations.
is_blank: constant delimiter array := initialize_delimiter_array;
-- End constant declarations.

procedure empty(s: out sequence) is
  sl: sequence; -- Default initialization gives an empty sequence.
begin
  s := sl;
end empty;

procedure add(x: in t; s: in out sequence) is
begin
  s.length := s.length + 1;
  if s.length <= block_size then
    s.elements(s.length) := x;
  elsif s.next = null then
    s.next := create(1, (others => x), null);
  else add(x, s.next.all);
  end if;
end add;

-- generic
-- with function eq(x, y: t) return boolean is <>;
procedure generic_remove(x: in t; s: in out sequence) is
  -- Remove all instances of x from s.
  ss: sequence; -- Initialized to empty.
  local_x: t := x; -- patch to work around compiler bug, verdix version 6.0.
begin
  -- Begin expansion of FOREACH loop macro.
  declare
procedure loop_body(y: t) is
  begin if not eq(local_x, y) then add(y, ss); end if;
end loop_body;

procedure execute_loop is new generic_scan(loop_body);
begin
  execute_loop(s);
end;

-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the m4 source file
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
recycle(s);
s := ss;
end generic_remove;

procedure append(s1, s2: in sequence; s: out sequence) is
  ss: sequence;
  begin
    -- Begin expansion of FOREACH loop macro.
    declare
      procedure loop_body(x: t) is
        begin add(x, ss);
      end loop_body;
      procedure execute_loop is new generic_scan(loop_body);
      begin
        execute_loop(s1);
      end;
    end;
    -- LIMITATIONS: Square brackets are used as macro quoting characters,
    -- so you must write [[x]] in the m4 source file
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the lower case spellings of
    -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
    -- or must quote them like this: [define].
    -- The implementation requires each package to be generated by
    -- a separate call to m4: put each package in a separate file.
    -- Exit and return statements inside the body of a FOREACH loop
    -- may not work correctly if FOREACH loops are nested.
    -- An expression returned from within a loop body must not
    -- mention any index variables of the loop.
    -- End expansion of FOREACH loop macro.
    declare
      *261

261
procedure loop_body(x: t) is
    begin add(x, ss);
    end loop_body;
procedure execute_loop is new generic_scan(loop_body);
begin
    execute_loop(s2);
end;

-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the m4 source file
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generalized by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
s := ss;
end append;

function fetch(s: sequence; n: natural) return t is
begin
    if n > s.length then raise bounds_error;
    elsif n <= block_size then return s.elements(n);
    else return fetch(s.next.all, n - block_size);
    end if;
end fetch;

procedure fetch(sl: sequence; ia: index_array; a: out sequence) is
    as: sequence;
    -- Initialized to empty.
begin
    for i in ia'range loop
        add(fetch(sl, ia(i)), ss);
    end loop;
    s := ss;
end fetch;

procedure fetch(sl: sequence; low, high: natural; s: out sequence) is
    -- sl[low .. high]
    ss: sequence; -- Initialized to empty.
begin
    for i in low .. high loop
        add(fetch(sl, i), ss);
    end loop;
    s := ss;
end fetch;

function length(s: sequence) return natural is

262
begin
    return s.length;
end length;

function domain(s: sequence) return natural_set is
    ns: natural_set;
begin
    empty(ns);
    for i in 1..s.length loop
        add(i, ns);
    end loop;
    return ns;
end domain;

-- generic
-- with function eq(x, y: t) return boolean is <>;
function generic_member(x: t; s: sequence) return boolean is
    local_x: t := x; -- patch to work around compiler bug, verdix version 6.0.
begin
    -- Begin expansion of FOREACH loop macro.
    declare
        procedure loop_body(y: t) is
            begin
                if eq(local_x, y) then raise return_from_foreach; end if;
            end loop_body;
        procedure execute_loop is new generic_scan(loop_body);
        execute_loop(s);
    exception
        when return_from_foreach => return true;
    end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write [[x]] in the m4 .source file
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: (define).
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
    return(false);
end generic_member;

-- generic
-- with function eq(x, y: t) return boolean is <>;
function generic_part_of(sl, s2: sequence) return boolean is
    n: natural := 0;
    -- The definition of "matches_at" is nested inside "member"
to provide access to the generic function parameter "eq".

```pascal
function matches_at(sl, s2: sequence; n: natural) return boolean is
i: natural := 0;
begin
while i < length(sl) loop
  -- Invariant: sl[i .. i] = s2[n .. n+i-1]
  if eq(fetch(sl, i + 1), fetch(s2, n + 1)) then i := i + 1;
  else return false; end if;
end loop;
return true;
end matches_at;
```
function generic_less_than_or_equal(s1, s2: sequence) return boolean is
  function lt is new generic_less_than;
  function equal is new generic_equal(eq);
begin
  return lt(s1, s2) or else equal(s1, s2);
end generic_less_than_or_equal;

-- generic
-- with function "<"(x, y: t) return boolean is <>;
function generic_greater_than(s1, s2: sequence) return boolean is
  function lt is new generic_less_than;
  function equal is new generic_equal(eq);
begin
  return lt(s2, s1);
end generic_greater_than;

-- generic
-- with function "<"(x, y: t) return boolean is <>;
-- with function eq(x, y: t) return boolean is <>;
function generic_greater_or_equal(s1, s2: sequence) return boolean is
  function lt is new generic_less_than;
  function equal is new generic_equal(eq);
begin
  return lt(s2, s1) or else equal(s1, s2);
end generic_greater_or_equal;

-- generic
-- with function eq(x, y: t) return boolean is <>;
function generic_subsequence(s1, s2: sequence) return boolean is
  i1, i2: natural := 0;
begin
  while i1 < s1.length loop
    -- Invariant: subsequence(s1[i1 .. i1], s2[i2 .. i2]).
    -- Invariant: i1 <= s1.length & i2 <= s2.length.
    if i2 = s2.length then return false; end if;
    if eq(fetch(s1, i1 + 1), fetch(s2, i2)) then i1 := i1 + 1; end if;
  end loop;
  return true;
end generic_subsequence;

-- The above algorithm can be speeded up by doing parallel
-- scans of s1 and s2, eliminating the use of fetch.
-- This was not done because it is complicated
-- and because we do not expect this to be a frequent operation.

-- generic
-- with function "<"(x, y: t) return boolean is <>;
-- with function successor(x: t) return t;
-- ALL(x y: t :: x < y => successor(x) <= y)
procedure generic_interval(x1, x2: t; s: out sequence) is
  ss: sequence; -- initialized to empty.
y: t := x1;
begi
while not (x2 < y) loop -- Invariant: x1 <= y.
    add(y, ss);
    y := successor(y);
end loop;

s := ss;
end generic_interval;

-- generic
-- type et is private;
-- type st is private; -- st = sequence(et)
-- with function f(x: et) return t;
-- with function length(s: st) return natural is <>;
-- with function fetch(s: st; n: natural) return et is <>;

procedure generic_apply(sl: st; s2: out sequence) is
    ss: sequence; -- Initialized to empty.
begin
    for i in 1 .. length(sl) loop
        add(f(fetch(sl, i)), ss);
    end loop;

    s2 := ss;
end generic_apply;

-- generic
-- with function f(x, y: t) return t;
-- identity: t;
function generic_reduce(s: sequence) return t is
    x: t := identity;
begin
    -- Begin expansion of FOREACH loop macro.
    declare
        procedure loop_body(y: t) is
            begin x := f(y, x);
        end loop_body;
        procedure execute_loop is new generic_scan(loop_body);
    begin
        execute_loop(s);
    end;
    -- LIMITATIONS: Square brackets are used as macro quoting characters,
    -- so you must write [[x]] in the m4 source file
    -- to get [x] in the generated Ada code.
    -- Ada programs using FOREACH loops must avoid the lower case spellings of
    -- the identifier names "DEFINE", "UNDEFINE", and "DNL",
    -- or must quote them like this: [define].
    -- The implementation requires each package to be generated by
    -- a separate call to m4; put each package in a separate file.
    -- Exit and return statements inside the body of a FOREACH loop
    -- may not work correctly if FOREACH loops are nested.
    -- An expression returned from within a loop body must not
    -- mention any index variables of the loop.
    -- End expansion of FOREACH loop macro.
    return x;
end generic_reduce;

-- generic
-- with function f(x, y: t) return t;
function generic_reducel(s: sequence) return t is
  x: t;
  i: natural := 1;
begin
  if s.length = 0 then raise empty_reduction_undefined; end if;
  x := fetch(s, 1);
-- Begin expansion of FOREACH loop macro.
declare
  procedure loop_body(y: t) is
    begin if i > 1 then x := f(y, x); end if; i := i + 1;
    end loop_body;
  procedure execute_loop is new generic_scan(loop_body);
begin
  execute_loop(s);
end;
-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write \([x]\) in the m4 source file
-- to get \[x\] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop macro.
return x;
end generic_reducel;

procedure generic_scan(s: sequence) is
  t: sequence := s;
begins
  while t.next /= null loop
    for i in 1 .. block_size loop
      generate(t.elements(i));
    end loop;
    t := t.next.all;
  end loop;
  for i in 1 .. t.length loop
    generate(t.elements(i));
  end loop;
end generic_scan;

-- System functions and local subprograms.
procedure assign(x: out sequence; y: in sequence) is
begin
  x.length := y.length;
  x.elements := y.elements;
  x.next := copy_list(y.next);
end assign;

function copy_list(l: link) return link is
begin
  if l = null then return l;
  else return create(l.length, l.elements, copy_list(l.next)); end if;
end copy_list;

function create(len: natural; e: elements_type; next: link) return link is
begin
  nodes_in_use := nodes_in_use + 1;
  if recycle_list = null then
    return new sequence'(len, e, next);
  else
    l := recycle_list;
    recycle_list := recycle_list.next;
    nodes_in_recycle_list := nodes_in_recycle_list - 1;
    l.length := len; l.elements := e; l.next := next;
    return l;
  end if;
end create;

procedure recycle(s: in sequence) is
l: link := s.next;
head, temp: link;
procedure free is new unchecked_deallocation(sequence, link);
begin
  while l /= null loop
    head := l; l := l.next;
    nodes_in_use := nodes_in_use - 1;
    if nodes_in_recycle_list < nodes_in_use then
      temp := recycle_list;
      recycle_list := head;
      recycle_list.next := temp;
      nodes_in_recycle_list := nodes_in_recycle_list + 1;
    else free(head);
    end if;
  end loop;
end recycle;

-- generic
--   with procedure input(item: out t) is <>;
procedure generic_input(item: out sequence) is
x: t;
s: sequence; -- Working copy of the result, initialized to empty.
begin
  if token /= ascii.l_bracket then raise data_error; end if;
skip_char; -- Pass over the opening left bracket.
while token /= ascii.r_bracket loop
  input(x); -- Read and pass over the next element of the sequence.
  add(x, s); -- Add the element to the sequence.
  if token = ',' then skip_char; -- Another element should follow.
  elsif token /= ascii.r_bracket then raise data_error;
  -- if there is no comma we should be at the end of the sequence.
  end if;
end loop; -- Now the closing right bracket is the lookahead character.
skip_char;
item := s;
exception
  when others => raise data_error;
end generic_input;

-- generic
-- with procedure input(item: out t) is <>;
procedure generic_file_input(item: out sequence; file: in file_type) is
  procedure get_sequence is new generic_input;
begin
  set_input(file); -- Connect the lookahead stream to the file.
  get_sequence(item);
  set_input(standard_input); -- Restore the standard input file.
end generic_file_input;

function token return character is
  -- Blank is a constant array, see top of package body.
begin
  -- Advance the lookahead stream to a non-blank character.
  while is_blank(peek) loop skip_char; end loop;
  -- Return the character without removing it from the stream.
  return peek;
end token;

-- generic
-- with procedure put(item: in t) is <>;
procedure generic_put(item: in sequence) is
begin
  put(ascii.l_bracket);
  if length(item) >= 1 then put(fetch(item, 1)); end if;
  for i in 2 .. length(item) loop
    put(", ");
    put(fetch(item, i));
  end loop;
  put(ascii.r_bracket);
end generic_put;

-- generic
-- with procedure put(file: in file_type; item: in t) is <>;
end
procedure generic_file_put(file: in file_type; item: in sequence) is
begin
    put(file, ascii.l_bracket);
    if length(item) >= 1 then put(file, fetch(item, 1)); end if;
    for i in 2 .. length(item) loop
        put(file, ", ");
        put(file, fetch(item, i));
    end loop;
    put(file, ascii.r_bracket);
end generic_file_put;
end generic_sequence_pkg;
APPENDIX O. GENERIC STACK PACKAGE

-- stacks.a

with lists; --| Implementation uses lists. (private)

generic
  type elem_type is private; --| Component element type.

package stack_pkg is

--| Overview:
--| This package provides the stack abstract data type. Element type is
--| a generic formal parameter to the package. There are no explicit
--| bounds on the number of objects that can be pushed onto a given stack.
--| All standard stack operations are provided.
--|
--| The following is a complete list of operations, written in the order
--| in which they appear in the spec. Overloaded subprograms are followed
--| by (n), where n is the number of subprograms of that name.
--|
--| Constructors:
--| create
--| push
--| pop (2)
--| copy
--| Query Operations:
--| top
--| size
--| is_empty
--| replace_top
--| reverse_stack
--| Heap Management:
--| destroy

--| Notes:
type stack is private;  --| The stack abstract data type.

-- Exceptions:

uninitialized_stack: exception;
  --| Raised on attempt to manipulate an uninitialized stack object.
  --| The initialization operations are create and copy.

empty_stack: exception;
  --| Raised by some operations when empty.

-- Constructors:

function create
  return stack;
  --| Effects:
  --| Return the empty stack.

procedure push(s: in out stack;
  e: elem_type);
  --| Raises: uninitialized_stack
  --| Effects:
  --| Push e onto the top of s.
  --| Raises uninitialized_stack iff s has not been initialized.

procedure pop(s: in out stack);
  --| Raises: empty_stack, uninitialized_stack
  --| Effects:
  --| Pops the top element from s, and throws it away.
  --| Raises empty_stack iff s is empty.
  --| Raises uninitialized_stack iff s has not been initialized.

procedure pop(s: in out stack;
  e: out elem_type);
  --| Raises: empty_stack, uninitialized_stack
  --| Effects:
  --| Pops the top element from s, returns it as the e parameter.
  --| Raises empty_stack iff s is empty.
  --| Raises uninitialized_stack iff s has not been initialized.
procedure replace_top(e: in elem_type;
    s: in out stack);

  --| Raises: empty_stack, uninitialized_stack
  --| Effects:
  --| replaces the top of the stack with the next e, ..
  --| .. returns s as the modified stack.
  --| Raises empty_stack iff s is empty.
  --| Raises uninitialized_stack iff s has not been initialized.

procedure reverse_stack(s: in out stack);

  --| Raises: empty_stack, uninitialized_stack
  --| Effects:
  --| reverses the order of the elements un the stack s, ..
  --| .. returns s as the modified stack.
  --| Raises empty_stack iff s is empty.
  --| Raises uninitialized_stack iff s has not been initialized.

function copy(s: stack)
  return stack;

  --| Raises: uninitialized_stack
  --| Return a copy of s.
  --| Stack assignment and passing stacks as subprogram parameters
  --| result in the sharing of a single stack value by two stack
  --| objects; changes to one will be visible through the others.
  --| copy can be used to prevent this sharing.
  --| Raises uninitialized_stack iff s has not been initialized.

-- Queries:

function top(s: stack)
  return elem_type;

  --| Raises: empty_stack, uninitialized_stack
  --| Effects:
  --| Return the element on the top of s. Raises empty_stack iff s is
  --| empty.
  --| Raises uninitialized_stack iff s has not been initialized.
function size(s: stack)
    return natural;

    --| Raises: uninitialized_stack
    --| Effects:
    --| Return the current number of elements in s.
    --| Raises uninitialized_stack iff s has not been initialized.

function is_empty(s: stack)
    return boolean;

    --| Raises: uninitialized_stack
    --| Effects:
    --| Return true iff s is empty.
    --| Raises uninitialized_stack iff s has not been initialized.

-- Heap Management:

procedure destroy(s: in out stack);

    --| Effects:
    --| Return the space consumed by s to the heap. No effect if s is
    --| uninitialized. In any case, leaves s in uninitialized state.

private

package elem_list_pkg is new lists(elem_type);
subtype elemlist is elem_list_pkg.list;

type stack_rec is
    record
        size: natural := 0;
        elts: elemlist := elem_list_pkg.create;
    end record;

type stack is access stack_rec;

    --| Let an instance of the representation type, r, be denoted by the
    --| pair, <size, elts>. Dot selection is used to refer to these
    --| components.
    --|
    --| Representation Invariants:
    --| r /= null
    --| elem_list_pkg.length(r.elts) = r.size.
-- Abstraction Function:
-- A(<size, elem_list_pkg.create>) = stack_pkg.create.
-- A(<size, elem_list_pkg.attach(e, l)>) = push(A(<size, l>), e).

end stack_pkg;

-- stack_b.a

package body stack_pkg is

-- Overview:
-- Implementation scheme is totally described by the statements of the
-- representation invariants and abstraction function that appears in
-- the package specification. The implementation is so trivial that
-- further documentation is unnecessary.

use elem_list_pkg;

-- Constructors:

function create return stack is
begin
return new stack rec'(size => 0, elts => create);
end create;

procedure push(s: in out stack; e: elem_type) is
begin
s.size := s.size + 1;
s.elts := attach(e, s.elts);
exception
when constraint_error =>
   raise uninitialized_stack;
end push;
procedure pop(s: in out stack) is
begin
    DeleteHead(s.elts);
    s.size := s.size - 1;
exception
    when EmptyList =>
        raise empty_stack;
    when constraint_error =>
        raise uninitialized_stack;
end pop;

procedure pop(s: in out stack; e: out elemtype) is
begin
    e := FirstValue(s.elts);
    DeleteHead(s.elts);
    s.size := s.size - 1;
exception
    when EmptyList =>
        raise empty_stack;
    when constraint_error =>
        raise uninitialized_stack;
end pop;

procedure replace_top(e: in elem_type; s: in out stack) is
begin
    temp_elem: elem_type;
    begin
        pop(s, temp_elem);
        push(s, e);
        push(s, temp_elem);
    exception
        when EmptyList =>
            raise empty_stack;
        when constraint_error =>
            raise uninitialized_stack;
end replace_top;

procedure reverse_stack(s: in out stack) is
begin
    ...
temp : stack := create;
begin
    while not is_empty(s) loop
        push(temp, top(s));
        pop(s);
    end loop;
    s := copy(temp);
    destroy(temp);
exception
    when EmptyList =>
        raise empty_stack;
    when constraint_error =>
        raise uninitialized_stack;
end reverse_stack;

function copy(s: stack)
    return stack is
begin
    if s = null then raise uninitialized_stack; end if;

    return new stack_rec'(size => s.size,
                           elts => copy(s.elts));
end;

-- Queries:

function top(s: stack)
    return elem_type is
begin
    return FirstValue(s.elts);
exception
    when EmptyList =>
        raise empty_stack;
    when constraint_error =>
        raise uninitialized_stack;
end top;

function size(s: stack)
    return natural is
begin
    return s.size;
exception
    when constraint_error =>
raise uninitialized_stack;
end size;

function is_empty(s: stack) return boolean is
begin
  return s.size = 0;
exception
  when constraint_error =>
    raise uninitialized_stack;
end is_empty;

-- Heap Management:

procedure destroy(s: in out stack) is
  procedure free_stack is
    new unchecked_deallocation(stack_rec, stack);
  begin
    destroy(s.elts);
    free_stack(s);
    exception
      when constraint_error => -- stack is null
        return;
    end destroy;
end stack_pkg;
APPENDIX P. GENERIC LIST PACKAGE

--:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:
-- lists.a
--:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:...:

generic
    type ItemType is private;  --| This is the data being manipulated.

    with function Equal ( X,Y: in ItemType) return boolean is
        "=*;  --| This allows the user to define
        --| equality on ItemType. For instance
        --| if ItemType is an abstract type
        --| then equality is defined in terms of
        --| the abstract type. If this function
        --| is not provided equality defaults to
        --| "=.

package Lists is

    --| This package provides singly linked lists with elements of type
    --| ItemType, where ItemType is specified by a generic parameter.

    --| Overview
    --| When this package is instantiated, it provides a linked list type for
    --| lists of objects of type ItemType, which can be any desired type. A
    --| complete set of operations for manipulation, and releasing
    --| those lists is also provided. For instance, to make lists of strings,
    --| all that is necessary is:
    --|
    --| type StringType is string(1..10);
    --|
    --| package StrList is new Lists(StringType); use StrList;
    --|
    --| L:List;
    --| S:StringType;
    --|
    --| Then to add a string S, to the list L, all that is necessary is
    --|
    --| L := Create;
    --| Attach(S,L);
    --|
    --| This package provides basic list operations.
    --|
    --| Attach                           append an object to an object, an object to a list,
    --|                        or a list to an object, or a list to a list.
Copycopy a list using := on elements

CopyDeep copy a list by copying the elements using a copy operation provided by the user

Create Creates an empty list

DeleteHead removes the head of a list

DeleteItem delete the first occurrence of an element from a list

DeleteItems delete all occurrences of an element from a list

Destroy remove a list

DestroyDeep destroy a list as well as the elements in that list

Equal are two lists equal

FirstValue get the information from the first element of a list

Forward advances an iterator

IsEmpty determines whether a given element is in a given list

IsEmpty returns true if the list is empty

LastValue return the last value of a list

Length Returns the length of a list

MakeList this takes a single element and returns a list

MakeListIter prepares for an iteration over a list

More are there any more items in the list

Next get the next item in a list.

ReplaceHead replace the information at the head of the list

ReplaceTail replace the tail of a list with a new list

Tail get the tail of a list

CellValue this takes an iterator and returns the value of the element whose position the iterator holds

---

N/A: Effects, Requires, Modifies, and Raises.

Notes

Types

---

type List is private;
type ListIter is private;

---

Exceptions

---

CircularList : exception; --| Raised if an attempt is made to
---| create a circular list. This
---| results when a list is attempted
---| to be attached to itself.

EmptyList : exception; --| Raised if an attempt is made to
---| manipulate an empty list.

ItemNotPresent : exception; --| Raised if an attempt is made to
---| remove an element from a list in
which it does not exist.

NoMore : exception;  -- Raised if an attempt is made to
-- get the next element from a list
-- after iteration is complete.

Operations

-- |

procedure Attach(   -- appends List2 to List1
    List1: in out List;  -- The list being appended to.
    List2: in List       -- The list being appended.
);

-- | Raises
-- | CircularList

-- | Effects
-- | Appends List1 to List2. This makes the next field of the last element
-- | of List1 refer to List2. This can possibly change the value of List1
-- | if List1 is an empty list. This causes sharing of lists. Thus if
-- | user Destroys List1 then List2 will be a dangling reference.
-- | This procedure raises CircularList if List1 equals List2. If it is
-- | necessary to Attach a list to itself first make a copy of the list and
-- | attach the copy.

-- | Modifies
-- | Changes the next field of the last element in List1 to be List2.

function Attach(   -- Creates a new list containing the two
    Element1: in ItemType;  -- This will be first element in list.
    Element2: in ItemType   -- This will be second element in list
) return List;

-- | Effects
-- | This creates a list containing the two elements in the order
-- | specified.

procedure Attach(   -- List L is appended with Element.
    L: in out List;      -- List being appended to.
    Element: in ItemType -- This will be last element in L list.
);
procedure Attach(  
    Element: in ItemType;  
    L: in out List)  
--| Makes Element first item in list L.  
--| This will be the first element in list.  
--| The List which Element is being  
--| prepended to.

);  

function Attach(  
    List1: in List;  
    List2: in List)  
--| attaches two lists  
--| first list  
--| second list
) return List;

--| Raises  
--| CircularList

--| Effects  
--| This returns a list which is List1 attached to List2. If it is desired  
--| to make List1 be the new attached list the following ada code should be  
--| used.  
--|  
--| List1 := Attach (List1, List2);  
--| This procedure raises CircularList if List1 equals List2. If it is  
--| necessary to Attach a list to itself first make a copy of the list and  
--| attach the copy.

function Attach(  
    Element: in ItemType;  
    L: in List)  
--| prepends an element onto a list  
--| element being prepended to list  
--| List which element is being added  
--| to
) return List;

--| Effects  
--| Returns a new list which is headed by Element and followed by L.
function Attach (L: in List; Element: in ItemType) --| Adds an element to the end of a list

) return L \ Element;

--| Effects
--| Returns a new list which is L followed by Element.

function Copy(L: in List) --| Returns a copy of L.

function CopyDeep(L: in List) --| Returns a copy of L using a user supplied copy function. This is helpful if the type of a list is an abstract data type.

generic

with function Copy(I: in ItemType) return ItemType;

function Create --| Returns an empty List

return List;

procedure DeleteHead(L: in out List) --| Remove the head element from a list.

); -- RAISES

-- EmptyList
procedure deleteItem\(\) \begin{verbatim}
L: in out List;  \hspace{1em} list element is being removed from
Element: in ItemType \hspace{1em} element being removed
\end{verbatim}
\end{verbatim}

--- EFFECTS
--- This will return the space occupied by the first element in the list
--- to the heap. If sharing exists between lists this procedure
--- could leave a dangling reference. If L is empty EmptyList will be
--- raised.

-----------------------------------------------

function deleteItem\(\) \begin{verbatim}
L: in List; \hspace{1em} list element is being removed from
Element: in ItemType \hspace{1em} element being removed
\end{verbatim}
\end{verbatim}

--- EFFECTS
--- Removes the first element of the list equal to Element. If there is
--- not an element equal to Element than ItemNotPresent is raised.

--- MODIFIES
--- This operation is destructive, it returns the storage occupied by
--- the elements being deleted.

-----------------------------------------------

function deleteItems\(\) \begin{verbatim}
L: in List; \hspace{1em} The List element is being removed from
Element: in ItemType \hspace{1em} element being removed
\end{verbatim}
\end{verbatim}

--- EFFECTS
--- This function returns a copy of the list L which has all elements which
--- have value Element removed.

-----------------------------------------------

procedure deleteItems\(\) \begin{verbatim}
L: in out List; \hspace{1em} The List element is being removed from
Element: in ItemType \hspace{1em} element being removed
\end{verbatim}
\end{verbatim}

284
--| EFFECTS
--| This procedure removes all occurrences of Element from the List L. This
--| is a destructive procedure.

procedure Destroy (  --| removes the list
L: in out List  --| the list being removed
);

--| Effects
--| This returns to the heap all the storage that a list occupies. Keep in
--| mind if there exists sharing between lists then this operation can leave
--| dangling references.

generic
  with procedure Dispose (I :in out ItemType);

procedure DestroyDeep (  --| Destroy a list as well as all objects which
  --| comprise an element of the list.
    L :in out List
);

--| OVERVIEW
--| This procedure is used to destroy a list and all the objects contained
--| in an element of the list. For example if L is a list of lists
--| then destroy L does not destroy the lists which are elements of L.
--| DestroyDeep will now destroy L and all the objects in the elements of L.
--| The procedure Dispose is a procedure which will destroy the objects which
--| comprise an element of a list. For example if package L was a list
--| of lists then Dispose for L would be the Destroy of list type package L was
--| instantiated with.

--| REQUIRE
--| This procedure requires no sharing between elements of lists.
--| For example if L_int is a list of integers and L_of_L_int is a list
--| of lists f integers and two elements of L_of_L_int have the same value
--| then doing a DestroyDeep will cause an access violation to be raised.
--| The best way to avoid this is not to have sharing between list elements
--| or use copy functions when adding to the list of lists.

function FirstValue(  --| returns the contents of the first record of the
L: in List  --| the list whose first element is being
    --| returned
);
function IsEmpty(  
  L: in List  
) return boolean;

function IsInList(  
  L: in List;  
  Element: in ItemType  
) return boolean;
---| Effects
---| Walks down the list L looking for an element whose value is Element.

---| function LastValue:
---| Returns the contents of the last record of the list.
---| L: in List The list whose first element is being returned.
) return ItemType;

---| Raises
---| EmptyList
---| Effects
---| Returns the last element in a list. If the list is empty EmptyList is raised.

---| function Length:
---| count the number of elements on a list
---| L: in List list whose length is being computed
) return integer;

---| function MakeList:
---| This takes in an element and returns a List.
---| E :in ItemType
) return List;

---| function MakeListIter:
---| Sets a variable to point to the head of the list. This will be used to prepare for iteration over a list.
---| L: in List The list being iterated over.
) return ListIter;

---| This prepares a user for iteration operation over a list. The iterater is an operation which returns successive elements of the list on successive calls to the iterator. There needs to be a mechanism which marks the position in the list, so on successive calls to the Next operation the next item in the list can be returned. This is the function of the MakeListIter and the type ListIter. MakeIter just sets the Iter to the beginning of the list. On subsequent calls to Next the Iter is updated with each call.
function More(  
  --| Returns true if there are more elements in  
  --| the and false if there aren't any more  
  --| the in the list.  
  L: in ListIter  
  --| List being checked for elements.  
) return boolean;

-----------------------------------------------------------------

procedure Next(  
  --| This is the iterator operation. Given  
  --| a ListIter in the list it returns the  
  --| current item and updates the ListIter.  
  --| If ListIter is at the end of the list,  
  --| More returns false otherwise it  
  --| returns true.  
  Place: in out ListIter;  
  --| The Iter which marks the position in  
  --| the list.  
  Info: out ItemType  
  --| The element being returned.  
);

--| The iterators subprograms MakeListIter, More, and Next should be used  
--| in the following way:
--|
--|   L: List;  
--|   Place: ListIter;  
--|   Info: SomeType;  
--|
--|   Place := MakeListIter(L);  
--|   while ( More(Place) ) loop  
--|     Next(Place, Info);  
--|     process each element of list L;  
--|

-----------------------------------------------------------------

procedure ReplaceHead(  
  --| Replace the Item at the head of the list  
  --| with the parameter Item.  
  L: in out List;  
  --| The list being modified.  
  Info: in ItemType  
  --| The information being entered.  
);

--| Raises  
--| EmptyList  

--| Effects  
--| Replaces the information in the first element in the list. Raises  
--| EmptyList if the list is empty.

-----------------------------------------------------------------
procedure ReplaceTail ( L: in out List; NewTail: in List );

--| Replace the Tail of a list 
--| with a new list.
--| List whose Tail is replaced.
--| The list which will become the 
--| tail of Oldlist.

--| Raises
--| EmptyList
--| Effects
--| Replaces the tail of a list with a new list. If the list whose tail 
--| is being replaced is null EmptyList is raised.

--------------------------------------------------------------------

function Tail( L: in List );

--| returns the tail of a list L
--| the list whose tail is being returned

--| Raises
--| EmptyList
--| Effects
--| Returns a list which is the tail of the list L. Raises EmptyList if 
--| L is empty. If L only has one element then Tail returns the Empty 
--| list.

--------------------------------------------------------------------

function CellValue ( I : in ListIter );

--| return ItemType;

--| OVERVIEW
--| This returns the value of the element at the position of the iterator.
--| This is used in conjunction with Forward.

--------------------------------------------------------------------

function Equal( List1: in List; List2: in List );

--| compares list1 and list2 for equality
--| first list
--| second list

--| Effects
--| Returns true if for all elements of List1 the corresponding element 
--| of List2 has the same value. This function uses the Equal operation 
--| provided by the user. If one is not provided then = is used.
private
  type Cell;

  type List is access Cell;  --| pointer added by this package
                              --| in order to make a list

  type Cell is
    record
      Info: ItemType;
      Next: List;
    end record;

  type ListIter is new List;  --| Cell for the lists being created
                               --| This prevents Lists being assigned to
                               --| iterators and vice versa

end Lists;

with unchecked_deallocation;

package body Lists is

  procedure Free is new unchecked_deallocation (Cell, List);

  function Last (L: in List) return List is
    Place_In_L: List;
    Temp_Place_In_L: List;
    --| Link down the list L and return the pointer to the last element
    --| of L. If L is null raise the EmptyList exception.
    begin
      if L = null then
        raise EmptyList;
      else
        --| Link down L saving the pointer to the previous element in

      end if;
  end function Last;
--| Temp_Place_In_L. After the last iteration Temp_Place_In_L
--| points to the last element in the list.

Place_In_L := L;
while Place_In_L /= null loop
    Temp_Place_In_L := Place_In_L;
    Place_In_L := Place_In_L.Next;
end loop;
return Temp_Place_In_L;
end if;
end Last;

-----------------------------------------------------------------------------------------------------------------------

procedure Attach (List1: in out List;
                  List2: in   List) is
    EndOfList1: List;

    --| Attach List2 to List1.
    --| If List1 is null return List2
    --| If List1 equals List2 then raise CircularList
    --| Otherwise get the pointer to the last element of List1 and change
    --| its Next field to be List2.

    begin
        if List1 = null then
            List1 := List2;
            return;
        elsif List1 = List2 then
            raise CircularList;
        else
            EndOfList1 := Last (List1);
            EndOfList1.Next := List2;
        end if;
    end Attach;

-----------------------------------------------------------------------------------------------------------------------

procedure Attach (L: in out List;
                  Element: in   ItemType) is
    NewEnd: List;

    --| Create a list containing Element and attach it to the end of L

    begin
        NewEnd := new Cell'(Info => Element, Next => null);
        Attach (L, NewEnd);
    end;
function Attach (Element1: in ItemType;
    Element2: in ItemType) return List is
    NewList: List;
    -- Create a new list containing the information in Element1 and
    -- attach Element2 to that list.
    begin
        NewList := new Cell'(Info => Element1, Next => null);
        Attach (NewList, Element2);
        return NewList;
    end;

procedure Attach (Element: in ItemType;
                L: in out List) is
    -- Create a new cell whose information is Element and whose Next
    -- field is the list L. This prepends Element to the List L.
    begin
        L := new Cell'(Info => Element, Next => L);
    end;

function Attach (List1: in List;
    List2: in List) return List is
    Last_Of_List1: List;
    begin
        if List1 = null then
            return List2;
        elsif List1 = List2 then
            raise CircularList;
        else
            Last_Of_List1 := Last (List1);
            Last_Of_List1.Next := List2;
            return List1;
        end if;
    end Attach;

function Attach (L: in List;
    Element: in ItemType) return List is
NewEnd: List;
Last_Of_L: List;

--| Create a list called NewEnd and attach it to the end of L.
--| If L is null return NewEnd
--| Otherwise get the last element in L and make its Next field
--| NewEnd.

begin
    NewEnd := new Cell'(Info => Element, Next => null);
    if L = null then
        return NewEnd;
    else
        Last_Of_L := Last (L);
        Last_Of_L.Next := NewEnd;
        return L;
    end if;
end Attach;

function Attach (Element: in ItemType; L: in List) return List is
begin
    return (new Cell'(Info => Element, Next => L));
end Attach;

function Copy (L: in List) return List is
--| If L is null return null
--| Otherwise recursively copy the list by first copying the information
--| at the head of the list and then making the Next field point to
--| a copy of the tail of the list.
begin
    if L = null then
        return null;
    else
        return new Cell'(Info => L.Info, Next => Copy (L.Next));
    end if;
end Copy;

function CopyDeep (L: in List) return List is
begin
  if L = null then
    raise EmptyList;
  else
    TempList := L.Next;
    Free (L);
    L := TempList;
  end if;
end DeleteHead;

function DeleteItem (L: in List; Element: in ItemType) return List is
  I :List;
  Result :List;
begin
  if L is null then
    return null;
  end if;
  if L.Info = Element then
    I := new Cell'( Info => Copy (L.Info), Next => CopyDeep(L.Next));
  else
    I := DeleteItem (L.Next, Element);
  end if;
  return new Cell'( Info => Copy (L.Info), Next => I);
end DeleteItem;
begin
  ALGORITHM
  Attach all elements of L to Result except the first element in L
  whose value is Element. If the current element pointed to by I
  is not equal to element or the element being skipped was found
  then attach the current element to Result.

  I := L;
  while (I /= null) loop
    if (not Equal (I.Info, Element)) or (Found) then
      Attach (Result, I.Info); 
    else
      Found := true;
    end if;
    I := I.Next;
  end loop;
  return Result;
end DeleteItem;

function DeleteItems (-- I remove all occurrences of Element
  L: in List; -- I The List element is being removed from
  Element: in ItemType -- I element being removed
) return List is
  I :List;
  Result :List;
begin
  ALGORITHM
  Walk over one list L and if the current element does not equal
  Element then attach it to the list to be returned.

  I := L;
  while I /= null loop
    if not Equal (I.Info, Element) then
      Attach (Result, I.Info);
    end if;
    I := I.Next;
  end loop;
  return Result;
end DeleteItems;

procedure DeleteItem (L: in out List;
  Element: in ItemType ) is
  Temp_L :List;
-- | Remove the first element in the list with the value Element.
-- | If the first element of the list is equal to element then
-- | remove it. Otherwise, recurse on the tail of the list.

begin
  if Equal(L.Info, Element) then
    DeleteHead(L);
  else
    DeleteItem(L.Next, Element);
  end if;
end DeleteItem;

--------------------------------------------------------------------------

procedure DeleteItems (L: in out List;
                         Element: in ItemType) is

  Place_In_L :List;  -- | Current place in L.
  Last_Place_In_L :List;  -- | Last place in L.
  Temp_Place_In_L :List;  -- | Holds a place in L to be removed.

  -- | Walk over the list removing all elements with the value Element.

begin
  Place_In_L := L;
  Last_Place_In_L := null;
  while (Place_In_L /= null) loop
    -- | Found an element equal to Element
    if Equal(Place_In_L.Info, Element) then
      -- | If Last_Place_In_L is null then we are at first element
      -- | in L.
      if Last_Place_In_L = null then
        Temp_Place_In_L := Place_In_L;
        L := Place_In_L.Next;
      else
        Temp_Place_In_L := Place_In_L;
      end if;
      -- | Relink the list Last's Next gets Place's Next
      Last_Place_In_L.Next := Place_In_L.Next;
      end if;
      -- | Move Place_In_L to the next position in the list.
      -- | Free the element.
      -- | Do not update the last element in the list it remains the
      -- | same.
      Place_In_L := Place_In_L.Next;
      Free (Temp_Place_In_L);
    else
      -- | Update the last place in L and the place in L.
      L := Place_In_L.Next;
      Last_Place_In_L := Place_In_L;
    end if;
  end loop;
end DeleteItems;
Last_Place_In_L := Place_In_L;
Place_In_L := Place_In_L.Next;
end if;
end loop;

--| If we have not found an element raise an exception.

end DeleteItems;

procedure Destroy (L: in out List) is
  Place_In_L: List;
  HoldPlace: List;

  --| Walk down the list removing all the elements and set the list to
  --| the empty list.

  begin
    Place_In_L := L;
    while Place_In_L /= null loop
      HoldPlace := Place_In_L;
      Place_In_L := Place_In_L.Next;
      Free (HoldPlace);
    end loop;
    L := null;
  end Destroy;

procedure DestroyDeep (L: in out List) is
  Place_In_L: List;
  HoldPlace: List;

  --| Walk down the list removing all the elements and set the list to
  --| the empty list.

  begin
    Place_In_L := L;
    while Place_In_L /= null loop
      HoldPlace := Place_In_L;
      Place_In_L := Place_In_L.Next;
      Dispose (HoldPlace.Info);
      Free (HoldPlace);
    end loop;
    L := null;
  end DestroyDeep;
function FirstValue (L: in List) return ItemType is

--| Return the first value in the list.

begin
  if L = null then
    raise EmptyList;
  else
    return (L.Info);
  end if;
end FirstValue;

procedure Forward (I: in out ListIter) is

--| Return the pointer to the next member of the list.

begin
  if I = null then
    raise NoMore;
  else
    I := ListIter (I.Next);
  end if;
end Forward;

function IsInList (L: in List; Element: in ItemType) return boolean is

Place_In_L: List;

--| Check if Element is in L. If it is return true otherwise return false.

begin
  Place_In_L := L;
  while Place_In_L /= null loop
    if Equal(Place_In_L.Info, Element) then
      return true;
    end if;
    Place_In_L := Place_In_L.Next;
  end loop;
  return false;
end IsInList;

function IsEmpty (L: in List) return boolean is
--| Is the list L empty.

begin
return (L = null);
end IsEmpty;

function LastValue (L: in List) return ItemType is

LastElement: List;

--| Return the value of the last element of the list. Get the pointer
--| to the last element of L and then return its information.

begin
LastElement := Last (L);
return LastElement.Info;
end LastValue;

function Length (L: in List) return integer is

--| Recursively compute the length of L. The length of a list is
--| 0 if it is null or 1 + the length of the tail.

begin
if L = null then
return (0);
else
return (1 + Length (Tail (L)));
end if;
end Length;

function MakeList (E : in ItemType) return List is

begin
return new Cell' (Info => E, Next => null);
end;

function MakeListIter (L: in List) return ListIter is

--| Start an iteration operation on the list L. Do a type conversion
--| from List to ListIter.
begin
    return ListIter (L);
end MakeListIter;

function More (L: in ListIter) return boolean is

--| This is a test to see whether an iteration is complete.

begin
    return L /= null;
end;

procedure Next (Place: in out ListIter;
                 Info: out ItemType ) is

    PlaceInList: List;

--| This procedure gets the information at the current place in the List
--| and moves the ListIter to the next position in the list.
--| If we are at the end of a list then exception NoMore is raised.

begin
    if Place = null then
        raise NoMore;
    else
        PlaceInList := List(Place);
        Info := PlaceInList.Info;
        Place := ListIter(PlaceInList.Next);
    end if;
end Next;

procedure ReplaceHead (L: in out List;
                       Info: in ItemType ) is

--| This procedure replaces the information at the head of a list
--| with the given information. If the list is empty the exception
--| EmptyList is raised.

begin
    if L = null then
        raise EmptyList;
    else
        L.Info := Info;
    end if;
end ReplaceHead;
procedure ReplaceTail (L: in out List;
      NewTail: in List) is
      Temp_L: List;
begin
      --| This destroys the tail of a list and replaces the tail with
      --| NewTail. If L is empty EmptyList is raised.
      Destroy(L.Next);
      L.Next := NewTail;
      exception
      when constraint_error =>
         raise EmptyList;
      end ReplaceTail;

function Tail (L: in List) return List is
      --| This returns the list which is the tail of L. If L is null
      --| EmptyList is raised.
begin
      if L = null then
         raise EmptyList;
      else
         return L.Next;
      end if;
end Tail;

function CellValue (I: in ListIter) return ItemType is
      L: List;
begin
      -- Convert I to a List type and then return the value it points to.
      L := List(I);
      return L.Info;
end CellValue;

function Equal (List1: in List;
      List2: in List) return boolean is
      PlaceInList1: List;
      PlaceInList2: List;
      Contents1:  ItemType;

This function tests to see if two lists are equal. Two lists are equal if for all the elements of List1 the corresponding element of List2 has the same value. Thus if the 1st elements are equal and the second elements are equal and so up to n.

Thus a necessary condition for two lists to be equal is that they have the same number of elements.

This function walks over the two list and checks that the corresponding elements are equal. As soon as we reach the end of a list (PlaceInList = null) we fall out of the loop.

If both PlaceInList1 and PlaceInList2 are null after exiting the loop then the lists are equal. If they both are not null the lists aren't equal. Note that equality on elements is based on a user supplied function Equal which is used to test for item equality.

begin
  PlaceInList1 := List1;
  PlaceInList2 := List2;
  while (PlaceInList1 /= null) and (PlaceInList2 /= null) loop
    if not Equal (PlaceInList1.Info, PlaceInList2.Info) then
      return false;
    end if;
    PlaceInList1 := PlaceInList1.Next;
    PlaceInList2 := PlaceInList2.Next;
  end loop;
  return ((PlaceInList1 = null) and (PlaceInList2 = null));
end Equal;
end Lists;

--------------------------------------------------------------------------------
APPENDIX Q. UTILITY PACKAGES

package Lookahead_Pkg is
  function Peek
    return CHARACTER;
  procedure Get_Char
    ( Item : out CHARACTER );
  procedure Skip_Char;

  End_Error : exception
    renames Io_Exceptions.End_Error;
    -- Attempt to read past end of file.

end Lookahead_Pkg;

package body Lookahead_Pkg is
  Buffer : CHARACTER;
  Empty : BOOLEAN := TRUE;
    -- (-empty => buffer is the next character in the stream).

  function Peek
    return CHARACTER is
    begin -- Peek
      if Empty then
        Get(Buffer);
        Empty := False;
      end if;
      return Buffer;
    end Peek;

end Lookahead_Pkg;
procedure Get_Char
  ( Item : o't CHARACTER ) is
begin -- Get_Char
  if Empty then
    Get(Item);
  else
    Item := Buffer;
    Empty := TRUE;
  end if;
end Get_Char;

procedure Skip_Char is
begin -- Skip_Char
  if Empty then
    Get(Buffer);
  else
    Empty := TRUE;
  end if;
end Skip_Char;
end Lookahead_Pkg;

package Delimiter_Pkg is
  type DELIMITER_ARRAY is
    array (CHARACTER) of BOOLEAN;
  function Initialize_Delimiter_Array
    return DELIMITER_ARRAY;
end Delimiter_Pkg;

package body Delimiter_Pkg is
  function Initialize_Delimiter_Array
    return DELIMITER_ARRAY is
begin -- Initialize_Delimiter_Array
  return (' ' | Ascii.Ht | Ascii.Cr | Ascii.Lf => TRUE, others => False);
end Initialize_Delimiter_Array;
end Delimiter_Pkg;
APPENDIX R. PACKAGE PSDL_LEX

-- A lexical scanner generated by aflex
with text_io; use text_io;
with psdl_lex_dfa; use psdl_lex_dfa;
with psdl_lex_io; use psdl_lex_io;
-- $ line 1 "psdl_lex.l"
-- $Source: /n/gemini/work/bayram/AYACC/parser/RCS/psdl_lex.l,v $
-- $Date: 1991/09/08 07:08:33 $
-- $Revision: 1.12 $

with psdl_tokens, a_strings, psdl_concrete_type_pkg;
use psdl_tokens, a_strings, psdl_concrete_type_pkg;
use text_io;

package psdl_lex is

  lines : positive := 1;
  num_errors : natural := 0;
  List_File: text_io.file_type;

  -- in the case that one id comes right after another id
  -- we save the previous one to get around the problem
  -- that look ahead token is saved into yytext
  -- This problem occurs in the optional_generic_param if
  -- an optional type declaration comes after that.
  -- IDENTIFIER
  the_prev_id_token : psdl_id := psdl_id(a_strings.empty);
  the_id_token : psdl_id := psdl_id(a_strings.empty);

  -- STRING_LITERAL
  the_string_literal : expression := expression(a_strings.empty);

  -- INTEGER_LITERAL (psdl_id or expression)
  the_integer_token : a_string := a_strings.empty;

  -- REAL_LITERAL
  the_real_token : expression := expression(a_strings.empty);

  -- TEXT_TOKEN
  the_text_token : text := empty_text;

  last_yylength: integer;

  procedure linenum;
  procedure myecho;

  function yylex return token;

end psdl_lex;
package body psdl_lex is

procedure myecho is
begin
  text_io.put(List_File, psdl_lex_dfa.yytext);
end myecho;

procedure linenum is
begin
  text_io.put(List_File, integer'image(lines) & ":");
  lines := lines + 1;
end linenum;

function YYLex return Token is
  subtype short is integer range -32768..32767;
  yy_act : integer;
  yy_c : short;

  -- returned upon end-of-file
  YY_END_TOK : constant integer := 0;
  YY_END_OF_BUFFER : constant := 85;
  subtype yy_state_type is integer;
  yy_current_state : yy_state_type;
  INITIAL : constant := 0;
  yy_accept : constant array(0..619) of short :=
( 0,
  0, 0, 85, 84, 83, 82, 84, 59, 60, 61,
  57, 55, 65, 56, 66, 58, 79, 64, 69, 54,
  68, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 62, 63, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 20, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
  77, 77, 77, 77, 77, 77, 77, 77, 77, 77,

306
yy_ec : constant array CHARACTER'FIRST..CHARACTER'LAST of short :=
{ 0,
  1, 1, 1, 1, 1, 1, 1, 1, 2, 3,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  7, 8, 9, 10, 10, 11, 11, 12, 13, 14,
  14, 14, 14, 14, 14, 14, 14, 15, 16, 16,
  17, 18, 19, 20, 20, 21, 21, 22, 22, 23,
  23, 23, 23, 24, 24, 25, 25, 26, 26, 27,
  27, 28, 28, 29, 29, 30, 30, 31, 31, 32,
  32, 32, 33, 33, 34, 34, 35, 35, 36, 36,
  37, 37, 38, 38, 39, 39, 40, 40, 41, 41,
  42, 42, 42, 42, 43, 43, 43, 44, 44, 45,
  45, 46, 46, 47, 47, 48, 48, 49, 49, 50,

yy_meta : constant array(0..74) of short :=
( 0,
  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
  1,  1,  1,  2,  1,  1,  1,  1,  2,  2,  2,  2,
  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
  2,  2,  2,  1,  1,  1,  2,  2,  2,  2,  2,  2,
  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
  3,  1,  1,  1,
 ) ;

yy_base : constant array(0..622) of short :=
( 0,
  0,  0,  725,  726,  726,  726,  726,  71,  726,  726,  726,
  716,  726,  726,  705,  726,  705,  64,  726,  704,  726,
  703,  57,  676,  61,  62,  60,  64,  61,  685,  64,
  0,  694,  71,  683,  67,  692,  691,  77,  78,  690,
  685,  678,  726,  726,  69,  640,  62,  73,  68,  76,
  62,  649,  74,  657,  87,  647,  78,  655,  654,  84,
  85,  653,  648,  642,  628,  726,  683,  108,  726, 125,
  726,  726,  726,  685,  138,  726,  726,  0,  661,  678,
  674,  668,  647,  84,  663,  660,  653,  653,  664,  666,
  133,  657,  654,  653,  665,  644,  0,  648,  109,  638,
  638,  136,  657,  0,  640,  654,  0,  638,  639, 127,
  653,  650,  127,  641,  132,  637,  634,  631,  632,  603,
  619,  615,  609,  159,  606,  603,  596,  596,  606,  608,
  123,  600,  597,  596,  607,  587,  591,  114,  581,  581,
  124,  599,  580,  596,  581,  582,  118,  595,  592,  130,
  584,  123,  580,  577,  574,  575,  564,  726,  622,  0,
  0,  0,  602,  177,  604,  148,  614,  611,  608,  0,
  607,  608,  0,  591,  600,  603,  591,  588,  593,  584,
  582,  579,  592,  582,  587,  160,  0,  0,  580,  581,
  587,  0,  168,  580,  591,  156,  577,  587,  586,  583,
  584,  563,  567,  578,  0,  543,  195,  545,  136,  554,
  551,  548,  547,  548,  532,  540,  543,  532,  529,  534,
  525,  523,  520,  532,  523,  528,  140,  521,  522,  527,
  154,  521,  531,  144,  518,  527,  526,  523,  524,  523,
  508,  518,  540,  540,  546,  535,  540,  528,  529,  0,
  528,  0,  529,  532,  538,  523,  523,  532,  520,  533,
  528,  516,  520,  521,  518,  523,  518,  510,  526,  507,
  512,  506,  510,  514,  502,  516,  201,  519,  501,
  511,  0,  0,  512,  507,  475,  475,  480,  470,  474,
  463,  464,  463,  464,  458,  472,  458,  458,  466,  455,
  467,  462,  451,  455,  456,  453,  457,  453,  445,  462,
  442,  447,  441,  445,  445,  448,  437,  450,  208,  453,

308
yy_def : constant array(0..622) of short :=
    (0,
    619,  1,  619,  619,  619,  619,  620,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  620,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    619,  619,  619,  619,  619,  619,  619,  619,  619,  619,
    621,  621,  621,  621,  621,  621,  621,  621,  621,  621,
    621,  621,  621,  621,  621,  621,  621,  621,  621,  621,
    621,  621,  621,  621,  621,  621,  621,  621,  621,  621,
    621,  621,  621,  621,  621,  621,  621,  621,  621,  621,
yy_nxt : constant array(0..800) of short :=

310
392, 391, 390, 389, 388, 387, 386, 385, 384, 383,
382, 381, 380, 379, 378, 377, 376, 375, 374,
373, 372, 371, 370, 369, 368, 367, 366, 365,
364, 363, 362, 361, 360, 359, 358, 357, 356,
355, 354, 353, 352, 351, 350, 349, 348, 347, 346,
345, 344, 343, 342, 341, 340, 339, 338, 337, 336,
335, 334, 333, 332, 331, 330, 329, 328, 327, 326,
325, 324, 323, 283, 282, 281, 280, 279, 278,
277, 276, 275, 274, 273, 272, 271, 270, 269, 268,
267, 266, 265, 264, 263, 262, 261, 260, 259, 258,
257, 256, 255, 254, 253, 252, 251, 250, 249, 248,
247, 246, 245, 244, 243, 242, 241, 240, 239, 238,
237, 236, 235, 234, 233, 232, 231, 230, 229, 228,
227, 226, 225, 224, 223, 222, 221, 220, 219, 218,
217, 216, 215, 214, 213, 212, 211, 210, 209, 208,
207, 206, 205, 204, 203, 202, 199, 198, 197, 196,
195, 194, 193, 192, 191, 190, 189, 188, 187, 186,
185, 184, 183, 182, 181, 180, 179, 178, 177, 176,
175, 174, 173, 172, 171, 170, 169, 168, 167, 166,
165, 164, 163, 162, 161, 160, 159, 158, 157, 156,
155, 154, 153, 152, 151, 150, 149, 148, 147, 146,
145, 144, 143, 142, 141, 140, 139, 138, 137, 136,
135, 134, 133, 132, 131, 130, 129, 128, 127, 126,
125, 124, 123, 122, 121, 120, 119, 118, 117, 116,
115, 114, 113, 112, 111, 110, 109, 108, 107, 106,
105, 104, 103, 102, 101, 100, 99, 98, 97, 96,
95, 94, 93, 92, 91, 90, 89, 88, 87, 86,
85, 84, 83, 82, 81, 80, 79, 78, 77, 76,
75, 74, 73, 72, 71, 70, 69, 68, 67, 66,
65, 64, 63, 62, 61, 60, 59, 58, 57, 56,
55, 54, 53, 52, 51, 50, 49, 48, 47, 46,
45, 44, 43, 42, 41, 40, 39, 38, 37, 36,
35, 34, 33, 32, 31, 30, 29, 28, 27, 26,
25, 24, 23, 22, 21, 20, 19, 18, 17, 16,
15, 14, 13, 12, 11, 10, 9, 8, 7, 6,
5, 4, 3, 2, 1, 0,

yy_chk : constant array(0..800) of short :=
{
0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
procedure ECHO is
begin
   text_io.put( yytext );
end ECHO;

-- enter a start condition.
-- Using procedure requires a () after the ENTER, but makes everything
-- much neater.

procedure ENTER( state : integer ) is
begin
   yy_start := 1 + 2 * state;
end ENTER;

-- action number for EOF rule of a given start state
function YY_STATE_EOF(state : integer) return integer is
begin
   return YY_END_OF_BUFFER + state + 1;
end YY_STATE_EOF;

-- return all but the first 'n' matched characters back to the input stream
procedure yyless(n : integer) is
begin
   yy_ch_buf(yy_cp) := yy_hold_char; -- undo effects of setting up yytext
   yy_cp := yy_bp + n;
   yy_c_buf_p := yy_cp;
   YY_DO_BEFORE_ACTION; -- set up yytext again
end yyless;

-- redefine this if you have something you want each time.
procedure YY_USER_ACTION is
begin
   null;
end;
function yy_get_previous_state return yy_state_type is
  yy_current_state : yy_state_type;
  yy_c : short;
begin
  yy_current_state := yy_start;
  for yy_cp in yytext_ptr..yy_c_buf_p - 1 loop
    yy_c := yy_sc(yy_ch_buf(yy_cp));
    if ( yy_accept(yy_current_state) /= 0 ) then
      yy_last_accepting_state := yy_current_state;
      yy_last_accepting_cpos := yy_cp;
    end if;
  end loop;
  while ( yy_chk(yy_base(yy_current_state) + yy_c) /= yy_current_state ) loop
    yy_current_state := yy_def(yy_current_state);
    if ( yy_current_state >= 620 ) then
      yy_c := yy_meta(yy_c);
    end if;
  end loop;
  yy_current_state := yy_nxt(yy_base(yy_current_state) + yy_c);
end loop;
return yy_current_state;
end yygetpreviousstate;

procedure yyrestart( input-file : file_type ) is
begin
  set_input(input_file);
  yy_init := true;
end yyrestart;

begin -- of YYLex
<<new_file>>
  -- this is where we enter upon encountering an end-of-file and
  -- yywrap() indicating that we should continue processing
  if ( yy_init ) then
    if ( yy_start = 0 ) then
      yy_start := 1; -- first start state
    end if;
    -- we put in the '\n' and start reading from [1] so that an
    -- initial match-at-newline will be true.
    yy_ch_buf(0) := ASCII.LF;
    yy_n_chars := 1;
    -- we always need two end-of-buffer characters. The first cause:
    -- a transition to the end-of-buffer state. The second causes
    -- a jam in that state.
    yy_ch_buf(yy_n_chars) := YY_END_OF_BUFFER_CHAR;
    yy_ch_buf(yy_n_chars + 1) := YY_END_OF_BUFFER_CHAR;
yy_eof_hasbeen_seen := false;

yytext_ptr := 1;
yy_c_buf_p := yytext_ptr;
yy_hold_char := yy_ch_buf(yy_c_buf_p);
yy_init := false;
end if; -- yy_init

loop -- loops until end-of-file is reached
    yy_cp := yy_c_buf_p;

-- support of yytext
yy_ch_buf(yy_cp) := yy_hold_char;

-- yy_bp points to the position in yy_ch_buf of the start of the
-- current run.

yy_bp := yy_cp;

yy_current_state := yy_start;
loop

    yy_c := yy_ec(yy_ch_buf(yy_cp));
    if ( yy_accept(yy_current_state) /= 0 ) then
        yy_last_accepting_state := yy_current_state;
        yy_last_accepting_cpos := yy_cp;
    end if;
    while ( yy_chk(yy_base(yy_current_state) + yy_c) /= yy_current_state ) loop
        yy_current_state := yy_def(yy_current_state);
        if ( yy_current_state >= 620 ) then
            yy_c := yy_meta(yy_c);
        end if;
    end loop;
    yy_current_state := yy_nxt(yy_base(yy_current_state) + yy_c);
    yy_cp := yy_cp + 1;
if ( yy_current_state = 619 ) then
    exit;
end if;
end loop;

yy_cp := yy_last_accepting_cpos;

yy_current_state := yy_last_accepting_state;

<<next_action>>
    yy_act := yy_accept(yy_current_state);
    YY_DO_BEFORE_ACTION;
    YY_USER_ACTION;

    if aflx_debug then -- output acceptance info. for (-d) debug mode
        text_11.put( Standard_Error, "-accepting rule ");
        text_11.put( Standard_Error, INTEGER'IMAGE(yy_act) );
        text_11.put_line( Standard_Error, "(" & yytext & ")");
    end if;

<<do_action>> -- this label is used only to access EOF actions
    case yy_act is
    when 0 => -- must backtrack
        -- undo the effects of YY_DO_BEFORE_ACTION
yy_ch_buf(yy_cp) := yy_hold_char;
yy_cp := yy_last_accepting_cpos;
yy_current_state := yy_last_accepting_state;
goto next_action;

when 1 =>
---$ line 66 "psdl_lex.l"
MYECHO; return (ADA_TOKEN);

when 2 =>
---$ line 67 "psdl_lex.l"
MYECHO; return (AXIOMS_TOKEN);

when 3 =>
---$ line 68 "psdl_lex.l"
MYECHO; return (BY_ALL_TOKEN);

when 4 =>
---$ line 69 "psdl_lex.l"
MYECHO; return (BY_REQ_TOKEN);

when 5 =>
---$ line 71 "psdl_lex.l"
MYECHO; return (BY_SOME_TOKEN);

when 6 =>
---$ line 72 "psdl_lex.l"
MYECHO; return (CONTROL_TOKEN);

when 7 =>
---$ line 73 "psdl_lex.l"
MYECHO; return (CONSTRAINTS_TOKEN);

when 8 =>
---$ line 74 "psdl_lex.l"
MYECHO; return (DATA_TOKEN);

when 9 =>
---$ line 75 "psdl_lex.l"
MYECHO; return (STREAM_TOKEN);

when 10 =>
---$ line 76 "psdl_lex.l"
MYECHO; return (DESCRIPTION_TOKEN);

when 11 =>
---$ line 77 "psdl_lex.l"
MYECHO; return (EDGE_TOKEN);

when 12 =>
---$ line 78 "psdl_lex.l"
MYECHO; return (END_TOKEN);
when 13 =>
  --$ line 79 "psdl_lex.l"
  MYECHO; return (EXCEPTION_TOKEN);

when 14 =>
  --$ line 80 "psdl_lex.l"
  MYECHO; return (EXCEPTION_TOKEN);

when 15 =>
  --$ line 81 "psdl_lex.l"
  MYECHO; return (FINISH_TOKEN);

when 16 =>
  --$ line 82 "psdl_lex.l"
  MYECHO; return (WITHIN_TOKEN);

when 17 =>
  --$ line 83 "psdl_lex.l"
  MYECHO; return (GENERIC_TOKEN);

when 18 =>
  --$ line 84 "psdl_lex.l"
  MYECHO; return (GRAPH_TOKEN);

when 19 =>
  --$ line 85 "psdl_lex.l"
  MYECHO; return (HOURS_TOKEN);

when 20 =>
  --$ line 86 "psdl_lex.l"
  MYECHO; return (IF_TOKEN);

when 21 =>
  --$ line 87 "psdl_lex.l"
  MYECHO; return (IMPLEMENTATION_TOKEN);

when 22 =>
  --$ line 88 "psdl_lex.l"
  MYECHO; return (INITIALLY_TOKEN);

when 23 =>
  --$ line 89 "psdl_lex.l"
  MYECHO; return (INPUT_TOKEN);

when 24 =>
  --$ line 90 "psdl_lex.l"
  MYECHO; return (KEYWORDS_TOKEN);

when 25 =>
  --$ line 91 "psdl_lex.l"
  MYECHO; return (MAXIMUM_TOKEN);

when 26 =>
  --$ line 92 "psdl_lex.l"
  MYECHO; return (EXECUTION_TOKEN);
when 27 =>
when 28 =>
when 29 =>
when 30 =>
when 31 =>
when 32 =>
when 33 =>
when 34 =>
when 35 =>
when 36 =>
when 37 =>
when 38 =>
when 39 =>
when 40 =>
MYECHO; return (START_TOKEN);

when 41 =>
   --# line 107 "psdl_lex.l"
   MYECHO; return (STATES_TOKEN);

when 42 =>
   --# line 108 "psdl_lex.l"
   MYECHO; return (STOP_TOKEN);

when 43 =>
   --# line 109 "psdl_lex.l"
   MYECHO; return (TIMER_TOKEN);

when 44 =>
   --# line 110 "psdl_lex.l"
   MYECHO; return (TRIGGERED_TOKEN);

when 45 =>
   --# line 111 "psdl_lex.l"
   MYECHO; return (TYPE_TOKEN);

when 46 =>
   --# line 112 "psdl_lex.l"
   MYECHO; return (VERTEX_TOKEN);

when 47 =>
   --# line 114 "psdl_lex.l"
   MYECHO; return (AND_TOKEN);

when 48 =>
   --# line 115 "psdl_lex.l"
   MYECHO; return (OR_TOKEN);

when 49 =>
   --# line 116 "psdl_lex.l"
   MYECHO; return (XOR_TOKEN);

when 50 =>
   --# line 117 "psdl_lex.l"
   MYECHO; return (GREATER_THAN_OR_EQUAL);

when 51 =>
   --# line 118 "psdl_lex.l"
   MYECHO; return (LESS_THAN_OR_EQUAL);

when 52 =>
   --# line 119 "psdl_lex.l"
   MYECHO; return (INEQUALITY);

when 53 =>
   --# line 120 "psdl_lex.l"
   MYECHO; return (ARROW);

when 54 =>
--$ line 121 "psdl_lex.l"
MYECHO; return ('=');

when 55 =>
--$ line 122 "psdl_lex.l"
MYECHO; return ('+');

when 56 =>
--$ line 123 "psdl_lex.l"
MYECHO; return ('-');

when 57 =>
--$ line 124 "psdl_lex.l"
MYECHO; return ('*');

when 58 =>
--$ line 125 "psdl_lex.l"
MYECHO; return ('/');

when 59 =>
--$ line 126 "psdl_lex.l"
MYECHO; return ('&');

when 60 =>
--$ line 127 "psdl_lex.l"
MYECHO; return ('(');

when 61 =>
--$ line 128 "psdl_lex.l"
MYECHO; return (')');

when 62 =>
--$ line 129 "psdl_lex.l"
MYECHO; return ('[');

when 63 =>
--$ line 130 "psdl_lex.l"
MYECHO; return (']');

when 64 =>
--$ line 131 "psdl_lex.l"
MYECHO; return (':\');

when 65 =>
--$ line 132 "psdl_lex.l"
MYECHO; return (',');

when 66 =>
--$ line 133 "psdl_lex.l"
MYECHO; return ('.');

when 67 =>
--$ line 134 "psdl_lex.l"
MYECHO; return ('|');
when 68 =>
    -- line 135 "psdl_lex.1"
    MYECHO; return ('>');

when 69 =>
    -- line 136 "psdl_lex.1"
    MYECHO; return ('<');

when 70 =>
    -- line 137 "psdl_lex.1"
    MYECHO; return (MOD_TOKEN);

when 71 =>
    -- line 138 "psdl_lex.1"
    MYECHO; return (REM_TOKEN);

when 72 =>
    -- line 139 "psdl_lex.1"
    MYECHO; return (EXP_TOKEN);

when 73 =>
    -- line 140 "psdl_lex.1"
    MYECHO; return (ABS_TOKEN);

when 74 =>
    -- line 141 "psdl_lex.1"
    MYECHO; return (NOT_TOKEN);

when 75 =>
    -- line 142 "psdl_lex.1"
    MYECHO; return (TRUE);

when 76 =>
    -- line 143 "psdl_lex.1"
    MYECHO; return (FALSE);

when 77 =>
    -- line 145 "psdl_lex.1"
    MYECHO;
    the_prev_id_token := the_id_token;
    the_id_token := to_a(psdl_lex_dfa.yytext);
    return (IDENTIFIER);

when 78 =>
    -- line 152 "psdl_lex.1"
    MYECHO;
    the_string_token := to_a(psdl_lex_dfa.yytext);
    return (STRING_LITERAL);

when 79 =>
    -- line 158 "psdl_lex.1"
MYECO;
the_integer_token := to_a(psdl_lex_dfa.yytext);
return (INTEGER_LITERAL);

when 80 =>
--$ line 164 "psdl_lex.l"

MYECO;
the_real_token := to_a(psdl_lex_dfa.yytext);
return (REAL_LITERAL);

when 81 =>
--$ line 170 "psdl_lex.l"

MYECO;
the_text_token := to_a(psdl_lex_dfa.yytext);
return (TEXT_TOKEN);

when 82 =>
--$ line 176 "psdl_lex.l"

MYECO; linenum;

when 83 =>
--$ line 177 "psdl_lex.l"
MYECO; null; -- ignore spaces and tabs

when 84 =>
--$ line 180 "psdl_lex.l"
raise AFLEX_SCANNER_JAMMED;
when YY_END_OF_BUFFER + INITIAL + 1 =>
  return End_Of_Input;
  when YY_END_OF_BUFFER =>
    -- undo the effects of YY_DO_BEFORE_ACTION
    yy_ch_buf(yy_cp) := yy_hold_char;
    yytext_ptr := yy_bp;
    case yy_get_next_buffer is
      when EOB_ACT_END_OF_FILE =>
        begin
        if ( yywrap ) then
          -- note: because we’ve taken care in
          -- yy_get_next_buffer() to have set up yytext,
          -- we can now set up yy_c_buf_p so that if some
          -- total hoser (like aflex itself) wants
          -- to call the scanner after we return the
          -- End_Of_Input, it’ll still work - another
          -- End_Of_Input will get returned.
          yy_c_buf_p := yytext_ptr;

523
yy_act := YY_STATE_EOF((yy_rstart - 1) / 2);

goto do_action;
else
  -- start processing a new file
  yy_init := true;
  goto new_file;
end if;

when EOB_ACT_RESTART_SCAN =>
  yy_c_buf_p := yytext_ptr;
  yy_hold_char := yy_ch_buf(yy_c_buf_p);
when EOB_ACT_LAST_MATCH =>
  yy_c_buf_p := yy_n_chars;
  yy_current_state := yy_get_previous_state;

  yy_cp := yy_c_buf_p;
  yy_bp := yytext_ptr;
  goto next_action;
when others => null;
end case; -- case yy_get_next_buffer()
when others =>
  text_io.put( "action $ " );
  text_io.put( INTEGER'IMAGE(yy_act) );
  text_io.new_line;
  raise AFLEX_INTERNAL_ERROR;
end case; -- case (yy_act)
end YYLex;

end psdl_lex;
APPENDIX S. PACKAGE PSDL_LEX_IO

with psdl_lex_dfa; use psdl_lex_dfa;
with text_i0; use text_i0;

package psdl_lex_io is
NULL_IN_INPUT : exception;
AFLEX_INTERNAL_ERROR : exception;
UNEXPECTED_LAST_MATCH : exception;
PUSHBACK_OVERFLOW : exception;
AFLEX_SCANNER_JAMMED : exception;
type eob_action_type is ( EOB_ACT_RESTART_SCAN,
                        EOB_ACT_END_OF_FILE,
                        EOB_ACT_LAST_MATCH );
YY_END_OF_BUFFER_CHAR : constant character:= ASCII.NUL;
yy_n_chars : integer; -- number of characters read into yy_ch_buf
-- true when we've seen an EOF for the current input file
yy_eof_has_been_seen : boolean;

procedure YY_INPUT(buf: out unbounded_character_array;
result: out integer; max_size: in integer);
function yy_get_next_buffer return eob_action_type;
procedure yyunput(c : character; yy_bp: in out integer );
procedure unput(c : character);
procedure yywrap return boolean;
procedure Open_Input(fname : in String);
procedure Close_Input;
procedure Create_Output(fname : in String := "");
procedure Close_Output;
end psdl_lex_io;

package body psdl_lex_io is
-- gets input and stuffs it into 'buf'. number of characters read, or YY_NULL,
-- is returned in 'result'.

procedure YY_INPUT(buf: out unbounded_character_array;
result: out integer; max_size: in integer) is
  c : character;
i : integer := 1;
loc : integer := buf'first;
begin
  while ( i <= ma._size ) loop
    if (end_of_line) then -- Ada ate our newline, put it back on the end.
      buf(loc) := ASCII.LF;
      skip_line(1);
    else

325
get(buf(loc));
end it;

loc := loc + 1;
i := i + 1;
end loop;

result := 1 - 1;

exception
  when END_ERROR => result := 1 - 1;
-- when we hit EOF we need to set yy_eof_has been seen
  yy_eof_has been seen := true;
end YYINPUT;

-- yy_get_next_buffer - try to read in new buffer
-- returns a code representing an action
-- EOB_ACT_LAST_MATCH -
-- EOB_ACT_RESTART_SCAN - restart the scanner
-- EOB_ACT_END_OF_FILE - end of file

function yy_get_next_buffer return eob_action_type is
  dest : integer := 0;
  source : integer := yytext_ptr - 1; -- copy prev. char, too
  number_to_move : integer;
  ret_val : eob_action_type;
  num_to_read : integer;

  begin
    if yy_c_buf_p > yy_n_chars + 1 then
      raise NULL_IN_INPUT;
    end if;

    -- try to read more data

    -- first move last chars to start of buffer
    number_to_move := yy_c_buf_p - yytext_ptr;

    for i in 0..number_to_move - 1 loop
      yy_ch_buf(dest) := yy_ch_buf(source);
      dest := dest + 1;
      source := source + 1;
    end loop;

    if yy_eof_has been seen then
      -- don't do the read, it's not guaranteed to return an EOF,
      -- just force an EOF

      yy_n_chars := 0;
    else
      num_to_read := YY_BUF_SIZE - number_to_move - 1;
      if num_to_read > YY_READ_BUF_SIZE then
        num_to_read := YY_READ_BUF_SIZE;
      end if;

      -- yy_get_next_buffer - try to read in new buffer
      -- returns a code representing an action
      -- EOB_ACT_LAST_MATCH -
      -- EOB_ACT_RESTART_SCAN - restart the scanner
      -- EOB_ACT_END_OF_FILE - end of file

      function yy_get_next_buffer return eob_action_type is
        dest : integer := 0;
        source : integer := yytext_ptr - 1; -- copy prev. char, too
        number_to_move : integer;
        ret_val : eob_action_type;
        num_to_read : integer;

        begin
          if yy_c_buf_p > yy_n_chars + 1 then
            raise NULL_IN_INPUT;
          end if;

          -- try to read more data

          -- first move last chars to start of buffer
          number_to_move := yy_c_buf_p - yytext_ptr;

          for i in 0..number_to_move - 1 loop
            yy_ch_buf(dest) := yy_ch_buf(source);
            dest := dest + 1;
            source := source + 1;
          end loop;

          if yy_eof_has been seen then
            -- don't do the read, it's not guaranteed to return an EOF,
            -- just force an EOF

            yy_n_chars := 0;
          else
            num_to_read := YY_BUF_SIZE - number_to_move - 1;
            if num_to_read > YY_READ_BUF_SIZE then
              num_to_read := YY_READ_BUF_SIZE;
            end if;

            -- yy_get_next_buffer - try to read in new buffer
            -- returns a code representing an action
            -- EOB_ACT_LAST_MATCH -
            -- EOB_ACT_RESTART_SCAN - restart the scanner
            -- EOB_ACT_END_OF_FILE - end of file

            function yy_get_next_buffer return eob_action_type is
              dest : integer := 0;
              source : integer := yytext_ptr - 1; -- copy prev. char, too
              number_to_move : integer;
              ret_val : eob_action_type;
              num_to_read : integer;

              begin
                if yy_c_buf_p > yy_n_chars + 1 then
                  raise NULL_IN_INPUT;
                end if;

                -- try to read more data

                -- first move last chars to start of buffer
                number_to_move := yy_c_buf_p - yytext_ptr;

                for i in 0..number_to_move - 1 loop
                  yy_ch_buf(dest) := yy_ch_buf(source);
                  dest := dest + 1;
                  source := source + 1;
                end loop;

                if yy_eof_has been seen then
                  -- don't do the read, it's not guaranteed to return an EOF,
                  -- just force an EOF

                  yy_n_chars := 0;
                else
                  num_to_read := YY_BUF_SIZE - number_to_move - 1;
                  if num_to_read > YY_READ_BUF_SIZE then
                    num_to_read := YY_READ_BUF_SIZE;
                  end if;

                  -- yy_get_next_buffer - try to read in new buffer
                  -- returns a code representing an action
                  -- EOB_ACT_LAST_MATCH -
                  -- EOB_ACT_RESTART_SCAN - restart the scanner
                  -- EOB_ACT_END_OF_FILE - end of file

                  function yy_get_next_buffer return eob_action_type is
                    dest : integer := 0;
                    source : integer := yytext_ptr - 1; -- copy prev. char, too
                    number_to_move : integer;
                    ret_val : eob_action_type;
                    num_to_read : integer;

                    begin
                      if yy_c_buf_p > yy_n_chars + 1 then
                        raise NULL_IN_INPUT;
                      end if;

                      -- try to read more data

                      -- first move last chars to start of buffer
                      number_to_move := yy_c_buf_p - yytext_ptr;

                      for i in 0..number_to_move - 1 loop
                        yy_ch_buf(dest) := yy_ch_buf(source);
                        dest := dest + 1;
                        source := source + 1;
                      end loop;

                      if yy_eof_has been seen then
                        -- don't do the read, it's not guaranteed to return an EOF,
                        -- just force an EOF

                        yy_n_chars := 0;
                      else
                        num_to_read := YY_BUF_SIZE - number_to_move - 1;
                        if num_to_read > YY_READ_BUF_SIZE then
                          num_to_read := YY_READ_BUF_SIZE;
                        end if;

                        -- yy_get_next_buffer - try to read in new buffer
                        -- returns a code representing an action
                        -- EOB_ACT_LAST_MATCH -
                        -- EOB_ACT_RESTART_SCAN - restart the scanner
                        -- EOB_ACT_END_OF_FILE - end of file

                        function yy_get_next_buffer return eob_action_type is
                          dest : integer := 0;
                          source : integer := yytext_ptr - 1; -- copy prev. char, too
                          number_to_move : integer;
                          ret_val : eob_action_type;
                          num_to_read : integer;

                          begin
                            if yy_c_buf_p > yy_n_chars + 1 then
                              raise NULL_IN_INPUT;
                            end if;

                            -- try to read more data

                            -- first move last chars to start of buffer
                            number_to_move := yy_c_buf_p - yytext_ptr;

                            for i in 0..number_to_move - 1 loop
                              yy_ch_buf(dest) := yy_ch_buf(source);
                              dest := dest + 1;
                              source := source + 1;
                            end loop;

                            if yy_eof_has been seen then
                              -- don't do the read, it's not guaranteed to return an EOF,
                              -- just force an EOF

                              yy_n_chars := 0;
                            else
                              num_to_read := YY_BUF_SIZE - number_to_move - 1;
                              if num_to_read > YY_READ_BUF_SIZE then
                                num_to_read := YY_READ_BUF_SIZE;
                              end if;

                              -- yy_get_next_buffer - try to read in new buffer
                              -- returns a code representing an action
                              -- EOB_ACT_LAST_MATCH -
                              -- EOB_ACT_RESTART_SCAN - restart the scanner
                              -- EOB_ACT_END_OF_FILE - end of file

                              function yy_get_next_buffer return eob_action_type is
                                dest : integer := 0;
                                source : integer := yytext_ptr - 1; -- copy prev. char, too
                                number_to_move : integer;
                                ret_val : eob_action_type;
                                num_to_read : integer;

                                begin
                                  if yy_c_buf_p > yy_n_chars + 1 then
                                    raise NULL_IN_INPUT;
                                  end if;

                                  -- try to read more data

                                  -- first move last chars to start of buffer
                                  number_to_move := yy_c_buf_p - yytext_ptr;

                                  for i in 0..number_to_move - 1 loop
                                    yy_ch_buf(dest) := yy_ch_buf(source);
                                    dest := dest + 1;
                                    source := source + 1;
                                  end loop;

                                  if yy_eof_has been seen then
                                    -- don't do the read, it's not guaranteed to return an EOF,
                                    -- just force an EOF

                                    yy_n_chars := 0;
                                  else
                                    num_to_read := YY_BUF_SIZE - number_to_move - 1;
                                    if num_to_read > YY_READ_BUF_SIZE then
                                      num_to_read := YY_READ_BUF_SIZE;
                                    end if;

                                    -- yy_get_next_buffer - try to read in new buffer
                                    -- returns a code representing an action
                                    -- EOB_ACT_LAST_MATCH -
                                    -- EOB_ACT_RESTART_SCAN - restart the scanner
                                    -- EOB_ACT_END_OF_FILE - end of file

                                    function yy_get_next_buffer return eob_action_type is
                                      dest : integer := 0;
                                      source : integer := yytext_ptr - 1; -- copy prev. char, too
                                      number_to_move : integer;
                                      ret_val : eob_action_type;
                                      num_to_read : integer;

                                      begin
                                        if yy_c_buf_p > yy_n_chars + 1 then
                                          raise NULL_IN_INPUT;
                                        end if;

                                        -- try to read more data

                                        -- first move last chars to start of buffer
                                        number_to_move := yy_c_buf_p - yytext_ptr;

                                        for i in 0..number_to_move - 1 loop
                                          yy_ch_buf(dest) := yy_ch_buf(source);
                                          dest := dest + 1;
                                          source := source + 1;
                                        end loop;

                                        if yy_eof_has been seen then
                                          -- don't do the read, it's not guaranteed to return an EOF,
                                          -- just force an EOF

                                          yy_n_chars := 0;
                                        else
                                          num_to_read := YY_BUF_SIZE - number_to_move - 1;
                                          if num_to_read > YY_READ_BUF_SIZE then
                                            num_to_read := YY_READ_BUF_SIZE;
                                          end if;

                                          -- yy_get_next_buffer - try to read in new buffer
                                          -- returns a code representing an action
                                          -- EOB_ACT_LAST_MATCH -
                                          -- EOB_ACT_RESTART_SCAN - restart the scanner
                                          -- EOB_ACT_END_OF_FILE - end of file

                                          function yy_get_next_buffer return eob_action_type is
                                            dest : integer := 0;
                                            source : integer := yytext_ptr - 1; -- copy prev. char, too
                                            number_to_move : integer;
                                            ret_val : eob_action_type;
                                            num_to_read : integer;

                                            begin
                                              if yy_c_buf_p > yy_n_chars + 1 then
                                                raise NULL_IN_INPUT;
                                              end if;

                                              -- try to read more data

                                              -- first move last chars to start of buffer
                                              number_to_move := yy_c_buf_p - yytext_ptr;

                                              for i in 0..number_to_move - 1 loop
                                                yy_ch_buf(dest) := yy_ch_buf(source);
                                                dest := dest + 1;
                                                source := source + 1;
                                              end loop;

                                              if yy_eof_has been seen then
                                                -- don't do the read, it's not guaranteed to return an EOF,
                                                -- just force an EOF

                                                yy_n_chars := 0;
                                              else
                                                num_to_read := YY_BUF_SIZE - number_to_move - 1;
                                                if num_to_read > YY_READ_BUF_SIZE then
                                                  num_to_read := YY_READ_BUF_SIZE;
                                                end if;
-- read in more data
YY_INPUT( yy_ch_buf(number_to_move..yy_ch_buf'last),
          yy_n_chars, num_to_read );
end if;
if ( yy_n_chars = 0 ) then
if ( number_to_move = 1 ) then
   ret_val := EOB_ACT_END_OF_FILE;
else
   ret_val := EOB_ACT_LAST_MATCH;
end if;
else
yy_eof_hasbeen_seen := true;
else
ret_val := EOB_ACT_RESTART_SCAN;
end if;

yy_n_chars := yy_n_chars + number_to_move;
yy_ch_buf(yy_n_chars) := YY_END_OF_BUFFER_CHAR;
yy_ch_buf(yy_n_chars + 1) := YY_END_OF_BUFFER_CHAR;

-- yytext begins at the second character in
-- yy_ch_buf; the first character is the one which
-- preceded it before reading in the latest buffer;
-- it needs to be kept around in case it's a
-- newline, so yy_get_previous_state() will have
-- with '^' rules active

yytext_ptr := 1;
return ret_val;
end yygetnext_buffer;

procedure yyunput( c : character; yy_bp: in out integer ) is
  number_to_move : integer;
  dest : integer;
  source : integer;
  tmp_yy_cp : integer;
beginn
  tmp_yy_cp := yy_c_buf_p;
  yy_ch_buf(tmp_yy_cp) := yy_hold_char; -- undo effects of setting up yytext

  if ( tmp_yy_cp < 2 ) then
    -- need to shift things up to make room
    number_to_move := yy_n_chars + 2; -- +2 for EOB chars
    dest := YY_BUF_SIZE + 2;
    source := number_to_move;

    while ( source > 0 ) loop
      dest := dest - 1;
      source := source - 1;
      yy_ch_buf(dest) := yy_ch_buf(source);
    end loop;
  end if;
  yy_bp := yy_bp + dest - source;
end yyunput;
yy_n_chars := YY_BUF_SIZE;

if ( tmp_yy_cp < 2 ) then
    raise PUSHBACK_OVERFLOW;
end if;
end if;

if ( tmp_yy_cp > yy_bp and yy_ch_buf(tmp_yy_cp-1) = ASCII.LF ) then
    yy_ch_buf(tmp_yy_cp-2) := ASCII.LF;
end if;

tmp_yy_cp := tmp_yy_cp - 1;
yy_ch_buf(tmp_yy_cp) := c;

-- Note: this code is the text of YY_D0_BEFORE_ACTION, only
-- here we get different yy_cp and yy_bp's
yytextyptr := yybp;
yyhold_char := yy_ch_buf(tmpyycp);
yy_ch_buf(tmpyycp) := ASCII.NUL;
end yyunput;

procedure unput(c : character) is
begin
    yyunput( c, yybp );
end unput;

function input return character is
    c : character;
    yy_cp : integer := yy_c_buf_p;
begin
    yy_ch_buf(yy_cp) := yy_hold_char;

    if ( yy_ch_buf(yy_c_buf_p) = YY_END_OF_BUFFER_CHAR ) then
        -- need more input
        yytext_ptr := yy_c_buf_p;
        yy_c_buf_p := yy_c_buf_p + 1;
    end if;

    case yy_get_next_buffer is
        -- this code, unfortunately, is somewhat redundant with
        -- that above

            when EOB_ACT_END_OF_FILE =>
                if ( yywrap ) then
                    yy_c_buf_p := yytext_ptr;
                    return ASCII.NUL;
                end if;

                yy_ch_buf(0) := ASCII.LF;
                yy_n_chars := 1;
                yy_ch_buf(yy_n_chars) := YY_END_OF_BUFFER_CHAR;
                yy_ch_buf(yy_n_chars + 1) := YY_END_OF_BUFFER_CHAR;
                yy_eof_has_been_seen := false;
                yy_c_buf_p := 1;
                yytext_ptr := yy_c_buf_p;

328
yy_holdchar := yy_ch_buf(yy_c_buf_p);

return (input);
when EOB_ACT_RESTART_SCAN =>
  yy_c_buf_p := yytext_ptr;
when EOB_ACT_LAST_MATCH =>
  raise UNEXPECTED_LAST_MATCH;
when others => null;
end case;
end if;

c := yy_ch_buf(yy_c_buf_p);
yy_c_buf_p := yy_c_buf_p + 1;
yy_holdchar := yy_ch_buf(yy_c_buf_p);

return c;
end input;

procedure output(c : character) is
begin
  text_io.put(c);
end output;

-- default yywrap function - always treat EOF as an EOF
function yywrap return boolean is
begin
  return true;
end yywrap;

procedure Open_Input(fname : in String) is
f : file_type;
begin
  yy_init := true;
  open(f, in_file, fname);
  set_input(f);
end Open_Input;

procedure Create_Output(fname : in String := ") is
f : file_type;
begin
  if (fname /= "") then
    create(f, out_file, fname);
    set_output(f);
  end if;
end Create_Output;

procedure Close_Input is
begin
  null;
end Close_Input;

procedure Close_Output is
begin
  null;
end Close_Output;

329
end Close_Output;
end psdl_lex_i0;
package psdl_lex_dfa is
aflexdebug : boolean := false;
yytext_ptr : integer; -- points to start of yytext in buffer

-- yy_ch_buf has to be 2 characters longer than YY_BUF_SIZE because we need
-- to put in 2 end-of-buffer characters (this is explained where it is
-- done) at the end of yy_ch_buf
YY_READ_BUF_SIZE : constant integer := 8192;
YY_BUF_SIZE : constant integer := YY_READ_BUF_SIZE * 2; -- size of input buffer
type unbounded_character_array is array(integer range <>) of character;
subtype ch_buf_type is unbounded_character_array(0..YY_BUF_SIZE + 1);
yy_ch_buf : ch_buf_type;
yy_cp, yy_bp : integer;

-- yy_hold_char holds the character lost when yytext is formed
yy_hold_char : character;
yy_c_buf_p : integer; -- points to current character in buffer

function YYText return string;
function YYLength return integer;
procedure YYDO_BEFORE_ACTION;
-- These variables are needed between calls to YYLex.
yy_init : boolean := true; -- do we need to initialize YYLex?
yy_start : integer := 0; -- current start state number
subtype yy_state_type is integer;
yy_last_accepting_state : yy_state_type;
yy_last_accepting_cpos : integer;
end psdl_lex_dfa;

package body psdl_lex_dfa is
function YYText return string is
  i : integer;
  str_loc : integer := 1;
  buffer : string(1..1024);
  EMPTY_STRING : constant string := "";
begín
  -- find end of buffer
  i := yytext_ptr;
  while ( yy_ch_buf(i) /= ASCII.NUL ) loop
    buffer(str_loc) := yy_ch_buf(i);
    i := i + 1;
    str_loc := str_loc + 1;
  end loop;

  -- return yy_ch_buf(yytext_ptr.. i - 1);
  if (str_loc < 2) then
    return EMPTY_STRING;
  else
    return buffer(1..str_loc);
  end if;
end YYText;
return buffer(1..str_loc-1);
end if;

end;

-- returns the length of the matched text
function YYLength return integer is
begin
  return yy_cp - yy_bp;
end YYLength;

-- done after the current pattern has been matched and before the
-- corresponding action - sets up yytext
procedure YY_DO_BEFORE_ACTION is
begin
  yytext_ptr := yy_bp;
  yy_hold_char := yy_ch_buf(yy_cp);
  yy_ch_buf(yy_cp) := ASCII.NUL;
  yy_c_buf_p := yy_cp;
end YY_DO_BEFORE_ACTION;

end psdl_lex_dfa;
APPENDIX U. PACKAGE PARSER

with Text_Io, Psdl_Component_Pkg, Psdl_Concrete_Type_Pkg, Stack_Pkg, Psdl_Graph_Pkg, Generic_Sequence_Pkg, A_String;
use Psdl_Component_Pkg, Psdl_Concrete_Type_Pkg, Psdl_Graph_Pkg;

package Parser is

-- Global Variable Which Is A Map From Psdl_Component Names To Psdl
-- Component Definitions
The_Program : Psdl_Program; -- Implemented

-- Global Variable For A Psdl_Component (Type Or Operator)
The_Component : Psdl_Component; -- Implemented

-- Global Variable Which Points To The Psdl_Component (Type Or Operator)
The_Component_Ptr : Component_Ptr; -- Implemented

-- Global Variable Which Points To The Psdl Operator (Type Or Operator)
The_Op_Ptr : Op_Ptr; -- Implemented

-- used to construct the operation map
The_Operator : Operator;

-- Global Variable For An Atomic Type -- Implemented
The_Atomic_Type

333
: Atomic_Type;
-- Global Variable For An Atomic Operator
The_Atom_Operator : Atomic_Operator;

-- Global Variable For A Composite Psdl Type
The_Composite_Type : Composite_Type;

-- Global Variable For A Composite Psdl Type
The_Composite_Operator : Composite_Operator;

-- /* Global Variables For All Psdl Components: */

-- Global Variable Which Holds The Name Of The Component
The_Psdl_Name : Psdl_Id;

-- Global Variable Which Holds The Ada_Id Variable Of Component Record
The_Ada_Name : Ada_Id;

-- Global Variable Which Holds The Generic Parameters
The_Gen_Par : Type_Declaration;

-- used for psdl_type part (for not to mix with operation map)
The_Type_Gen_Par : Type_Declaration;

-- Global Variable Which Holds The Keywords
The_Kwds : Id_Set;

The_Description : Text;

The_Axioms : Text;

-- A Temporary Variable To Hold Output_Id To Construct Out_Guard Map
The_Output_Id : Output_Id;

-- A Temporary Variable To Hold Excep_Id To Construct Excep_Trigger Map
The_Exception
: Exception;

-- Global Variables For All PSDL Types:

-- Used For Creating All Types

The_Model
: Type_Declaration;

The_Operation_Map
: Operation_Map;

-- Used For Creating Composite Types

The_Data_Structure
: Type_Name;

-- Global Variables For All Operators:

The_Input
: Type_Declaration;

The_Output
: Type_Declaration;

The_State
: Type_Declaration;

The_Initial_Expression
: Init_Map;

The_Exceptions
: Id_Set;

The_Specified_Met
: Millisec;

-- Global Variables For Composite Operators:

The_Graph
: PSDL_Graph;

The_Streams
: Type_Declaration;

The_Timers
: Id_Set;

The_Trigger
: Trigger_Map;
The_Exec_Guard : Exec_Guard_Map;

The_Out_Guard : Out_Guard_Map;

The_Excep_Trigger : Excep_Trigger_Map;

The_Timer_Op : Timer_Op_Map;

The_Per : Timing_Map;

The_Fw : Timing_Map;

The_Map : Timing_Map;

The_Mrt : Timing_Map;

The_Impl_Desc
 : Text := Empty_Text;

-- Is Used For Storing The Operator Names In Control Constraints Part

The.Operator_Name
 : Psdl_Id;

-- A Place Holder To For Time Values

The_Time
 : Millisec;

-- True If The Psdl_Component Is An Atomic One

Is_Atomical_Type
 : Boolean;

Is_Atomical_Operator : Boolean;

-- Holds The Name Of The Edge (I.E Stream Name)

The_Edge_Name
 : Psdl_Id;
Procedure Bind_Program
( Name : In Psdl_Id;
  Component : In Component_Ptr;
  Program : In Out
          Psdl_Program )
Renames Bind;

Procedure Bind_Type_Decl_Map
( Key : In Psdl_Id;
  Result : In Type_Name;
  Map : In Out
        Type_Declaration )
Renames Type_Declaration_Pkg. Bind;

Procedure Bind_Operation
( Key : In Psdl_Id;
  Result : In Op_Ptr;
  Map : In Out Operation_Map )
Renames Bind;

Procedure Bind_Trigger
( Key : In Psdl_Id;
  Result : In Trigger_Record;
  Map : In Out Trigger_Map )
Renames Trigger_Map_Pkg.Bind;

Procedure Bind_Timing
( Key : In Psl_Id;
     Result : In Millisec;
     Map : In Out Timing_Map )
Renames Timing_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Out_Guard Map Is A Mapping From Output Stream Id’S To
-- .. Expression Strings

Procedure Bind_Out_Guard
( Key : In Output_Id;
     Result : In Expression;
     Map : In Out Out_Guard_Map )
Renames Out_Guard_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Init_Map Is A Mapping From Psl Id’S To ..
-- .. Expression Strings

Procedure Bind_Init_Map
( Key : In Psl_Id;
     Result : In Expression;
     Map : In Out Init_Map )
Renames Init_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Timer.Op_Map Is A Mapping From Psl Id’S To ..
-- .. Timer.Op_Set

Procedure Bind_Timer.Op
( Key : In Psl_Id;
     Result : In Timer.Op_Set;
     Map : In Out Timer.Op_Map )
Renames Timer.Op_Map_Pkg.Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Exception Trigger Map Is A Mapping From Psl Id’S To ..
-- .. Expression Strings

Procedure Bind_Excep_Trigger
( Key : In Excep_Id;
     Result : In Expression;
     Map : In Out Excep_Trigger_Map )
Renames Excep_Trigger_Map_Pkg.

Bind;

-- Renames The Procedure Bind In Generic Map Package
-- Exec_Guard Map Is A Mapping From Pdld Id'S To ..
-- .. Expression Strings

Procedure Bind_Exec_Guard
( Key : In Pdld_Id;
  Result : In Expression;
  Map : In Out Exec_Guard_Map
)
Renames Exec_Guard_Map_Pkg.Bind;

-- Implements A Temporary Storage For Type Declaration.

Package Type_Decl_Stack_Pkg Is
  New Stack_Pkg (Type_Declaration);
;
Use Type_Decl_Stack_Pkg;

Subtype Type_Decl_Stack Is
  Type_Decl_Stack_Pkg.Stack;

  -- A Stack Declaration And Initialization For Type_Declaration

The_Type_Decl_Stack
  : Type_Decl_Stack :=
  Type_Decl_Stack_Pkg.Create;

Package Id_Set_Stack_Pkg Is
  New Stack_Pkg (Id_Set);

Subtype Id_Set_Stack Is
  Id_Set_Stack_Pkg.Stack;

  -- A Stack Declaration And Initialization For Id

The_Id_Set_Stack
  : Id_Set_Stack :=
  Id_Set_Stack_Pkg.Create;

  -- Global Declaration For Type_Id_Set
The_Id_Set : Id_Set;

The_Id_Set_Size : Natural;

Package Expression_Stack_Pkg Is
   New Stack_Pkg (Expression);

Subtype Expression_Stack Is
   Expression_Stack_Pkg.Stack;

   -- A Stack Declaration And Initialization For Id

The_Expression(Stack
   : Expression_Stack :=
   Expression_Stack_Pkg.Create;

Package Exp_seq_Pkg Is
   New Generic_Sequence_Pkg (T =>
   Expression, Block_Size => 24
);

Subtype Exp_Seq Is
   Exp_Seq_Pkg.Sequence;

   -- returns an empty expression sequence
function Empty_Exp_Seq return Exp_Seq;

The_Exp_Seq :
   Exp_Seq;

The_Init_Exp_Seq : Exp_Seq; -- Used For Constructing Init_Map
Temp_Init_Exp_Seq : Exp_Seq;

package Init_Exp_Seq_Stack_Pkg is
   new Stack_Pkg (Exp_Seq);

   subtype Init_Exp_Seq_Stack is Init_Exp_Seq_Stack_Pkg.Stack;

The_Init_Exp_Seq_Stack :
   Init_Exp_Seq_Stack := Init_Exp_Seq_Stack_Pkg.Create;

Procedure Remove_Expr_From_Seq Is
   New Exp_Seq_Pkg.Generic_Remove(Eq => "=");

Package Id_Seq_Pkg Is

340
New Generic_Sequence_Pkg (T => Psdl_Id, Block_Size => 24);

Subtype Id_Seq Is
   Id_Seq_Pkg.Sequence;

The_Id_Seq : Id_Seq;

The_Init_Map_Id_Seq: Id_Seq; -- to hold the id's to construct init map
   -- these are the same id's used in state map.

-- Holds The Name Of The Types;

The_Type_Name : Type_Name;

   -- Used For The Type Decl Part Of Type_Name
The_Type_Name_Decl : Type_Declaration;

   -- A Temporary Type_DECL
Temp_Type_Decl : Type_Declaration;

   -- A Temporary Variable For Holding The Identifiers
The_String : Psdl_Id;

   -- A Temporary Variable For Trigger_Record
The_Trigger_Record : Trigger_Record;

   -- A Temp Variable For Holding The Value Of Timer_Op
The_Timer_Op_Record : Timer_Op;

The_Timer_Op_Set : Timer_Op_Set;

   -- A Temp Variable For Producing The Expression String
The_Expression_String
   : Expression := Expression(
       A_Strings.Empty);
A Temp Variable For Producing The Time String

The_Time_String
  : Expression := Expression(
      A_Strings.Empty);

Echo
  : Boolean := False;

Number_Of_Errors
  : Natural := 0;

Semantic_Error : Exception;

Procedure Yyparse;

procedure GET(Item : out PSDL_PROGRAM);

procedure GET(Input_File_N : in String;
  Output_File_N in String := ";
  Item out PSDL_PROGRAM);

end Parser;

-- this flag is set to true when optional_generic_param
-- rule is parsed, to overcome the problem when two
id's come after one another. See psdl_lex.l file

Type_Spec_Gen_Par : Boolean := FALSE;

function Empty_Exp_Seq

function Empty_Exp_Seq return Exp_Seq is
    S: Exp_Seq;
begin
    Exp_Seq_Pkg.Empty(S);
    return S;
end Empty_Exp_Seq;

procedure Yyerror

procedure Yyerror
( S : In String :=
    "Syntax Error" ) is
    Space : Integer;
begin
    Number_Of_Errors :=
        Number_Of_Errors + 1;
    Text_Io.New_Line;
    Text_Io.Put("Line" & Integer'
        Image(Lines - 1) & " :");
    Text_Io.Put_Line(Psdl_Lex_Dfa.
        Yytext);
    Space := Integer(Psdl_Lex_Dfa.
        Yytext'Length) + Integer'
        Image(Lines)'Length + 5;
    for I In 1 .. Space loop
        Put("-"); end loop;
    Put_line("^ " & S); end Yyerror;

function Convert_To_Digit

function Convert_To_Digit
( String_Digit : String )
Return Integer Is
    Multiplier
        : Integer := 1;
    Digit, Nat_Value
        : Integer := 0;
Begin -- Convert_To_Digit
For I In Reverse 1 ..
  String_Digit'Length Loop
  Case String_Digit(I) Is
    When '0' =>
      Digit := 0;
    When '1' =>
      Digit := 1;
    When '2' =>
      Digit := 2;
    When '3' =>
      Digit := 3;
    When '4' =>
      Digit := 4;
    When '5' =>
      Digit := 5;
    When '6' =>
      Digit := 6;
    When '7' =>
      Digit := 7;
    When '8' =>
      Digit := 8;
    When '9' =>
      Digit := 9;
    When Others =>
      Null;
  End Case;
  Nat_Value := Nat_Value + (Multiplier * Digit);
  Multiplier := Multiplier * 10;
End Loop;
Return Nat_Value;
end Convert_To_Digit;

------------------------------------------------------------------------
procedure GET
-- Reads the psdl source file, parses it and creates the PSDL ADT
-- input file is line numbered and saved into a file
-- input file name .lst in the current directory. So if
-- there is no write permission for that directory, exception
-- Use_Error is raised and program aborts. if the second argument
-- is passed psdl file resulted form PSDL ADT is written into a
-- file with that name.
------------------------------------------------------------------------
procedure GET(Input_File_N : in String;
  Output_File_N : in String := "");
  Item : out PSDL_PROGRAM ) is
begin
procedure GET

-- Reads the standard input, parses it and creates the
-- PSDL ADT. Input file is line numbered and saved into a
-- file input file name .lst in the current directory. So if
-- there is no write permission for that directory, exception
-- Use_Error is raised and program aborts.

procedure GET(Item : out PSDL_PROGRAM) is
begin
Text_Io.Create(Psdl_Lex.List_File, Out_File, "stdin.psdl.lst");
Psdl_Lex.Linenum;
YYParse;
Psdl_Lex_Io.Close_Input;
Psdl_Lex_Io.Close_Output;
Item := The_Program;
Text_Io.Close(Psdl_Lex.List_File);
end Get;

procedure Bind_Type_Declaration is

-- Bind Each Id In Id The Id
-- Set To The Type Name
-- Return Temp_Type_Decl
Procedure Bind_Type_Declaration(I_s: In Id_Set;
    Tn : In Type_Name;
    Td : in out Type_Declaration) is
begin
  --/* m4 code
  --/* foreach([Id: Psdl_Id], [Id_Set_PKG.Generic_Scan],
  --/*     [I_s],
  --/*     [Bind_Type_Decl_Map(Id, Tn, Td));
  --/*

  --/* Begin expansion of FOREACH loop macro.
  declare
    procedure Loop_Body(Id: Psdl_Id) is
    begin
      Bind_Type_Decl_Map(Id, Tn, Td);
    end Loop_Body;

    procedure Execute_Loop is
    new Id_Set_PKG.Generic_Scan(Loop_Body);
    begin
      execute_loop(I_s);
    end;
  --/* end of expansion of FOREACH loop macro.

  end Bind_Type_Declaration;


procedure Bind_Initial_State( State : in Type_Declaration;
    Init_Seq : in Exp_Seq;
    Init_Exp_Map: out Init_Map) is
i : Natural := 1;

  --/* M4 macro code for binding each initial expression in
  --/* the_init_expr_seq to the id's in state declaration map
  --/*
  --/* foreach([Id: in Psdl_Id; Tn: in Type_Name],
  --/* [Type_Declaration_PKG.Generic_Scan],
  --/* [State],
  --/* [Bind_Init_Map(Id, Exp_Seq_PKG.Fetch(The_Init_Exp_Seq, i)],
  --/* The_Initial_Expression)
  --/*
  --/* i := i + 1;

begin
  -- Begin expansion of FOREACH loop macro.
  declare
    procedure Loop_Body(Id: in Psdl_Id; Tn: in Type_Name) is

begin
  if i > Exp_Seq_Pkg.Length(The_Init_Expr_Seq) then
    Yyerror("SEMANTIC ERROR - Some states are not initialized.");
    Raise SEMANTIC_ERROR;
  else
    Bind_Init_Map(Id, Exp_Seq_Pkg.Fetch(The_Init_Expr_Seq, i),
    The_Initial_Expression);
    i := i + 1;
  end if;
end LoopBody;

procedure execute_loop is new TypeDeclaration_Pkg.Generic_Scan(LoopBody);
beginn
  execute_loop(State);
end;

-- LIMITATIONS: Square brackets are used as macro quoting characters,
-- so you must write "[x]" in the m4 source file
-- to get [x] in the generated Ada code.
-- Ada programs using FOREACH loops must avoid the lower case spellings of
-- the identifier names "DEFINE", "UNDEFINE", and "DNL",
-- or must quote them like this: [define].
-- The implementation requires each package to be generated by
-- a separate call to m4: put each package in a separate file.
-- Exit and return statements inside the body of a FOREACH loop
-- may not work correctly if FOREACH loops are nested.
-- An expression returned from within a loop body must not
-- mention any index variables of the loop.
-- End expansion of FOREACH loop mer:ro.

if (i-1) < Exp_Seq_Pkg.Length(The_Init_Expr_Seq) then
  Yyerror("SEMANTIC ERROR - There are more initializations than the states");
  raise SEMANTIC_ERROR;
end if;

end Bind_Initial_State;

-------------------------------------------------------------
procedure Make_PSDL_Type
  --
  -- construct the PSDL TYPE using global variables
  --
-------------------------------------------------------------

procedure Build_PSDL_Type
  (C_Name : in Psdl_Id;
   C_a_Name : in Ada_Id;
   Md1 : in Type_Declaration;
   D_Str : in Type_Name;
   Ops : in Operation_Map;
   G_Par : in out Type_Declaration;
   Kwr : in out Id_Set;
   I_Desc : in out Text;
   F_Desc : in out Text;
   IsAtomic : in Boolean;
The Type in out Data_Type) is

begin

if IS_ATOMIC then
    The_Type := Make_Atomc_Type
        ( PSDL_Name => C_Name,
          Ada_Name => C_A_Name,
          Model    => Mdl,
          Gen_Par  => G_Par,
          Operations=> Ops,
          Keywords => Kwr,
          Informal_Description  => I_Desc,
          Axioms    => F_Desc );

else
    The_Type := Make_Composite_Type
        ( Name       => C_Name,
          Model      => Mdl,
          Data_Structure => D_Str,
          Operations => Ops,
          Gen_Par    => G_Par,
          Keywords   => Kwr,
          Informal_Description => I_Desc,
          Axioms     => F_Desc );
end if;

-- /* After constructing the component */
-- /* initialized the global variables for */
-- /* optional attributes */
G_Par      := Empty_Type_Declaration;
Kwr        := Empty_Id_Set;
I_Desc     := Empty_Text;
F_Desc     := Empty_Text;
end Build_PSDL_Type;

--------------------------------------------------------------------------------

procedure Build_PSDL_Operator

--------------------------------------------------------------------------------
-- construct the PSDL OPERATOR using global variables
--------------------------------------------------------------------------------

procedure Build_PSDL_Operator
    ( C_Name        : in PSDL_Id;
      C_A_Name      : in Ada_Id;
      G_Par         : in out Type_Declaration;
      Kwr           : in out Id_Set;
      I_Desc        : in out Text;
      F_Desc        : in out Text;

348
begin

if IS_ATOMIC then

The_Opr := Make_Atomic_Operator

( Psdl_Name => C_Name,
  Ada_Name => C_A_Name,
  Gen_Par => G_Par,
  Keywords => Kwr,
  Informal_Description => I_Desc,
  Axioms => F_Desc,
  Input => Inp,
  Output => Otp,
  State => St,
  Initialization_Map => I_Exp_Map,
  Exceptions => Excps,
  Specified_Met => S_Met);

else

The_Opr := Make_Composite_Operator

( Name => C_Name,
  Gen_Par => G_Par,
  Keywords => Kwr,
  Informal_Description => I_Desc,
  Axioms => F_Desc,
  Input => Inp,
  Output => Otp,
  State => St,
  Initialization_Map => I_Exp_Map,
  Exceptions => Excps,
  Specified_Met => S_Met,
  Graph => Gr,

end if;
Streams => D_Stream,
Timers => Tmrs,
Trigger => Trigs,
Exec_Guard => E_Guard,
Out_Guard => O_Guard,
Excep_Trigger => E_Trigger,
Timer_Op => T_Op,
Per => Per,
Fw => Fw,
Mcp => Mcp,
Mrt => Mrt,
Impl_Desc => Im_Desc);
end if;

-- /* After constructing the component */
-- /* initialized the global variables for */
-- /* optional attributes */
G_Par := Empty_Type_Declaration;
Kwr := Empty_Id_Set;
I_Desc := Emtpy_Text;
F_Desc := Empty_Text;
Inp := Empty_Type_Declaration;
Otp := Empty_Type_Declaration;
St := Empty_Type_Declaration;
I_Exp_Map := Empty_Init_Map;
Excps := Empty_Id_Set;
S_Met := 0;
Gr := Empty_Psdl_Graph;
D_Stream := Empty_Type_Declaration;
Tmrs := Empty_Id_Set;
Trigs := Empty_Trigger_Map;
E_Guard := Empty_Exec_Guard_Map;
O_Guard := Empty_Out_Guard_Map;
E_Trigger := Empty_Excet_Trigger_Map;
T_Op := Empty_Timer_Op_Map;
Per := Empty_Timing_Map;
Fw := Empty_Timing_Map;
Mcp := Empty_Timing_Map;
Mrt := Empty_Timing_Map;
Im_Desc := Empty_Text;
end Build_Psdl_Operator;

-- -- procedure Add_Op_Impl_To_Op_Map
-- --
-- -- Uses the operation map we constructed only with the
-- -- specification part.
-- -- Fetches the operator from the map, uses to create a new one--
-- -- with it (specification part) and add the implementation --
-- -- to it.
procedure Add_Op_Impl_To_Op_Map(Op_Name : in Psdl_Id;
   A_Name : in Ada_Id;
   Is.Atomic : in Boolean;
   O_Map : in out Operation_Map;
   Gr : in out Psdl_Graph;
   D_Stream : in out Type_Declaration;
   Tmrs : in out Id_Set;
   Trigs : in out Trigger_Map;
   E_Guard : in out Exec_Guard_Map;
   O_Guard : in out Out_Guard_Map;
   E_Trigger : in out Excep_Trigger_Map;
   T_Op : in out Timer_Op_Map;
   Per : in out Timing_Map;
   Fw : in out Timing_Map;
   Mcp : in out Timing_Map;
   Mrt : in out Timing_Map;
   Im_Desc : in out Text ) is

   Temp_Op : Operator;
   Temp_Op_Ptr : Op_Ptr;

   begin
      if Operation_Map_Pkg.Member(Op_Name, Operation_Map_Pkg.Map(O_Map)) then
         Temp_Op :=
            Operation_Map_Pkg.Fetch(Operation_Map_Pkg.Map(O_Map), Op_Name).all;
         Operation_Map_Pkg.Remove(Op_Name, Operation_Map_Pkg.Map(O_Map));
         if Is.Atomic then
            Temp_Op := Make.AtomicOperator
               (Psdl_Name => Op_Name,
                Ada_Name => A_Name,
                Gen_Par => Generic_Parameters(Temp_Op),
                Keywords => Keywords(Temp_Op),
                Informal_Description => Informal_Description(Temp_Op),
                Axioms => Axioms(Temp_Op),
                Input => Inputs(Temp_Op),
                Output => Outputs(Temp_Op),
                State => States(Temp_Op),
                Initialization_Map => Get_Init_Map(Temp_Op),
                Exceptions => Exceptions(Temp_Op),
                Specified_Met =>
                   Specified_Maximum_Execution_Time(Temp_Op) );

            Temp_Op_Ptr := new Operator (Category => Psdl-operator,
                                          Granularity => Atomic);
            Temp_Op_Ptr.all := Temp_Op;
         else
            Temp_Op := Make_Composite_Operator
               (Name => Op_Name,
Gen_Par => Generic_Parameters(Temp_Op),
    Keywords => Keywords(Temp_Op),
    Informal_Description => Informal_Description(Temp_Op)
    Axioms => Axioms(Temp_Op),
Input => Inputs(Temp_Op),
Output => Outputs(Temp_Op),
State => States(Temp_Op),
Initialization_Map => Get_Init_Map(Temp_Op),
Exceptions => Exceptions(Temp_Op),
Specified_Met =>
    Specified_Maximum_Execution_Time(Temp_Op),
    Graph => Gr,
    Streams => D_Stream,
    Timers => Tmrs,
    Trigger => Trigs,
    Exec_Guard => E_Guard,
    Out_Guard => O_Guard,
    Excep_Trigger => E_Trigger,
    Timer_Op => T_Op,
    Per => Per,
    Fw => Fw,
    Mcp => Mcp,
    Mrt => Mrt,
    Impl_Desc => Im_Desc);

Temp_Op_Ptr := new Operator (Category => Psdl_Operator,
    Granularity => Composite);
Temp_Op_Ptr.all := Temp_Op;
end if;
Bind_Operation(Op_Name, Temp_Op_Ptr, O_Map);

-- reset everything after you are done.(the variables that
-- have default values)
Gr := Empty_Psdl_Graph;
D_Stream := Empty_Type_Declaration;
Tmrs := Empty_Id_Set;
Trigs := Empty_Trigger_Map;
E_Guard := Empty_Exec_Guard_Map;
O_Guard := Empty_Out_Guard_Map;
E_Trigger := Empty_Excep_Trigger_Map;
T_Op := Empty_Timer_Op_Map;
Per := Empty_Timing_Map;
Fw := Empty_Timing_Map;
Mcp := Empty_Timing_Map;
Mrt := Empty_Timing_Map;
Im_Desc := Empty_Text;
else
    Put("Warning: The specification of operator ");
    Put_Line(Op_Name.s & ";' was not given, implementation ignored.");
end if;
end Add_Op_Impl_ToOpMap;

procedure YYParse is
-- Rename User Defined Packages to Internal Names.
package yy_goto_tables renames Psdl_Goto;
package yy_shift_reduce_tables renames Psdl_Shift_Reduce;
package yy_tokens renames Psdl_Tokens;

use yy_tokens, yy_goto_tables, yy_shift_reduce_tables;

procedure yyerrok;
procedure yyclearin;

package yy is

-- the size of the value and state stacks
stack_size : constant Natural := 300;

-- subtype rule is natural;
subtype rule is natural;
-- subtype parse_state is natural;
-- subtype nonterminal is integer;

-- encryption constants
default : constant := -1;
first_shift_entry : constant := 0;
accept_code : constant := -1001;
error_code : constant := -1000;

-- stack data used by the parser
tos : natural := 0;
value_stack : array(0..stack_size) of yy_tokens.yystype;
state_stack : array(0..stack_size) of parse_state;

-- current input symbol and action the parser is on
action : integer;
rule_id : rule;
input_symbol : yy_tokens.token;

-- error recovery flag
error_flag : natural := 0;
-- indicates 3 - (number of valid shifts after an error occurs)
lookAhead : boolean := true;
index : integer;

-- Is Debugging option on or off
DEBUG : constant boolean := FALSE;

end yy;

function goto_state
(state : yy.parse_state;
sym : nonterminal) return yy.parse_state;

function parse_action
(state : yy.parse_state;
 t : yy_tokens.token) return integer;
pragma inline(goto_state, parse_action);

function goto_state(state : yy.parse_state;
 sym : nonterminal) return yy.parse_state is
index : integer;
begin
index := goto_offset(state);
while integer(goto_matrix(index).nonterm) /= sym loop
index := index + 1;
end loop;
return integer(goto_matrix(index).newstate);
end goto_state;

function parse_action(state : yy.parse_state;
 t : yy_tokens.token) return integer is
index : integer;
tok_pos : integer;
default : constant integer := -1;
begin
tok_pos := yy_tokens.token'pos(t);
index := shift_reduce_offset(state);
while integer(shift_reduce_matrix(index).t) /= tok_pos and then
integer(shift_reduce_matrix(index).t) /= default
loop
index := index + 1;
end loop;
return integer(shift_reduce_matrix(index).act);
end parse_action;

-- error recovery stuff

procedure handle_error is
  temp_action : integer;
begin
  if yy.error_flag = 3 then -- no shift yet, clobber input.
    if yy.debug then
      put_line("Ayacc.YYParse: Error Recovery Clobbers " &
        yy_tokens.token'image(yy.input_symbol));
    end if;
    if yy.input_symbol = yy_tokens.end_of_input then -- don't discard,
    if yy.debug then
      put_line("Ayacc.YYParse: Can't discard END_OF_INPUT, quitting...");
    end if;
    raise yy_tokens.syntax_error;
  end if;

354
```plaintext
yy.look_ahead := true; -- get next token
return; -- and try again...
end if;

if yy.error_flag = 0 then -- brand new error
  yyerror("Syntax Error");
end if;

yy.error_flag := 3;
-- find state on stack where error is a valid shift --
if yy.debug then
  put_line("Ayacc.YYParse: Looking for state with error as valid shift");
end if;

loop
  if yy.debug then
    put_line("Ayacc.YYParse: Examining State ",
      yy.parse_state'image(yy.state_stack(yy.tos)));
  end if;
  temp_action := parse_action(yy.state_stack(yy.tos), error);
  if temp_action >= yy.first_shift_entry then
    yy.tos := yy.tos + 1;
    yy.state_stack(yy.tos) := temp_action;
    exit;
  end if;
end loop;

Decrement Stack Pointer:
begin
  yy.tos := yy.tos - 1;
except
  when Constraint_Error =>
    yy.tos := 0;
end Decrement Stack Pointer;

if yy.tos = 0 then
  if yy.debug then
    put_line("Ayacc.YYParse:Error recovery popped entire stack, aborting...");
  end if;
  raise yy_tokens.syntax_error;
  end if;
end loop;

if yy.debug then
  put_line("Ayacc.YYParse: Shifted error token in state ",
    yy.parse_state'image(yy.state_stack(yy.tos)));
end if;
end handle_error;

-- print debugging information for a shift operation
procedure shift_debug(state_id: yy.parse_state; lexeme: yy_tokens.token) is begin
```

355
put_line("Ayacc.YYParse: Shift " & yy.parse_state\'image(state_id) & " on input symbol " & yy_tokens.token\'image(lexeme) ");

end;

-- print debugging information for a reduce operation
procedure reduce_debug(rule_id: rule; state_id: yy.parse_state) is
begin
  put_line("Ayacc.YYParse: Reduce by rule " & rule\'image(rule_id) & " goto state \\
  yy.parse_state\'image(state_id));
end;

-- make the parser believe that 3 valid shifts have occured.
-- used for error recovery.
procedure yyerrok is
begin
  yy.error_flag := 0;
end yyerrok;

-- called to clear input symbol that caused an error.
procedure yyclearin is
begin
  -- yy.input_symbol := yylex;
  yy.look-ahead := true;
end yyclearin;

begin
  -- initialize by pushing state 0 and getting the first input symbol
  yy.state_stack(yy.tos) := 0;

loop

  yy.index := shift_reduce_offset(yy.state_stack(yy.tos));
  if integer(shift_reduce_matrix(yy.index).t) = yy.default then
    yy.action := integer(shift_reduce_matrix(yy.index).act);
  else
    if yy.look-ahead then
      yy.look-ahead := false;
      yy.input_symbol := yylex;
    end if;
    yy.action :=
      parse_action(yy.state_stack(yy.tos), yy.input_symbol);
  end if;

  if yy.action >= yy.first_shift_entry then -- SHIFT

    if yy.debug then
      shift_debug(yy.action, yy.input_symbol);
    end if;

    -- Enter new state
    yy.tos := yy.tos + 1;
    yy.state_stack(yy.tos) := yy.action;

  end if;

end loop;
yy.value_stack(yy.tos) := yylval;

if yy.error_flag > 0 then -- indicate a valid shift
  yy.error_flag := yy.error_flag - 1;
end if;

-- Advance lookahead
yy.lookahead := true;

elsif yy.action = yy.error_code then -- ERROR
  handle_error;

elsif yy.action = yy.accept_code then
  if yy.debug then
    put_line("Ayacc.YYParse: Accepting Grammar...");
  end if;
  exit;
else -- Reduce Action

  -- Convert action into a rule
  yy.rule_id := -1 * yy.action;

  -- Execute User Action
  -- user_action(yy.rule_id);
  case yy.rule_id is

    when 1 =>
    --#line 358
    The_PROGRAM := Empty_PSDL_Program;

    when 3 =>
    --#line 366
    the_component_ptr := new PSDL_COMPONENT;

    when 4 =>
    --#line 369
    --/* the created object should always be constrained */
    --/* since object is a record with discriminants. */

      The_Component_Ptr :=
      new PSDL_Component
      (Category => Component_Category(The_Component),
       Granularity => Component_Granularity(The_Component));

      The_Component_Ptr.all := The_Component;
      Bind_Program (Name(The_Component),
                   The_Component_Ptr,
                   The_PROGRAM);

    when 8 =>
    --#line 401

yyval := (Token_Category => Psdl_Id_String,
Psdl_Id_Value => The_IdToken);
    The_Operation_Map := Empty_Operation_Map;

when 9 =>
--#line 408
-- construct the psdl type using global variables
-- psdl component record fields that have default values
-- are passed as in out parameters, so that after
-- building the component, they are initialized
-- back to their default values.

Build_Psdl_Type(
    yy.valuestack(yy.tos-2).Psdl_Id_Value,
    The_Ada_Name, The_Model, The_Data_Structure,
    The_Operation_Map, The_Type_Gen_Par, The_Keywords,
    The_Description, The_Axioms, Is_Atomic_Type, The_Component);

when 11 =>
--#line 440
    Type_DeclStack_Pkg.Push(The_Type_Decl_Stack,
        Empty_Type_Declaration);
    Type_Spec_Gen_Par := TRUE;

when 12 =>
--#line 447
    Type_DeclStack_Pkg.Pop(The_Type_Decl_Stack,
        The_Type_Gen_Par);
    Type_Spec_Gen_Par := FALSE;

when 14 =>
--#line 458
    Type_DeclStack_Pkg.Push(The_Type_Decl_Stack,
        Empty_Type_Declaration);

when 15 =>
--#line 464
Type_Dec1_Stack_Pkg.Pop(The_Type_Dec1_Stack, The_Model);

when 17 =>
   --#line 476
   The_Op_Ptr := new Operator;
when 18 =>
   --#line 479

   yyval := (Token_Category => Psdl_Id_String,
            Psdl_Id_Value => The_Id_Token);
   -- create a new operator(composite) to put in ops map
   -- make it composite because we don't know what
   -- the granularity is at this point.
   The_Op_Ptr := new Operator(Category => Psdl_Operator,
                                Granularity => Composite);

when 19 =>
   --#line 491

   Build_Psold_Operator(
The_Op_Ptr.all := The_Operator;
Bind_Operation(
    yy.value_stack(yy.tos-1).Psdl_Id_Value,
    The_Op_Ptr,
    The_Operation_Map);

when 21 =>
  --$line 533

  yyval := (Token_Category => Psdl_Id_String,
            Psdl_Id_Value => The_Id.Token);

when 22 =>
  --$line 539

  -- construct the psdl operatot
  -- using the global variables

  Build_Psdl_Operator(
    yy.value_stack(yy.tos-2).Psdl_Id_Value,
    The_Ada_Name,
    The_Gen_par,
    The_Keywords,
    The_Description,
    The_Axioms,
    The_Input,
    The_Output,
    The_State,
    The_Initial_Expression,
    The_Exceptions,
    The_Specified_Met,
    The_Graph,
    The_Streams,
    The_Timers,
    The_Trigger,
    The_Exec_Guard,
    The_Out_Guard,
    The_Exception_Trigger,
    The_Timer.Op,
    The_Per,
    The_Fw,
    The_Mcp,
    The_Mrt,
    The_Impl.Desc,
    Is_Atom_Operator,
    The_Component);

when 26 =>
  --$line 589

  Type_Decl_stack_Pkg.Push(The_Type_Decl_Stack,
                           Empty_Types_Declaration);
when 27 =>
--#line 595

Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
The_Gen_Par);

when 28 =>
--#line 602

Type_Decl_Stack_Pkg.Push(The_Type_Decl_Stack,
Empty_Type_Declaration);

when 29 =>
--#line 609

Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
The_Input);

when 30 =>
--#line 616

Type_Decl_Stack_Pkg.Push(The_Type_Decl_Stack,
Empty_Type_Declaration);

when 31 =>
--#line 622

Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
The_Output);

when 32 =>
--#line 629

Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
Empty_Type_Declaration);
Id_Seq_Pkg.Empty(The_Id_Seq);
-- empcv id seq, to use with init map

when 33 =>
--#line 637

Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
The_State);
The_Init_Map_Id_Seq := The_Id_Seq;
-- hold the id's for init map.
when 34 =>
--#line 647
    Init_Exp_Seq_Stack_Pkg.Push(The_Init_Exp_Seq_Stack,
        Empty_Exp_Seq);
    The_Expression_String := Expression(A_Strings.Empty);

when 35 =>
--#line 655
    Init_Exp_Seq_Stack_Pkg.Pop(The_Init_Exp_Seq_Stack,
        The_Init_Exp_Seq);
    Bind_Initial_State(The_State,
        The_Init_Expr_Seq,
        The_Initial_Expression);

when 36 =>
--#line 665
    Id_Set_Pkg.Empty(The_Id_Set);

when 37 =>
--#line 670
    Id_Set_Pkg.Assign(The_Exceptions, The_Id_Set);

when 38 =>
--#line 678
    The_Specified_Met :=
        yy.value_stack(yy.tos).Integer_Value;

when 41 =>
--#line 695
    The_Id_Set := Empty_Id_Set;

when 42 =>
--#line 700
    The_Expression_String := The_Expression_String & " : ";
    Id_Set_Stack_Pkg.Push(The_Id_Set_Stack, The_Id_Set);

when 43 =>
--#line 706
    Type_Deci_Stack_Pkg.Pop(The_Type_Deci_Stack,
        Temp_Type_Deci);
--/* Bind each id in id the id set to the type name */
--/* in the internal stack($5), return temp_type_decl */
Bind_Type_Declaration(
    Id_Set_Stack_Pkg.Top(The_Id_Set_Stack),
    yy.value_stack(yy.tos).Type_Name_Value,
    Temp_Type_Deci);

Type_Deci_Stack_Pkg.Push(The_Type_Deci_Stack,
    Temp_Type_Deci);

--/* pop the stack after bind */
Id_Set_Stack_Pkg.Pop(The_Id_Set_Stack);

when 44 =>
---#line 729

yyval := (Token_Category => Psdl_Id_String,
    Psdl_Id_Value => The_IdToken);

The_Expression_String := The_Expression_String & " "
& Expression(The_IdToken);

when 45 =>
---#line 738

Type_Deci_Stack_Pkg.Push(The_Type_Deci_Stack,
    Empty_Type_Declaration);

The_Expression_String := The_Expression_String & " [";

when 46 =>
---#line 746

The_Type_Name := New_Type_Name_Record;
    The_Type_Name.Name :=
    yy.value_stack(yy.tos-3).Psdl_Id_Value;

    The_Type_Name.Gen_Par := Type_Deci_Stack_Pkg.Top(The_Type_Deci_Stack);

yyval := (Token_Category => Type_Name_String,
    Type_Name_Value => The_Type_Name);

Type_Deci_Stack_Pkg.Pop(The_Type_Deci_Stack);

when 47 =>
---#line 758

The_Expression_String := The_Expression_String & "] ";

363
when 48 =>

-- #line 761
\[\text{this an awkward way of working around the problem we get when we have two identifiers one after another if TypeSpec GenPar and not IdSetPkg.Member(ThePrevIdToken, The_Id_Set) then}
\]
\[
\begin{align*}
\text{The Type Name} & := \\
\text{New TypeNameRecord'(ThePrevIdToken, EmptyTypeDeclaration);} \\
\text{The Expression String} & := \text{The Expression String } \& \text{""} \\
& \text{& Expression(ThePrevIdToken);} \\
\text{else} & \\
\text{The Type Name} & := \\
\text{New TypeNameRecord'(TheIdToken, EmptyTypeDeclaration);} \\
\text{The Expression String} & := \text{The Expression String } \& \text{""} \\
& \text{& Expression(TheIdToken);} \\
\text{end if;}
\end{align*}
\]

yyval := (Token_Category => TypeNameString,
Type_Name_Value => The_Type_Name);

when 49 =>

-- #line 793
\[
\text{The Expression String} := \text{The Expression String } \& \text{"", "} ;
\]

when 50 =>

-- #line 796
\[
\begin{align*}
\text{Id Set Pkg.Add(TheId_Token, The_Id_Set);} \\
\text{The String} & := \text{The String } \& \"", " \& \text{TheId_Token;} \\
\text{IdSeq Pkg.Add(TheId_Token, TheIdSeq);} \\
\text{The Expression String} & := \text{The Expression String } \& \"" \\
& \text{& Expression(TheId_Token);} \\
\end{align*}
\]

when 51 =>

-- #line 805
\[
\begin{align*}
\text{Id Set Pkg.Add(TheId_Token, The_Id_Set);} \\
\text{The String} & := \text{The Id Token;} \\
\text{IdSeq Pkg.Add(TheId_Token, TheIdSeq);} \\
\text{The Expression String} & := \text{The Expression String } \& \"" \\
& \text{& Expression(TheId_Token);} \\
\end{align*}
\]

when 55 =>

-- #line 828
\[
\text{Id Set Pkg.Empty(TheId Set);} \\
\]

364
when 56 =>
    --#line 833
    Id_Set_Pkg.Assign(The_Keyword, The_id_Set);

when 57 =>
    --#line 837
    The_Keyword := Empty_Id_Set;

when 58 =>
    --#line 843
    The_Description := The_Text_Token;
    The_Impl_Desc := The_Text_Token;

when 60 =>
    --#line 853
    The_Axioms := The_Text_Token;

when 62 =>
    --#line 862
    Is.Atomic_Type := True;
    The_Ada_Name := Ada_Id(The_Id_Token);

when 64 =>
    --#line 871
    Is.Atomic_Type := False;
    The_Data_Structure :=
        yy.value_stack(yy.tos).Type_Name_Value;

when 66 =>
    --#line 883
    The_Op_Ptr := New Operator;

when 67 =>
    --#line 886

    yyval := (Token_Category => Psdl_Id_String,
              Psdl_Id_Value  => The_Id_Token);

when 68 =>
    --#line 891
-- add implementation part to the operator in the operation map
AddOpImplToOpMap(
  yy.valuestack(yy.tos-1).Psdl_Id_Value,
  The_Ada_Name,
  The_Operation_Map,
  The_Graph,
  The_Streams,
  The_Timers,
  The_Trigger,
  The_Exec_Guard,
  The_Out_Guard,
  The_Excep_Trigger,
  The_Timer_Op,
  The_Per,
  The_Fw,
  The_Mcp,
  The_Mrt,
  The_Impl_Desc);

when 70 => 
  --#line 917

  Is.Atomic-operator := True;
  The_Ada_Name := Ada_Id(The_Id.Token);

when 72 => 
  --#line 925

  Is.Atomic-operator := False;

when 74 => 
  --#line 934
  The_Impl.Desc := Empty_Text;

when 76 => 
  --#line 942
  The_Graph := Empty_Psdl_Graph;

when 78 => 
  --#line 950

  The_Graph := Psdl_Graph_Pkg.Add_Vertex(
    yy.valuestack(yy.tos-1).Psdl_Id_Value,
    The_Graph,
    yy.valuestack(yy.tos).Integer_Value);

when 80 => 
  --#line 961
  The_Edge_Name := The_Id.Token;
when 81 =>
--$line 964

   The_Graph := Psdl_Graph_Pkg.Add_Edge(
            yy.value_stack(yy.tos-2).Psdl_Id_Value,
            yy.value_stack(yy.tos).Psdl_Id_Value,
            The_Edge_Name,
            The_Graph,
            yy.value_stack(yy.tos-3).Integer_Value);

when 83 =>
--$line 978

   yyval := (Token_Category => Psdl_Id_String,
            Psdl_Id_Value => The_Id_Token);

when 84 =>
--$line 984

   yyval := ( Token_Category => Psdl_Id_String,
             Psdl_Id_Value =>
             yy.value_stack(yy.tos-1).Psdl_Id_Value
             &
             yy.value_stack(yy.tos).Psdl_Id_Value );

when 85 =>
--$line 993

   The_String := Psdl_Id(A_Strings.Empty);

when 86 =>
--$line 996

   yyval := ( Token_Category => Psdl_Id_String,
             Psdl_Id_Value => "(" & The_String);
             The_String := Psdl_Id(A_Strings.Empty);

when 87 =>
--$line 1004

   yyval := ( Token_Category => Psdl_Id_String,
             Psdl_Id_Value =>
             yy.value_stack(yy.tos-3).Psdl_Id_Value
             & "|" & The_String & ")" );
when 88 =>
  -- #line 1010
 yyval := (Token_Category => Psdl_Id_String,
  Psdl_Id_Value => Psdl_Id(A_Strings.Empty));

when 91 =>
  -- #line 1026
 yyval := (Token_Category => Integer_Literal,
  Integer_Value => yy.value_stack(yy.tos).Integer_Value);

when 92 =>
  -- #line 1031
 yyval := (Token_Category => Integer_Literal,
  Integer_Value => 0);

when 93 =>
  -- #line 1038
 Type_Decl_Stack_Pkg.Push(The_Type_Decl_Stack,
  Empty_Type_Declaration);

when 94 =>
  -- #line 1044
 Type_Decl_Stack_Pkg.Pop(The_Type_Decl_Stack,
  The_Streams);

when 96 =>
  -- #line 1059
 Id_Set_Pkg.Empty(The_Id_Set);

when 97 =>
  -- #line 1064
 Id_Set_Pkg.Assign(The_Timers, The_Id_Set);

when 98 =>
  -- #line 1068
 Id_Set_Pkg.Assign(The_Timers, Empty_Id_Set);
when 99 =>
  -- #line 1077
  TheOperator_Name := The_IdToken;
The_Trigger := Empty_Trigger_Map;
The_Per := Empty_Timing_Map;
The_Fw := Empty_Timing_Map;
The_Mcp := Empty_Timing_Map;
The_Mrt := Empty_Timing_Map;
The_Exec_Guard := Empty_Exec_Guard_Map;
The_Out_Guard := Empty_Out_Guard_Map;
The_Excep_Trigger := Empty_Excep_Trigger_Map;
The_Timer_Op := Empty_Timer_Op_Map;

when 101 =>
  -- #line 1094
  TheOperator_Name := The_IdToken;

when 103 =>
  -- #line 1102
  TheOperator_Name := The_IdToken;

when 105 =>
  -- #line 1113
  The_Id_Set := Empty_Id_Set;
The_Expression_String := Expression(A_Strings.Empty);
The_Output_Id.Op := The.Operator_Name;

when 106 =>
  -- #line 1120
  The_Expression_String := Expression(A_Strings.Empty);

when 107 =>
  -- #line 1125

  -- Begin Expansion Of Foreach Loop Macro.
  declare
    procedure Loop_Body(Id : Pidl_Id) is
    begin
      The_Output_Id.Stream := Id;
      Bind_Out_Guard(The_Output_Id, The_Expression_String, The_Out_Guard);

  369
end Loop_Body;
procedure Execute_Loop is
  new Id_Set_Pkg.Generic_Scan(Loop_Body);
begin
  Execute_Loop(The_Id_Set);
end;

when 108 =>
  yyval := (Token_Category => Psdl_Id_String,
             Psdl_Id_Value  => The_Id_Token);
  Th_Expression_String := Expression(A_Strings.Empty);

when 109 =>
  The_Excep_Id.Op := The_Operator_Name;
  The_Excep_Id.Excep :=
  yy.value_stack(yy.tos-2).Psdl_Id_Value;
  Bind_Excep_Trigger( The_Excep_Id, The_Excep_Trigger);

when 110 =>
  yyval := (Token_Category => Psdl_Id_String,
             Psdl_Id_Value  => The_Id_Token);
  The_Expression_String := Expression(A_Strings.Empty);

when 111 =>
  The_Timer_Op_Record.Op_Id :=
  yy.value_stack(yy.tos-4).Timer_Op_Id_Value;
  The_Timer_Op_Record.Timer_Id :=
  yy.value_stack(yy.tos-2).Psdl_Id_Value;
  The_Timer_Op_Record.Guard := The_Expression_String;
  Timer_Op_Set_Pkg.Add (The_Timer_Op_Record, The_Timer_Op_Set);
  Bind_Timer_Op(The_Operator_Name, The_Timer_Op_Set, The_Timer_Op);

when 113 =>
  yyval := (Token_Category => Psdl_Id_String,
             Psdl_Id_Value  => The_Id_Token);
  Th_Expression_String := Expression(A_Strings.Empty);
The_Expression_String := Expression(A_Strings.Empty);

when 114 =>
---#line 1191

Bind_Exec_Guard(The_Operator_Name,
    The_Expression_String,
    The_Exec_Guard);

when 116 =>
---#line 1202

The_Id_Set := Empty_Id_Set;

when 117 =>
---#line 1207

    The_Trigger_Record.Tt := By_All;
    The_Trigger_Record.Streams := The_Id_Set;

    Bind_Trigger(The_Operator_Name,
        The_Trigger_Record,
        The_Trigger);

when 118 =>
---#line 1217

    The_Id_Set := Empty_Id_Set;

when 119 =>
---#line 1222

    The_Trigger_Record.Tt := By_Some;
    The_Trigger_Record.Streams := The_Id_Set;
    Bind_Trigger(The_Operator_Name,
        The_Trigger_Record,
        The_Trigger);

when 120 =>
---#line 1232
    -- we don't care what is in the id set
    The_Trigger_Record.Tt := None;
    The_Trigger_Record.Streams := The_Id_Set;
    Bind_Trigger(The_Operator_Name,
        The_Trigger_Record,
        The_Trigger);
when 121 =>
--$line 1245

    Bind_Timing(The_Operator_Name,
    yy.value_stack(yy.tos).Integer_Value,
    The_Per);

when 123 =>
--$line 1257

    Bind_Timing(The_Operator_Name,
    yy.value_stack(yy.tos-1).Integer_Value,
    The_Fw);

when 125 =>
--$line 1268

    Bind_Timing(The_Operator_Name,
    yy.value_stack(yy.tos-1).Integer_Value,
    The_Mcp);

when 127 =>
--$line 1279

    Bind_Timing(The_Operator_Name,
    yy.value_stack(yy.tos).Integer_Value,
    The_Mrt);

when 130 =>
--$line 1295

    yyval := (Token_Category    => Timer_Op_Id_String,
               Timer_Op_Id_Value => Reset);

when 131 =>
--$line 1302

    yyval := (Token_Category    => Timer_Op_Id_String,
               Timer_Op_Id_Value => Start);

when 132 =>
--$line 1309
yyval := (Token_Category => Timer_Op_Id_String,
  Timer_Op_Id_Value => Stop);

when 135 =>
  --#line 1335
  The_Expression_String := Expression(A_Strings.Empty);

when 136 =>
  --#line 1340
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    TemplInitExpr_Seq);
  Exp_Seq_Pkg.Add (yy.value_stack(yy.tos).Expression_Value,
    Templ_Init_Expr_Seq);
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    Templ_Init_Expr_Seq);

when 137 =>
  --#line 1350
  The_Expression_String := Expression(A_Strings.Empty);

when 138 =>
  --#line 1355
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    Templ_Init_Expr_Seq);
  Exp_Seq_Pkg.Add (yy.value_stack(yy.tos).Expression_Value,
    Templ_Init_Expr_Seq);
  Init_Exp_Seq_Pkg.Pop (The_Init_Exp_Seq_Stack,
    Templ_Init_Expr_Seq);

when 139 =>
  --#line 1381
  yyval := (Token_Category => Expression_String,
    Expression_Value => To_A( "True"));

when 140 =>
  --#line 1388
  yyval := (Token_Category => Expression_String,
Expression_Value => To_A("False");

when 141 =>
   --#line 1395

   yyval := (Token_Category => Expression_String,
              Expression_Value => Expression(The_IntegerToken));

when 142 =>
   --#line 1401

   yyval := (Token_Category => Expression_String,
              Expression_Value => The_RealToken);

when 143 =>
   --#line 1407

   yyval := (Token_Category => Expression_String,
              Expression_Value => The_StringToken);

when 144 =>
   --#line 1413

   yyval := (Token_Category => Expression_String,
              Expression_Value => Expression(The_IdToken));

when 145 =>
   --#line 1423

   The_Expression_String := The_Expression_String & "." &
                           Expression(The_IdToken);

   yyval := (Token_Category => Expression_String,
              Expression_Value => The_Expression_String);

when 146 =>
   --#line 1431

   yyval := (Token_Category => Expression_String,
              Expression_Value => The_Expression_String & "."
                              & Expression(The_IdToken));

when 147 =>

374
Init_Exp_Seq_Stack_Pkg.Push(The_Init_Exp_Seq_Stack, Empty_Exp_Seq);

when 148 =>
--$line 1444
  */ we remove expression resulted by the */
  */ previous rule, since expression will */
  */ be concatenation of Type_name.ID and */
  */ value of previous production */
Init_Exp_Seq_Stack_Pkg.Pop(The_Init_Exp_Seq_Stack, Temp_Init_Exp_Seq);

The_Expression_String := Expression(A_Strings.Empty);

for i in 1 .. Exp_Seq_Pkg.Length(Temp_Init_Exp_Seq) loop
  if i > 1 then
    The_Expression_String := The_Expression_String & ",";
    end if;
    The_Expression_String :=
      The_Expression_String &
      Exp_Seq_Pkg.Fetch(Temp_Init_Exp_Seq, i);
  end loop;
Exp_Seq_Pkg.Recycle(Temp_Init_Exp_Seq); -- throw it away

yyval := (Token_Category => Expression_String,
  Expression_Value =>
    yy.value_stack(yy.tos-4).Expression_Value & "(" &
    The_Expression_String & ")");

when 149 =>
--$line 1471

yyval := (Token_Category => Expression_String,
  Expression_Value => To_A("(") &
    yy.value_stack(yy.tos-1).Expression_Value &
    To_A(")"));

when 150 =>
--$line 1480

yyval := (Token_Category => Expression_String,
  Expression_Value =>
    yy.value_stack(yy.tos-1).Expression_Value &
yy.value_stack(yy.tos).Expression_Value);

when 151 =>
--#line 1487

yyval := (Token_Category => Expression_String,
            Expression_Value =>
            yy.value_stack(yy.tos-1).Expression_Value &
            yy.value_stack(yy.tos).Expression_Value);

when 152 =>
--#line 1497

yyval := (Token_Category => Expression_String,
            Expression_Value =>
            yy.value_stack(yy.tos-2).Expression_Value &
            yy.value_stack(yy.tos-1).Expression_Value &
            yy.value_stack(yy.tos).Expression_Value);

when 153 =>
--#line 1507

yyval := (Token_Category => Expression_String,
            Expression_Value =>
            To_A("-") &
            yy.value_stack(yy.tos).Expression_Value);

when 154 =>
--#line 1513

yyval := (Token_Category => Expression_String,
            Expression_Value =>
            To_A("+") &
            yy.value_stack(yy.tos).Expression_Value);

when 155 =>
--#line 1522

yyval := (Token_Category => Expression_String,
            Expression_Value =>
            yy.value_stack(yy.tos-2).Expression_Value &
            yy.value_stack(yy.tos-1).Expression_Value &
yy.value_stack(yy.tos).Expression_Value);

when 156 =>
   --#line 1533

yyval := (Token_Category => Expression_String,
   Expression_Value =>
   yy.value_stack(yy.tos-2).Expression_Value &
   yy.value_stack(yy.tos-1).Expression_Value &
   yy.value_stack(yy.tos).Expression_Value);

when 157 =>
   --#line 1544

   yyval := (Token_Category => Expression_String,
   Expression_Value =>
   ToA(" EXP ") &
   yy.value_stack(yy.tos).Expression_Value);

when 158 =>
   --#line 1555
   --Exp_Seq_Pkg.Add( The_Expression_String, The_Exp_Seq);

   yyval := (Token_Category => Expression_String,
   Expression_Value => ToA(" NOT ") &
   yy.value_stack(yy.tos).Expression_Value);

when 159 =>
   --#line 1565

   yyval := (Token_Category => Expression_String,
   Expression_Value => ToA(" NOT ") &
   yy.value_stack(yy.tos).Expression_Value);

when 160 =>
   --#line 1575

   yyval := (Token_Category => Expression_String,
   Expression_Value => ToA(" AND "));
when 161 =>
-- #line 1581

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" OR "));

when 162 =>
-- #line 1587

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" XOR "));

when 163 =>
-- #line 1597

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" < "));

when 164 =>
-- #line 1603

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" > "));

when 165 =>
-- #line 1609

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" = "));

when 166 =>
-- #line 1615

yyval := (Token_Category => Expression_String,
          Expression_Value => To_A(" >= "));

when 167 =>
-- #line 1622

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" <= ");

when 166 =>
--$line 1625

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" /= ");

when 169 =>
--$line 1640

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" + ");

when 170 =>
--$line 1646

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" - ");

when 171 =>
--$line 1652

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" & ");

when 172 =>
--$line 1661

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" + ");

when 173 =>
--$line 1667

yyval := (Token_Category => Expression_String,
Expression_Value => To_A(" - ");

when 174 =>
--$line 1673

379
yyval := (Token_Category => Expression_String,
Expression_Value => To_A("MOD "));

when 175 =>
--$line 1679

yyval := (Token_Category => Expression_String,
Expression_Value => To_A("REM "));

when 176 =>
--$line 1689

yyval := (Token_Category => Integer_Literal,
Integer_Value => yy.value_stack(yy.tos-1).Integer_Value + 999)/1000);
The_Time_String :=
To_A(Integer'Image(yy.value_stack(yy.tos-1).Integer_Value) & " microsec");

when 177 =>
--$line 1697

yyval := (Token_Category => Integer_Literal,
Integer_Value => yy.value_stack(yy.tos-1).Integer_Value);
The_Time_String :=
To_A(Integer'Image(yy.value_stack(yy.tos-1).Integer_Value) & " ms");

when 178 =>
--$line 1705

yyval := (Token_Category => Integer_Literal,
Integer_Value => yy.value_stack(yy.tos-1).Integer_Value * 1000);
The_Time_String :=
To_A(Integer'Image(yy.value_stack(yy.tos-1).Integer_Value) & " sec");

when 179 =>
--$line 1714

yyval := (Token_Category => Integer_Literal,
Integer_Value => yy.value_stack(yy.tos-1).Integer_Value * 60000);
The_Time_String :=
To_A(Image(yyval := (Token_Category => Integer_Literal, Integer_Value => yy.value_stack(yy.tos-1).Integer_Value * 3600000);
The_Time_String := To_A(Image(yyval := (Token_Category => Integer_Literal, Integer_Value => Convert_To_Digit(The_Integer_Token.S));

when 186 =>
--$line 1771

The_Expression_String := The_Expression_String & " TRUE ";

when 187 =>
--$line 1776

The_Expression_String := The_Expression_String & " FALSE ";

when 1f3 =>
--$line 1782

The_Expression_String := The_Expression_String & " " & Expression(The_Integer_Token);

381
when 169 =>
---#line 1788

   The_Expression_String := The_Expression_String & " " &
   The_Time_String;

when 190 =>
---#line 1794

   The_Expression_String := The_Expression_String & " " &
   The_Real_Token;

when 191 =>
---#line 1800

   The_Expression_String := The_Expression_String & " " &
   The_String_Token;

when 192 =>
---#line 1806

   The_Expression_String := The_Expression_String & " " &
   Expression(The_Id_Token);

when 193 =>
---#line 1814

   The_Expression_String := The_Expression_String & "." &
   Expression(The_Id_Token);

when 194 =>
---#line 1820

   The_Expression_String := The_Expression_String & "." &
   Expression(The_Id_Token);

when 195 =>
---#line 1826

   The_Expression_String := The_Expression_String & " (";

when 196 =>
---#line 1829

   The_Expression_String := The_Expression_String & ") "
   Exp_Seq_Pkg.Add( The_Expression_String, The_Exp_Seq);

when 197 =>
---#line 1836
The.Expression_String := The.Expression_String & " (";

when 198 =>
   --$line 1839
   The.Expression_String := The.Expression_String & " ) ";

when 199 =>
   --%line 1842
   
   The.Expression_String :=
   The.Expression_String &
   yy.value_stack(yy.tos).Expression_Value;

when 201 =>
   --%line 1851
   
   The.Expression_String :=
   The.Expression_String &
   yy.value_stack(yy.tos).Expression_Value;

when 203 =>
   --%line 1859
   The(Expression_String := The.Expression_String & "-";

when 205 =>
   --%line 1864
   The(Expression_String := The.Expression_String & "+");

when 207 =>
   --%line 1869
   
   The(Expression_String :=
   The.Expression_String &
   yy.value_stack(yy.tos).Expression_Value;

when 209 =>
   --%line 1877
   
   The(Expression_String :=
   The.Expression_String &
   yy.value_stack(yy.tos).Expression_Value;

when 211 =>
   --%line 1885
   
   The(Expression_String :=
   The.Expression_String & " EXP ";

when 213 =>
   --%line 1892
   The.Expression_String := To_A(" NOT ");
when 215 =>
--@line 1897
The_Expression_String := To_A(" ABS ");

   when others => null;
end case;

-- Pop RHS states and goto next state
yy.tos := yy.tos - rule_length(yy.rule_id) + 1;
yy.state_stack(yy.tos) := goto_state(yy.state_stack(yy.tos-1),
                                 get_lhs_rule(yy.rule_id));

yy.value_stack(yy.cos) := yyval;

if yy.debug then
   reduce_debug(yy.rule_id,
              goto_state(yy.state_stack(yy.tos-1),
                          get_lhs_rule(yy.rule_id)));
end if;

end if;
end loop;

end yyparse;

end Parser;
package Psdl_Goto is

  type Small_Integer is range -32_000 .. 32_000;

  type Goto_Entry is record
    Nonterm : Small_Integer;
    Newstate : Small_Integer;
  end record;

  -- pragma suppress(index_check);

  subtype Row is Integer range -1 .. Integer'Last;

  type Goto_Parse_Table is array (Row range <>) of Goto_Entry;

  Goto_Matrix : constant Goto_Parse_Table :=
    ((-1,-1) -- Dummy Entry.
    -- State 0
    ,(-3, 1),(-2, 2)
    -- State 1
    ,(-4, 3)
    -- State 2
    -- State 3
    ,(-5, 5)
    -- State 4
    -- State 5
    ,(-8, 7),(-7, 6),(-6, 10)
    -- State 6
    -- State 7
    -- State 8
    -- State 9
    -- State 10
    -- State 11
    ,(-9, 13)
    -- State 12
    ,(-22, 14)
    -- State 13
    ,(-10, 16)
    -- State 14
    ));
<table>
<thead>
<tr>
<th>State</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>167</td>
<td>(-98, 179), (-102, 196)</td>
</tr>
<tr>
<td>168</td>
<td>(-106, 199), (-105, 198), (-104, 197)</td>
</tr>
<tr>
<td>169</td>
<td>(-98, 201), (-40, 166)</td>
</tr>
<tr>
<td>170</td>
<td>(-98, 202)</td>
</tr>
<tr>
<td>171</td>
<td>(-98, 203), (-40, 166)</td>
</tr>
<tr>
<td>172</td>
<td>(-98, 204)</td>
</tr>
<tr>
<td>173</td>
<td>(-40, 166)</td>
</tr>
<tr>
<td>174</td>
<td>(-84, 206)</td>
</tr>
<tr>
<td>175</td>
<td>(-65, 207)</td>
</tr>
<tr>
<td>176</td>
<td>(-71, 209)</td>
</tr>
<tr>
<td>177</td>
<td>(-35, 208)</td>
</tr>
<tr>
<td>178</td>
<td>(-98, 210), (-40, 166)</td>
</tr>
<tr>
<td>179</td>
<td>(-106, 199)</td>
</tr>
<tr>
<td>180</td>
<td>(-105, 198), (-104, 197), (-102, 196)</td>
</tr>
<tr>
<td>181</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td></td>
</tr>
<tr>
<td>185</td>
<td></td>
</tr>
<tr>
<td>186</td>
<td></td>
</tr>
<tr>
<td>187</td>
<td></td>
</tr>
<tr>
<td>188</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
</tr>
</tbody>
</table>
-- State 214
   (-106, 199), (-105, 198)
   (-104, 197), (-102, 196)
-- State 215
   (-106, 199), (-105, 198)
   (-104, 197), (-102, 196)
-- State 216
   (-106, 199), (-105, 198)
   (-104, 197), (-102, 196)
-- State 217
   (-106, 199), (-105, 198)
   (-104, 197), (-102, 196)
-- State 218
   (-78, 225)
-- State 219
   (-92, 228)
-- State 220
   (-79, 230)
-- State 221
   (-65, 231)
-- State 222
-- State 223
-- State 224
   (-106, 199), (-105, 198)
   (-104, 197), (-102, 196)
-- State 225
   (-79, 234)
-- State 226
   (-94, 235)
-- State 227
   (-95, 236)
-- State 228
   (-93, 237)
-- State 229
   (-107, 107), (-36, 238)
-- State 230
   (-80, 240)
-- State 231
-- State 232
   (-71, 241), (-35, 208)
-- State 233
   (-101, 242)
-- State 234
   (-80, 243)
-- State 235
   (-35, 244)
-- State 236
   (-35, 245)
-- State 237
\((-89, 247)\)

-- State 238
\((-26, 248)\)

-- State 239

-- State 240
\((-81, 251)\)

-- State 241

-- State 242
\((-99, 151), (-33, 253)\)

-- State 243
\((-81, 254)\)

-- State 244

-- State 245

-- State 246
\((-107, 107), (-87, 264), (-40, 262)\)
\((-36, 258)\)

-- State 247
\((-26, 269)\)

-- State 248

-- State 249
\((-107, 107), (-36, 270)\)

-- State 250

-- State 251
\((-96, 272), (-82, 274)\)

-- State 252

-- State 253

-- State 254
\((-96, 272), (-82, 276)\)

-- State 255

-- State 256

-- State 257

-- State 258

-- State 259

-- State 260

-- State 261
\((-41, 51)\)
-- State 330
,(-26, 335)
-- State 331

-- State 332
,(-107, 107),(-87, 338),(-40, 262)
,(-36, 258)
-- State 333
,(-107, 107),(-87, 339),(-40, 262)
,(-36, 258)
-- State 334

-- State 335

-- State 336
,(-109, 340)
-- State 337

-- State 338
,(-106, 282),(-105, 281)
,(-104, 280),(-102, 279)
-- State 339
,(-106, 282),(-105, 281)
,(-104, 280),(-102, 279),(-26, 341)
-- State 340
,(-107, 107)
,(-87, 342),(-40, 262),(-36, 258)
-- State 341

-- State 342
,(-106, 282)
,(-105, 281),(-104, 280),(-102, 279)
);

-- The offset vector
GOTO_OFFSET : array (0..342) of Integer :=
( 0,
2, 3, 4, 4, 7, 7, 7, 7, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
18, 19, 19, 22, 25, 25, 28, 31, 32, 33,
33, 34, 35, 36, 37, 38, 39, 40, 40, 41,
41, 42, 43, 43, 43, 44, 44, 44, 45, 45,
46, 48, 49, 50, 51, 51, 54, 57, 60, 63,
64, 64, 65, 65, 66, 66, 66, 67, 68, 69,
69, 71, 71, 71, 71, 72, 72, 73, 73, 73,
73, 73, 73, 73, 74, 74, 76, 76, 76, 76,
77, 78, 79, 80, 80, 80, 80, 81, 82, 82,
82, 85, 85, 85, 85, 85, 85, 85, 85, 85,
88, 89, 89, 90, 90, 91, 91, 92, 92, 93,
94, 94, 95, 95, 95, 95, 95, 95, 95, 95,
96, 97, 97, 98, 99, 99, 99, 100, 100, 101,
103, 104, 104, 105, 107, 109, 109, 109, 109, 110,
110, 112, 112, 112, 113, 113, 113, 113, 113, 114,
114, 114, 114, 114, 114, 115, 115, 117, 121, 123,
125, 127, 129, 129, 130, 131, 133, 135, 135, 139,
139, 139, 139, 139, 139, 139, 139, 139, 139)
subtype Rule is Natural;
subtype Nonterminal is Integer;

RuleLength : array (Rule range 0 .. 216) of Natural := (2,
0, 2, 0, 3, 0, 1, 1, 1,
5, 6, 0, 3, 0, 0, 2, 0,
0, 0, 6, 0, 0, 5, 4, 3,
0, 0, 3, 0, 3, 0, 3, 0,
0, 0, 7, 0, 3, 4, 3, 1,
0, 0, 5, 0, 0, 0, 7, 1,
0, 4, 1, 2, 0, 3, 0, 3,
0, 2, 0, 2, 0, 0, 5, 0,
5, 0, 0, 6, 0, 0, 5, 0,
4, 0, 6, 0, 4, 4, 0, 0,
8, 0, 0, 3, 0, 0, 7, 0,
1, 0, 2, 0, 0, 4, 0, 0,
3, 0, 0, 4, 0, 10, 0, 8,
0, 0, 8, 0, 6, 0, 6, 0,
0, 5, 0, 0, 3, 0, 3, 0,
3, 0, 4, 0, 4, 0, 3, 0,
3, 1, 1, 1, 2, 0, 0, 4,
0, 2, 1, 1, 1, 1, 1, 1,
3, 0, 0, 8, 3, 0, 4, 3,
2, 2, 3, 3, 3, 2, 2, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 2,
2, 2, 2, 2, 1, 0, 4, 0,
2, 1, 1, 1, 1, 1, 1, 1,
3, 0, 0, 8, 0, 4, 0, 4,
0, 4, 0, 3, 0, 3, 0, 4,
0, 4, 0, 4, 0, 3, 0, 3);
GetLHS_Rule: array (Rule range 0 .. 216) of Nonterminal := (-1,
-3,-2,-5,-4,-4,-6,-9,
-7,-10,-16,-12,-12,-18,-13,-13,
-19,-20,-14,-14,-22,-8,-21,-24,
-24,-27,-25,-28,-25,-29,-25,-30,
-31,-32,-25,-34,-25,-25,-17,-17,
-38,-39,-37,-41,-42,-43,-40,-40,
-44, -35, -35, -26, -26, -15, -48, -45,
-45, -46, -46, -47, -47, -49, -11, -50,
-11, -52, -53, -51, -51, -54, -23, -56,
-23, -61, -55, -62, -57, -63, -63, -67,
-64, -64, -68, -65, -70, -72, -69, -69,
-71, -71, -66, -66, -73, -58, -58, -74,
-59, -59, -75, -60, -77, -76, -84, -76,
-85, -66, -83, -88, -83, -91, -83, -83,
-93, -78, -78, -94, -92, -95, -92, -92,
-79, -79, -80, -80, -81, -81, -82, -82,
-96, -90, -90, -90, -89, -89, -97, -33,
-99, -33, -98, -98, -98, -98, -98, -98,
-98, -100, -101, -98, -98, -103, -98, -98,
-98, -98, -98, -98, -98, -98, -98, -102,
-102, -102, -104, -104, -104, -104, -104,
-105, -105, -105, -106, -106, -106, -36,
-36, -36, -36, -36, -36, -107, -109, -110,
-108, -87, -87, -87, -87, -87, -87,
-87, -111, -112, -87, -113, -87, -114, -87,
-115, -87, -116, -87, -117, -87, -118, -87,
-119, -87, -120, -87, -121, -87, -122, -87);
end Padi_Goto;
APPENDIX W. PACKAGE PSDL_SHIFT_REDUCE

package PSDL_Shift_Reduce is

  type Small_Integer is range -32_000 .. 32_000;

  type Shift_Reduce_Entry is record
    T : Small_Integer;
    Act : Small_Integer;
  end record;

  subtype Row is Integer range -1 .. Integer'Last;

  type Shift_Reduce_Array
    is array (Row range <>) of Shift_Reduce_Entry;

  Shift_Reduce_Matrix : constant Shift_Reduce_Array :=
    ( (-1,-1) -- Dummy Entry
      , (44,-3), (59,-3),(-1,-2)
      , (59, 8),(-1,-1000)
      , (62, 11),(-1,-1000)
      , (62, 12)
      , (-1,-4)
      , (-1,-8)
      , (-1,-7)
      , (-1,-8)
      , (-1,-1000)
      , (-1,-1000)
      , (-1,-1000)
      , (-1,-1000)
      , (-1,-1000)
    );
, (-1, -21)
-- state 13
, (51, 15), (-1, -1000)
-- state 14
, (51, 17), (-1, -1000)

-- state 15
, (29, 19), (-1, -13)
-- state 16
, (33, 21), (-1, -1000)

-- state 17
, (-1, -25)
-- state 18
, (33, 24), (-1, -1000)
-- state 19
, (-1, -11)

-- state 20
, (62, -14), (-1, -16)
-- state 21
, (13, 30), (62, 29)
, (-1, -1000)
-- state 22
, (-1, -9)
-- state 23
, (25, 37), (29, 33)
, (35, 34), (36, 39), (37, 38), (46, 35)
, (53, 36), (-1, -57)
-- state 24
, (13, 44), (-1, -76)

-- state 25
, (-1, -22)
-- state 26
, (-1, -41)
-- state 27
, (-1, -41)
-- state 28
, (-1, -20)

-- state 29
, (5, -44), (-1, -48)
-- state 30
, (62, 52), (-1, -1000)

-- state 31
, (-1, -64)
-- state 32
, (22, 54), (-1, -59)
-- state 33
, (-1, -26)

-- state 34
,(66, 105),(-1,-1000)
-- state 81
,(-1,-54)

-- state 82
,(4, 71),(-1,-27)
-- state 83
,(4, 71),(-1,-29)

-- state 84
,(4, 71),(-1,-31)
-- state 85
,(4, 71),(-1,-33)

-- state 86
,(4, 98),(-1,-37)
-- state 87
,(63, 108),(-1,-1000)

-- state 88
,(4, 98),(-1,-56)
-- state 89
,(4, 98),(-1,-52)

-- state 90
,(-1,-93)
-- state 91
,(-1,-96)
-- state 92
,(19, 112),(-1,-1000)

-- state 93
,(60, 115),(-1,-82)
-- state 94
,(24, 116),(-1,-1000)

-- state 95
,(-1,-73)
-- state 96
,(-1,-39)
-- state 97
,(-1,-42)
-- state 98
,(-1,-49)

-- state 99
,(-1,-10)
-- state 100
,(62, 119),(-1,-1000)
-- state 101
,(-1,-41)

-- state 102
,(-1,-63)
-- state 103
,(-1,-65)
  -- state 104
  ,( 44, 121),(-1,-1000)
  -- state 105
  ,(-1,-60)
  -- state 106
  ,( 34, 122),(-1,-1000)
  -- state 107
  ,( 31, 127)
  ,( 39, 123), ( 40, 126), ( 41, 124), ( 50, 125)
  ,(-1,-1000)
  -- state 108
  ,(-1,-181)
  -- state 109
  ,(-1,-38)
  -- state 110
  ,(-1,-41)
  -- state 111
  ,( 62, 73),(-1,-1000)
  -- state 112
  ,( 20, 130),(-1,-1000)
  -- state 113
  ,(-1,-74)
  -- state 114
  ,( 23, 132),(-1,-77)
  -- state 115
  ,( 62, 133)
  ,(-1,-1000)
  -- state 116
  ,(-1,-71)
  -- state 117
  ,( 62, 29),(-1,-1000)
  -- state 118
  ,( 62, 136),(-1,-1000)
  -- state 119
  ,(-1,-18)
  -- state 120
  ,( 4, 71)
  ,(-1,-46)
  -- state 121
  ,( 62, 139),(-1,-1000)
  -- state 122
  ,(-1,-34)
  -- state 123
  ,(-1,-176)
  -- state 124
  ,(-1,-177)
  -- state 125
  ,(-1,-178)
  -- state 126
, (-1, -179)

-- state 127
, (-1, -180)

-- state 128
, (4, 71), (-1, -94)

-- state 129
, (4, 98)
, (-1, -97)

-- state 130
, (-1, -99)

-- state 131
, (22, 54), (-1, -59)

-- state 132
, (62, 143), (-1, -1000)

-- state 133
, (-1, -83)

-- state 134
, (7, 145)
, (-1, -92)

-- state 135
, (-1, -43)

-- state 136
, (-1, -50)

-- state 137
, (51, 17)
, (-1, -1000)

-- state 138
, (6, 148), (-1, -1000)

-- state 139
, (-1, -67)

-- state 140
, (-1, -137)

-- state 141
, (44, 153), (-1, -1000)

-- state 142
, (-1, -75)

-- state 143
, (-1, -80)

-- state 144
, (7, -88), (19, -88)
, (21, -88), (23, -88), (57, -88), (60, -88)
, (-1, -85)

-- state 145
, (63, 108), (-1, -1000)

-- state 146
, (-1, -78)

-- state 147
, (-1, -19)

-- state 148
, (-1, -47)
-- state 149
, (33, 24), (-1, -1000)

-- state 150
, (4, 159), (-1, -35)

-- state 151
, (2, 167), (11, 160)
, (12, 161), (43, 171), (62, 165), (63, 162)
, (64, 163), (65, 164), (77, 170), (78, 169)
, (87, 172), (-1, -1000)

-- state 152
, (44, 173), (-1, -100)

-- state 153
, (62, 174), (-1, -1000)

-- state 154
, (7, 145), (-1, -92)

-- state 155
, (2, 176), (-1, -1000)

-- state 156
, (-1, -84)

-- state 157
, (-1, -91)

-- state 158
, (-1, -68)

-- state 159
, (-1, -135)

-- state 160
, (-1, -139)

-- state 161
, (-1, -140)

-- state 162
, (-1, -141)

-- state 163
, (-1, -142)

-- state 164
, (-1, -143)

-- state 165
, (5, -44)

-- state 166
, (8, -44), (-1, -144)

-- state 167
, (2, 167), (11, 160), (12, 161), (43, 171)
, (62, 165), (63, 162), (64, 163), (65, 164)
, (77, 170), (78, 169), (87, 172), (-1, -1000)

-- state 168
, (42, 194), (45, 181), (67, 180), (68, 182)
, (70, 183), (71, 184), (72, 185), (73, 186)
, (74, 187), (75, 188), (77, 189), (78, 190)
-- state 219
, (15, 226)
, (17, 227), (-1, -120)
-- state 220
, (47, 229), (-1, -122)

-- state 221
, (62, 133), (-1, -1000)
-- state 222
, (9, 232), (-1, -1000)

-- state 223
, (2, 233), (-1, -1000)
-- state 224
, (42, 194), (70, 183)
, (71, 184), (72, 185), (73, 186), (74, 187)
, (75, 188), (77, 189), (78, 190), (79, 191)
, (82, 192), (83, 193), (84, 195), (86, 200)
, (-1, -151)
-- state 225
, (47, 229), (-1, -122)
-- state 226
, (-1, -116)

-- state 227
, (-1, -1.8)
-- state 228
, (-1, -113)
-- state 229
, (63, 108), (-1, -1000)

-- state 230
, (28, 239), (-1, -124)
-- state 231
, (-1, -81)
-- state 232
, (62, 73)
, (-1, -90)
-- state 233
, (-1, -147)
-- state 234
, (28, 239), (-1, -124)

-- state 235
, (62, 73), (-1, -1000)
-- state 236
, (62, 73), (-1, -1000)

-- state 237
, (32, 246), (-1, -134)
-- state 238
, (16, 64), (-1, -53)

-- state 239
, (61, 249), (-1, -1000)
-- state 240
,(38, 250),(-1,-126)

-- state 241
,(3, 252),(-1,-1000)
-- state 242
,( -1,-137)
-- state 243
,(38, 250)
,(-1,-126)
-- state 244
,(4, 98),(-1,-117)
-- state 245
,(4, 98)
,(-1,-119)
-- state 246
,(2, 263), (11, 255), (12, 256)
,(43, 267), (62, 261), (63, 257), (64, 259)
,(65, 260), (77, 266), (78, 265), (87, 268)
,(-1,-1000)
-- state 247
,(16, 64),(-1,-53)
-- state 248
,(-1,-121)

-- state 249
,(63, 108),(-1,-1000)
-- state 250
,(18, 271),(-1,-1000)

-- state 251
,(37, 273),(-1,-128)
-- state 252
,(-1,-87)
-- state 253
,(3, 275)
,(4, 159),(-1,-1000)
-- state 254
,(37, 273),(-1,-128)

-- state 255
,(-1,-186)
-- state 256
,(-1,-187)
-- state 257
,(31,-181), (39,-181)
,(40,-181), (41,-181), (50,-191), (-1,-188)

-- state 258
,(-1,-189)
-- state 259
,(-1,-190)
-- state 260
,(-1,-191)
-- state 261

414
\[ (5, -44), \quad (8, -48), \quad (-1, -192) \]
\[
-- \text{ state } 262
\]
\[ (8, 277), \quad (-1, -1000) \]
\[
-- \text{ state } 263
\]
\[ (-1, 197) \]
\[
-- \text{ state } 264
\]
\[ (42, 194), \quad (45, 181), \quad (67, 180) \]
\[ (68, 182), \quad (70, 183), \quad (71, 184), \quad (72, 185) \]
\[ (73, 186), \quad (74, 187), \quad (75, 188), \quad (77, 189) \]
\[ (78, 190), \quad (79, 191), \quad (82, 192), \quad (83, 193) \]
\[ (84, 195), \quad (86, 283), \quad (-1, -133) \]
\[
-- \text{ state } 265
\]
\[ (-1, -203) \]
\[
-- \text{ state } 266
\]
\[ (-1, -205) \]
\[
-- \text{ state } 267
\]
\[ (-1, -213) \]
\[
-- \text{ state } 268
\]
\[ (-1, -215) \]
\[
-- \text{ state } 269
\]
\[ (-1, -114) \]
\[
-- \text{ state } 270
\]
\[ (16, 64), \quad (-1, -53) \]
\[
-- \text{ state } 271
\]
\[ (63, 108), \quad (-1, -1000) \]
\[
-- \text{ state } 272
\]
\[ (63, 108), \quad (-1, -1000) \]
\[
-- \text{ state } 273
\]
\[ (49, 291), \quad (-1, -1000) \]
\[
-- \text{ state } 274
\]
\[ (-1, -104) \]
\[
-- \text{ state } 275
\]
\[ (-1, -148) \]
\[
-- \text{ state } 276
\]
\[ (-1, -112) \]
\[
-- \text{ state } 277
\]
\[ (62, 293) \]
\[ (-1, -1000) \]
\[
-- \text{ state } 278
\]
\[ (2, 263), \quad (11, 255), \quad (12, 256) \]
\[ (43, 267), \quad (62, 261), \quad (63, 257), \quad (64, 259) \]
\[ (65, 260), \quad (77, 266), \quad (78, 265), \quad (87, 268) \]
\[ (-1, -1000) \]
\[
-- \text{ state } 279
\]
\[ (-1, -199) \]
\[
-- \text{ state } 280
\]
\[ (-1, -201) \]
\[
-- \text{ state } 281
\]
\[ (-1, -207) \]
-- state 282
,(-1,-209)
-- state 283
,(-1,-211)
-- state 284
,(2,263),(11,255)
,(12,256),(43,267),(62,261),(63,257)
,(64,259),(65,260),(77,266),(78,265)
,(87,268),(-1,-1000)
-- state 285
,(2,263),(11,255)
,(12,256),(43,267),(62,261),(63,257)
,(64,259),(65,260),(77,266),(78,265)
,(87,268),(-1,-1000)
-- state 286
,(2,263),(11,255)
,(12,256),(43,267),(62,261),(63,257)
,(64,259),(65,260),(77,266),(78,265)
,(87,268),(-1,-1000)
-- state 287
,(2,263),(11,255)
,(12,256),(43,267),(62,261),(63,257)
,(64,259),(65,260),(77,266),(78,265)
,(87,268),(-1,-1000)
-- state 288
,(-1,-123)
-- state 289
,(16,64)
,(-1,-53)
-- state 290
,(16,64),(-1,-53)
-- state 291
,(56,306)
,(-1,-1000)
-- state 292
,(26,311),(46,310),(48,307)
,(52,308),(54,309),(-1,-102)
-- state 293
,(3,-193)
,(4,-193),(16,-193),(22,-193),(24,-193)
,(26,-193),(28,-193),(37,-193),(38,-193)
,(42,-193),(44,-193),(45,-193),(46,-193)
,(47,-193),(48,-193),(52,-193),(54,-193)
,(67,-193),(68,-193),(70,-193),(71,-193)
,(72,-193),(73,-193),(74,-193),(75,-193)
,(77,-193),(78,-193),(79,-193),(82,-193)
,(83,-193),(84,-193),(86,-193),(-1,-194)
-- state 294
,(3,314),(42,194),(45,191),(67,180)
,(68,182),(70,183),(71,184),(72,185)
,(73,186),(74,187),(75,188),(77,189)
,(78,190),(79,191),(82,192),(83,193)
,(84,195),(86,263),(-1,-1000)
-- state 312
,( 62, 322),(-1,-1000)

-- state 313
,( 2, 323),(-1,-1000)
-- state 314
,(-1,-198)
-- state 315
,(-1,-198)

-- state 316
,(-1,-1000)

-- state 317
,(-1,-1000)

-- state 318
,(-1,-1000)

-- state 319
,(-1,-1000)

-- state 320
,(-1,-1000)

-- state 321
,(-1,-1000)

-- state 322
,(-1,-1000)

-- state 323
,(-1,-1000)

-- state 324
,(-1,-1000)

-- state 325
,(-1,-1000)

-- state 326
,(-1,-1000)

-- state 327
,(-1,-1000)

-- state 328
,(-1,-1000)

-- state 329
,(-1,-1000)

-- state 330
,(-1,-1000)

-- state 331
,(-1,-1000)
-- state 332
( 2, 263), (11, 255), (12, 256)
(43, 267), (62, 261), (63, 257), (64, 259)
(65, 260), (77, 266), (78, 265), (87, 268)
(-1,-1000)
-- state 333
( 2, 263), (11, 255), (12, 256)
(43, 267), (62, 261), (63, 257), (64, 259)
(65, 260), (77, 266), (78, 265), (87, 268)
(-1,-1000)
-- state 334
(-1,-109)
-- state 335
(-1,-111)
-- state 336
(-1,-182)
-- state 337
(-1,-196)
-- state 338
( 42, 194), (45, 181), (67, 180)
( 68, 182), (70, 183), (71, 184), (72, 185)
( 73, 186), (74, 187), (75, 188), (77, 189)
( 78, 190), (79, 191), (82, 192), (83, 193)
( 84, 195), (86, 283), (-1,-185)
-- state 339
( 16, 64)
( 42, 194), (45, 181), (67, 180), (68, 182)
( 70, 183), (71, 184), (72, 185), (73, 186)
( 74, 187), (75, 188), (77, 189), (78, 190)
( 79, 191), (82, 192), (83, 193), (84, 195)
( 86, 283), (-1,-53)
-- state 340
( 2, 263), (11, 255)
( 12, 256), (43, 267), (62, 261), (63, 257)
( 64, 259), (65, 260), (77, 266), (78, 265)
( 87, 268), (-1,-1000)
-- state 341
(-1,-107)
-- state 342
( 42, 194)
( 45, 181), (67, 180), (68, 182), (70, 183)
( 71, 184), (72, 185), (73, 186), (74, 187)
( 75, 188), (77, 189), (78, 190), (79, 191)
( 82, 192), (83, 193), (84, 195), (86, 283)
(-1,-183)
);
-- The offset vector
SHIFT_REDUCE_OFFSET : array (0..342) of Integer :=
( 1, 2, 4, 7, 8, 11, 12, 13, 15, 17,
18, 19, 20, 22, 24, 26, 28, 29, 31, 32,
34, 37, 38, 46, 48, 49, 50, 51, 52, 54,
56, 57, 59, 60, 61, 62, 63, 64, 66, 67,
69, 71, 73, 75, 77, 78, 80, 81, 83, 85,
88, 90, 91, 92, 94, 96, 97, 98, 99, 100,
102, 104, 106, 107, 109, 110, 112, 114, 115, 116,
118, 119, 122, 123, 125, 127, 128, 130, 132, 133,
135, 136, 138, 140, 142, 144, 146, 148, 150, 152,
153, 154, 156, 158, 160, 161, 162, 163, 164, 165,
167, 168, 169, 170, 172, 173, 175, 181, 182, 183,
184, 186, 188, 189, 191, 193, 194, 196, 198, 199,
201, 203, 204, 205, 206, 207, 208, 209, 211, 213,
214, 216, 218, 219, 221, 222, 223, 225, 227, 228,
229, 231, 232, 233, 241, 243, 244, 245, 246, 248,
250, 262, 264, 266, 268, 270, 271, 272, 273, 274,
275, 276, 277, 278, 279, 282, 284, 296, 314, 326,
338, 350, 362, 364, 365, 367, 369, 381, 383, 402,
403, 404, 405, 406, 408, 409, 410, 411, 412,
413, 414, 415, 416, 417, 418, 419, 431, 443, 455,
467, 473, 479, 480, 481, 482, 484, 486, 488, 489,
507, 538, 539, 551, 560, 562, 564, 565, 567, 570,
572, 574, 576, 578, 593, 595, 596, 597, 598, 600,
602, 603, 605, 606, 608, 610, 612, 614, 616, 618,
620, 622, 623, 625, 627, 629, 641, 643, 644, 646,
648, 650, 651, 654, 656, 657, 658, 664, 665, 666,
667, 670, 672, 673, 691, 692, 693, 694, 695, 696,
698, 700, 702, 704, 705, 706, 707, 709, 721, 722,
723, 724, 725, 726, 738, 750, 762, 774, 775, 777,
779, 781, 787, 820, 839, 851, 863, 875, 887, 899,
905, 911, 912, 913, 914, 915, 916, 917, 918, 919,
920, 922, 924, 926, 927, 942, 951, 957, 959, 960,
962, 963, 964, 965, 968, 970, 972, 973, 974, 976,
978, 981, 993, 1005, 1006, 1007, 1008, 1009, 1027, 1046,
1058, 1059);
end Psdl_Shift_Reduce;
APPENDIX X. PACKAGE PSDL_TOKENS

with Psdl_Concrete_Type_Pkg;
use Psdl_Concrete_Type_Pkg;
package Psdl_Tokens is

    type TOKENCATEGORY_TYPE is (INTEGER_LITERAL,
                              PSDL_ID_STRING,
                              EXPRESSION_STRING,
                              TYPE_NAME_STRING,
                              TYPE_DECLARATION_STRING,
                              TIME_STRING,
                              TIMER_OP_ID_STRING,
                              NO_VALUE);

type YYStype (Token_Category : TOKENCATEGORY_TYPE := NO_VALUE) is
    record
        case Token_Category is
            when INTEGER_LITERAL =>
                Integer_Value : INTEGER;
            when PSDL_ID_STRING =>
                Psdl_Id_Value : Psdl_Id;
            when TYPE_NAME_STRING =>
                TypeName_Value : TypeName;
            when TYPE_DECLARATION_STRING =>
                Type_Declaration_Value : Type_Declaration;
            when EXPRESSION_STRING =>
                Expression_Value : Expression;
            when TIME_STRING =>
                Time_Value : Millisec;
            when TIMER_OP_ID_STRING =>
                Timer_Op_Id_Value : Timer_Op_Id;
            when NO_VALUE =>
                White_Space : Text := Empty_Text;
        end case;
    end record;

    YSYLVal, YYVal : YYStype;

    type Token is
        (End_Of_Input, Error, '(' , ')' ,
         ',', '[', ']' ,
         ';', '.', '|');
Syntax_Error : exception;

end Psdl_Tokens;
INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center  
   Cameron Station  
   Alexandria, VA 22304-6145  

2. Dudley Knox Library  
   Code 52  
   Naval Postgraduate School  
   Monterey, CA 93943  

3. Computer Science Department  
   Code CS  
   Naval Postgraduate School  
   Monterey, CA 93943  

4. Office of the Assistant Secretary of the Navy  
   Research Development and Acquisition  
   Department of the Navy  
   Attn: Mr. Gerald A. Cann  
   Washington, DC 20380-1000  

5. Office of the Chief of Naval Operations  
   OP-094  
   Department of the Navy  
   Attn: VADM J. O. Tuttle, USN  
   Washington, DC 20301-3040  

6. Director of Defense Information  
   Office of the Assistant Secretary of Defense  
   (Command, Control, Communications, & Intelligence)  
   Attn: Mr. Paul Strassmann  
   Washington, DC 20301-0208  

7. Center for Naval Analysis  
   4401 Ford Avenue  
   Alexandria, VA 22302-0268  

8. Director of Research Administration  
   Attn: Prof. Howard  
   Code 08Hk  
   Naval Postgraduate School  
   Monterey, CA 93943  

423
<table>
<thead>
<tr>
<th>No.</th>
<th>Name and Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.</td>
<td>Chairman, Code CS&lt;br&gt;Computer Science Department&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943-5100</td>
</tr>
<tr>
<td>10.</td>
<td>Prof. Luqi, Code CSLq&lt;br&gt;Computer Science Department&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943</td>
</tr>
<tr>
<td>11.</td>
<td>Prof. Valdis Berzins, Code CSBe,&lt;br&gt;Computer Science Department&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943</td>
</tr>
<tr>
<td>12.</td>
<td>Chief of Naval Research&lt;br&gt;800 N. Quincy Street&lt;br&gt;Arlington, VA 22217</td>
</tr>
<tr>
<td>13.</td>
<td>Director, Ada Joint Program Office&lt;br&gt;OUSDRE (R&amp;AT)&lt;br&gt;Room 3E114, The Pentagon&lt;br&gt;Attn: Dr. John P. Solomond&lt;br&gt;Washington, DC 20301-0208</td>
</tr>
<tr>
<td>14.</td>
<td>Carnegie Mellon University&lt;br&gt;Software Engineering Institute&lt;br&gt;Attn: Dr. Dan Berry&lt;br&gt;Pittsburgh, PA 15260</td>
</tr>
<tr>
<td>15.</td>
<td>Office of Naval Technology (ONT)&lt;br&gt;Code 227&lt;br&gt;Attn: Dr. Elizabeth Wald&lt;br&gt;800 N. Quincy St.&lt;br&gt;Arlington, VA 22217-5000</td>
</tr>
<tr>
<td>16.</td>
<td>Defense Advanced Research Projects Agency (DARPA)&lt;br&gt;Integrated Strategic Technology Office (ISTO)&lt;br&gt;Attn: Dr. B. Boehm&lt;br&gt;1400 Wilson Boulevard&lt;br&gt;Arlington, VA 22209-2308</td>
</tr>
<tr>
<td>17.</td>
<td>Defense Advanced Research Projects Agency (DARPA)&lt;br&gt;ISTO&lt;br&gt;1400 Wilson Boulevard&lt;br&gt;Attn: LCol Eric Mattala&lt;br&gt;Arlington, VA 2209-2308</td>
</tr>
</tbody>
</table>
18. Defense Advanced Research Projects Agency (DARPA)
   Director, Tactical Technology Office
   1400 Wilson Boulevard
   Arlington, VA 2209-2308

19. Attn: Dr. Charles Harland
   Computer Science
   Department of the Air Force
   Bolling Air Force Base, DC 20332-6448

20. Chief of Naval Operations
   Attn: Dr. R. M. Carroll (OP-01B2)
   Washington, DC 20350

21. Dr. Robert M. Balzer
   USC-Information Sciences Institute
   4676 Admiralty Way
   Suite 1001
   Marina del Ray, CA 90292-6695

22. Dr. Ted Lewis
   OR State University
   Computer Science Department
   Corvallis, OR 97331

   12710 Research Boulevard, Suite 301
   Attn: Dr. R. T. Yeh
   Austin, TX 78759

24. Kestrel Institute
   Attn: Dr. C. Green
   1801 Page Mill Road
   Palo Alto, CA 94304

25. National Science Foundation
   Division of Computer and Computation Research
   Attn: K. C. Tai
   Washington, DC 20550

26. Commander Space and Naval Warfare Systems Command
   SPAWAR 3212
   Department of the Navy
   Attn: Cdr M. Romeo
   Washington, DC 20363-5100
27. Naval Ocean Systems Center
   Attn: Linwood Sutton, Code 423
   San Diego, CA 92152-5000

28. Office of Naval Research
   Computer Science Division, Code 1133
   Attn: Dr. Gary Koob
   800 N. Quincy Street
   Arlington, VA 22217-5000

29. Commander, Naval Sea Systems Command (PMS-4123H)
   Attn: William Wilder
   Washington, DC 20380-1000

30. New Jersey Institute of Technology
    Computer Science Department
    Attn: Dr. Peter Ng
    Newark, NJ 07102

31. Office of Naval Research
    Computer Science Division, Code 1133
    Attn: Dr. A. M. Van Tilborg
    800 N. Quincy Street
    Arlington, VA 22217-5000

32. Office of Naval Research
    Computer Science Division, Code 1133
    Attn: Dr. R. Wachter
    800 N. Quincy Street
    Arlington, VA 22217-5000

33. OR Graduate Center
    Portland (Beaverton)
    Attn: Dr. R. Kieburtz
    Portland, OR 97005

34. Santa Clara University
    Department of Electrical Engineering and
    Computer Science
    Attn: Dr. M. Ketabchi
    Santa Clara, CA 95053

35. Software Group, MCC
    9430 Research Boulevard
    Attn: Dr. L. Belady
    Austin, TX 78759
36. University of CA at Berkeley
   Department of Electrical Engineering and
   Computer Science
   Computer Science Division
   Attn: Dr. C.V. Ramamoorthy
   Berkeley, CA 90024

37. University of CA at Irvine
   Department of Computer and Information Science
   Attn: Dr. Nancy Leveson
   Irvine, CA 92717

38. Chief of Naval Operations
   Attn: Dr. Earl Chavis (OP-16T)
   Washington, DC 20350

39. Office of the Chief of Naval Operations
   Attn: Dr. John Davis (OP-094H)
   Washington, DC 20350-2000

40. University of Illinois
    Department of Computer Science
    Attn: Dr. Jane W. S. Liu
    Urbana Champaign, IL 61801

41. University of MD
    College of Business Management
    Tydings Hall, Room 0137
    Attn: Dr. Alan Hevner
    College Park, MD 20742

42. University of MD
    Computer Science Department
    Attn: Dr. N. Roussapoulos
    College Park, MD 20742

43. University of Massachusetts
    Department of Computer and Information Science
    Attn: Dr. John A. Stankovic
    Amherst, MA 01003

44. University of Pittsburgh
    Department of Computer Science
    Attn: Dr. Alfs Berztiss
    Pittsburgh, PA 15260
45. University of TX at Austin
   Computer Science Department
   Attn: Dr. Al Mok
   Austin, TX 78712

46. Commander, Naval Surface Warfare Center,
   Code U-33
   Attn: Dr. Philip Hwang
   10901 New Hampshire Avenue
   Silver Spring, MD 20903-5000

47. Attn: George Sumiall
   US Army Headquarters
   CECOM
   AMSEL-RD-SE-AST-SE
   Fort Monmouth, NJ 07703-5000

48. Attn: Joel Trimble
   1211 South Fern Street, C107
   Arlington, VA 22202

49. United States Laboratory Command
   Army Research Office
   Attn: Dr. David Hislop
   P. O. Box 12211
   Research Triangle Park, NC 27709-2211

50. George Mason University
    Computer Science Department
    Attn: Dr. David Rine
    Fairfax, VA 22030-4444

51. Hewlett Packard Research Laboratory
    Mail Stop 321
    1501 Page Mill Road
    Attn: Dr. Martin Griss
    Palo Alto, CA 94304

52. Carnegie Mellon University
    SEI
    Attn: Dr. Mario Barbacci
    Pittsburgh, PA 15213
53. Persisten: Data Systems
   75 W. Chapel Ridge Road
   Attn: Dr. John Nester
   Pittsburgh, PA 15238

54. Sun Microsystems Inc.
   MS MTV100-01
   Silicon Valley Government District
   1842 N. Shoreline Boulevard
   Attn: Vice President c/o Don Chandler
   Mountain View, CA 94043

55. Turkish Embassy
   Office of the Defense Attache
   2202 Mass. Ave., NW
   Washington, DC, 20008-2876

56. Deniz Kuvvetleri K.ligi,
    Teknik Baskanlik,
    06100 Bakanliklar, ANKARA
    TÜRKiYE

57. Deniz Harp Okulu K.ligi,
    Kütüphane
    Tuzla, İSTANBUL
    TÜRKiYE

58. Stuleyman Bayramoğlu
    19 Mayıs Mah.
    Alacali Sok. 17/1,
    06290 İncirli, ANKARA
    TÜRKiYE