An architectural model for software component search

Nguyen, Doan Han
Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/31350

Downloaded from NPS Archive: Calhoun
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
An important problem in software development process is to make better use of software libraries by improving the search and retrieval process, that is, by making it easier to find the few components you may want among the many you do not want. The problem with the current production approaches is that they do not consider the behavior of components as a part of the retrieval process. As the result, it is impossible to obtain high recall and precision. In contrast, research approaches using syntactic and specification can be used to improve upon recall and precision. However, these approaches require a lot more computational effort. Without a library structure to support a retrieval process, they would be impractical. This dissertation concentrates on two major themes. First, how to provide efficient and effective retrieval capabilities and an interactive friendly interface to support users to search for software components. Second, how to construct a library that can assist the librarian with cataloging software components and help to facilitate the search process. The first prototype has been implemented to verify the proposed ideas. Several studies have been performed to measure the system performance. The result confirms and strongly supports the proposed ideas.
AN ARCHITECTURAL MODEL FOR SOFTWARE COMPONENT SEARCH

Nguyen Doan Han
B.S., University of California, Davis., 1982
M.S., California State University, Sacramento, 1991

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1995

Author: Nguyen Doan Han

Approved by:

Luqi
Professor of Computer Science
Dissertation Supervisor

Ted Lewis
Chairman,
Department of Computer Science

Valdis Berzins
Professor of Computer Science

Dan Boger
Professor and Chairman
of Command and Control

Le Ngoc Ly
Research Associate Professor
Department of Oceanography

Approved by:

Ted Lewis, Chairman, Department of Computer Science

Maurice Weir, Associate Provost for Instruction
ABSTRACT

An important problem in software development process is to make better use of software libraries by improving the search and retrieval process, that is, by making it easier to find the few components you may want among the many you do not want. The problem with the current production approaches is that they do not consider the behavior of components as a part of the retrieval process. As the result, it is impossible to obtain high recall and precision. In contrast, research approaches using syntactic and specification can be used to improve upon recall and precision. However, these approaches require a lot more computational effort. Without a library structure to support a retrieval process, they would be impractical. This dissertation concentrates on two themes. First, how to provide efficient and effective retrieval capabilities and an interactive friendly interface to support users to search for software components. Second, how to construct a library that can assist the librarian with cataloging software components and help to facilitate the search process. The first prototype has been implemented to verify the proposed ideas. Several studies have been performed in to measure the system performance. The result confirms and strongly supports the proposed ideas.
# TABLE OF CONTENTS

I. INTRODUCTION ................................................................. 1

II. BACKGROUND AND RELATED WORK .................................... 5
   A. CLASSICAL APPROACHES ............................................. 5
   B. THE FACET APPROACH .............................................. 6
   C. AI APPROACHES ..................................................... 6
   D. SPECIFICATION-BASED APPROACHES ............................... 6
   E. THE CAPS APPROACH ................................................. 8
   F. DISCUSSION ........................................................... 12

III. ARCHITECTURE OF THE SEARCH PROCESS ......................... 13

IV. SYNTACTIC FILTERING ..................................................... 17
   A. KEYWORD MATCHING ............................................... 17
   B. SIGNATURE MATCHING ............................................. 18
   C. PROFILE MATCHING ................................................ 22
      1. Software Base Partitioning with Profiles .................... 25
      2. Retrieving Components with Profile Matching ............... 30
   D. SIGNATURE MATCHING ALGORITHM ............................... 31

V. SEMANTIC FILTERING ....................................................... 37
   A. ORDERING SEMANTIC MATCHES ................................... 38

VI. IMPLEMENTATION ........................................................ 43
   A. CHOICE OF LANGUAGES AND SUPPORT SYSTEMS ............... 43
   B. QUERY SUBMISSION AND PROFILE GENERATION ................. 44
   C. RETRIEVAL OF CANDIDATE SOFTWARE COMPONENTS ........... 46
3. Loading of OBJ3 Environment ........................................... 90

D. SUGGESTIONS FOR FUTURE RESEARCH .............................. 90

1. Environment for Evaluation of Test Cases in Queries ............. 90

2. The Choice for Rank Functions ...................................... 91

3. Possible Improvements for the Signature Matching Algorithm .... 91

4. Experimentation with other Components from other Libraries ... 92

5. Exploring other Specification Languages ............................ 93

E. SUGGESTION CHANGES TO GET PRODUCTION QUALITY ......... 93

REFERENCES .................................................................. 95

APPENDIX A - BACKGROUND INFORMATION ............................ 99

A. OBJ3 AND ALGEBRAIC SPECIFICATION ............................... 99

B. GROUND EQUATION .................................................... 99

C. TERM REWRITING ...................................................... 100

D. OPERATIONS ............................................................ 100

E. ACCESSOR AND CONTRUCTOR OPERATIONS .................... 101

APPENDIX B - OBJ3 COMPONENTS ....................................... 103

APPENDIX C - ADA SOURCE CODE ..................................... 121

APPENDIX D - RUNOBJ AND SCS BUILD FILES ....................... 197

APPENDIX E - A PROOF FOR THEOREM 1 ............................ 199

INITIAL DISTRIBUTION LIST ............................................. 201
ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Professors Luqi and Berzins whose teaching, support, guidance, and encouragement assisted me during my study at NPS. Without their help, this work would have been impossible to accomplish. I would like to thank Professor Goguen for his support and guidance. I also would like to thank Professor Zhang for his advise, support and encouragement. Special thanks and deep gratitude are also due to Professor Shing for teaching and advising.

I wish to thank Professors Lewis, Boger, and Lely for their support and advise with many valuable comments to improve the quality of the dissertation.

I also would thank many good friends Ibrahim, Tuan, and Vincent for their help, support and encouragement during the difficult times. Their help is greatly appreciated.

I would like to thank the management at the Department of U.S. Air Force at McClellan AFB for supporting me for this program. Without their help, I would not have been able to get to this point. Their help and encouragement is greatly appreciated.

I must acknowledge the unfailing and unconditional support I have from my wife, Yvonne and my daughter, Deanna. They have sacrificed a lot for me without asking much. I also would like to thank my parents and parents’s in-law for the caring of my wife and daughter while I was away from home.

Finally, I thank God for giving me the courage and health in order that I can complete this program. This obtained knowledge definitely will be put to good use to our society.
I. INTRODUCTION

Billions of dollars are spent each year on computer software. Much of this effort is spent on creating and testing new source code. In order to save money, increase productivity, and improve reliability, the Department of Defense is constructing repositories of reusable software components that can be used across applications. Devising an effective way to retrieve components from software libraries, referred to as the Software Component Search problem (or simply, the Search Problem), is of increasing importance for many applications. For example, rapid prototyping has been used to validate and refine system requirements, and check the consistency of proposed designs, before undertaking a full implementation. Automated retrieval of relevant reusable software components is important for this area.

The problem with the current production approaches is that they do not consider the behavior of components as part of the retrieval process. As the result, it is impossible to obtain high recall and precision. In contrast, the research approaches using syntactic and specification can be used to improve upon recall and precision. However, they require a lot more computational effort. Without a library structure to support a retrieval process, these approaches would be impractical. This dissertation concentrates on two themes. First, how to provide efficient and effective retrieval capabilities and an interactive friendly interface to support users to search for components. Second, how to construct a library that can assist the librarian with cataloging components and help to facilitate the search process.
In practice, there may be no component in the software base that does exactly what is wanted, but there may be some component that can be easily modified to do the job. This implies that given a query, we do not just seek components that match it exactly, but instead we seek a set of approximate candidates, ordered by how well they match the query. In other words, the choice set should consist of ranked partial matches.

These considerations motivate the following requirements for solutions to the search problem:

1. The retrieval process should be automated, since a software library may contain thousands of components, so that it would be virtually impossible for a human being to identify the desired component(s) quickly and accurately.
2. The retrieval process should be accurate, in the sense that the choice set should include the closest match, if there is one.
3. The search process should be effective, in that it should be fast, and the choice set should not be too large.
4. The user interface should allow flexible, easy query formulation, and should provide helpful feedback to the user.

Professor Luqi [19, 22] has suggested associating a semantic specification with each module in the software base to support retrieval against semantic queries, as has Professor Goguen [7]. This idea has been shown viable in work reported in [21, 32, 33], where the algebraic specification language OBJ3 [6, 10, 16] was used in software search experiments in the context of the Computer Aided Prototyping System (CAPS) project. Recent work [14] has carried this further by showing how to treat generic modules, how to use semantic information in a limited efficient way, and how to rank candidate modules by their likelihood of success (see [33] for discussion of an earlier ranking method). Ranking modules by how well they satisfy the query makes the search process
more robust, that is, better able to tolerate errors in the query and in how components are classified. We must expect such errors in practice.

Given a query $Q$ and a component $M$ with corresponding specification $T_M$, then $M$ is a correct answer for the query $Q$ if there is a translation of the syntax of $Q$ into the syntax of $T_M$ such that each translated equation from $Q$ is a consequence\(^1\) of $T_M$. Finding a correct answer in this sense is really a theorem proving task that could take too much time to be practical if not limited. However, finding candidates that satisfy adequate necessary conditions for being a correct answer is a practical goal. This will allow many irrelevant candidates to be rejected, resulting in a more focused search and raising the confidence in the components found.

A brief summary of related work is given in Chapter II. An overview of our software architecture for automated component retrieval is given in Chapter III. Background information on algebraic specification, including basic definitions, is given in Appendix A. Chapters IV and V describe syntactic and semantic filtering, respectively. Examples illustrating the search process are given in Chapter VI, while Chapter VII gives an overview of the structure of the design of the prototype. Chapter VIII provides the experimental results. Finally, Chapter IX summarizes the dissertation and proposes directions for further work.

\(^1\) Such consequences may be either equational consequences or inductive consequences, depending of whether a "loose" or an "initial" semantics is given to the specification $T_M$ (these terms are explained in [26]). An advantage of our approach is that it is insensitive to which of these semantics is assumed.
II. BACKGROUND AND RELATED WORK

Previous work on reusable software component retrieval can be classified as classical, facet oriented, AI, or pure specification. More information may be found in [5, 4, 17, 19, 25, 29]. We do not attempt to survey the entire relevant literature here, but instead we first describe some publications that seem most closely related to the present work.

A. CLASSICAL APPROACHES

The most classical approach to retrieval is to classify items by keywords, and then search for items that have certain given keywords [23]. Experience shows that this works poorly for retrieving software components from even moderately large libraries. One problem is that the user must be familiar with both the classification scheme and the particular library. Also, it is very difficult to get both high precision and high recall\(^2\). This suggests that for ranked filtering, it would be most appropriate to use a small number of keywords.

Another classical approach is browsing. Browsing systems depend on links among the items to be searched, and upon the user following those links to find the desired item. Experience shows that browsing through large structures can be very frustrating and time-consuming. Often, existing links seem random or even perverse, while the links you really want may not be present.

---

\(^2\) Precision and recall are classical terms from information retrieval. Let \(Q\) be the set of items that should be returned in answer to the query and let \(R\) be the choice set actually returned. Then the precision of \(R\) is defined to be \(|R \cap Q|/|Q|\) while the recall of \(R\) is \(|R \cap Q|/|R|\).
B. THE FACET APPROACH

Prieto-Diaz [30] has proposed using facets, which are groups of related terms in a subject area. For example, a facet to describe the functions performed by components might use terms chosen from *find, compare, sort, update, send, receive,*..... A scheme is developed in [30] to describe Unix components using four facets: the function performed by the component, the objects that are manipulated, the data structure used, and the system to which the function belongs. This provides a better description of Unix components than a pure keyword approach due to its well-structured. However, it still relies on an informal description of components, using a limited set of facets and terms. Facet is also suffering the same problem as Keyword approach. Namely, it also tends to miss potentially useful components because the people who classify the components in the library cannot anticipate all potential applications of each component.

C. AI APPROACHES

AI-based work includes [3,27] and some recent work by Henninger [18], which uses a knowledge-base and statistical information to retrieve reusable components, based on keyword search from texts describing the components. However, because the characterization of the component behavior is completely informal, the behavior is unpredictable.

D. SPECIFICATION-BASED APPROACHES

Recent work using semantics for software component retrieval is reported in [19, 25]. The primary aim is to check that retrieved components yield the behavior specified in the user's query, therefore increasing the precision of retrieval. Using formal
specifications as search keys has two main problems. The first problem is practical: not all users are sophisticated enough to write formal specifications, much less correct ones. The second problem is that semantic matching is very time consuming, because some form of theorem proving must be done.

The Venari project at Carnegie Mellon University, headed by Prof. Jeannette Wing, is devoted to retrieving components from software libraries, and has produced a number of interesting publications. Here we will not discuss their work on transactions and other infrastructural support for retrieval, but only their work on the search problem.

Rollins and Wing [31] discuss signature matching for retrieving higher-order functions from an MetaLanguage (ML) library, using λ-Prolog for matching user queries to component signatures. λ-Prolog is used to implement various matching modulo theories, in order to support (what we call) partial matches. They also use λ-Prolog to check simple pre- and post-conditions for ML functions. Although this paper demonstrates that higher-order logic is useful for such applications, we feel that higher-order logic is more powerful and expressive than necessary, and that higher-order logic tools like λ-Prolog are too inefficient. Of course, a higher-order language like ML requires the use of higher-order types, but these are first-order expressions, so that first-order matching could be used. Rollins and Wing point out that equational reasoning

---

3. This name is from the Latin verb “to hunt”.
4. For these authors, the word “signature” refers to the rank of a higher-order function, rather than to the syntactic specification of a software module, as in the algebraic tradition.
5. The Venari project uses the term “partial match” in a more restricted sense than we do; their term corresponding to our “partial match” is “relaxed match”.
could dramatically increase precision, and they also discuss the possibility of specification matching.

Zaremski and Wing [34] extend this work. First, they consider signatures in two different senses, as the rank of a function, and as the interface of a module; the second sense involves search and retrieval of modules, not just of functions. Second, they consider a wider variety of matching procedures and their combinations, although some of these are needed only because of the awkwardness of the higher-order encoding of operation ranks (e.g., uncurrying). Third, they implemented their matching procedures in ML, experimented with retrieving functions from actual ML libraries, and presented some interesting statistics on these experiments.

In more recent work, Zaremski and Wing [35] focus on specification matching, using the Larch/ML interface language to express pre- and post-conditions in first order logic, and the Larch prover to verify that candidate components satisfy these conditions. Various senses of matching are defined, but neither ranking nor partial semantic matching are considered. This approach has not resulted in a practical automated method for specification based retrieval.

E. THE CAPS APPROACH

The CAPS project at the Naval Postgraduate School, headed by Professors Luqi and Valdis Berzins, supports rapid prototyping for hard real time embedded systems. CAPS consists of an integrated set of software tools that help design, translate and execute prototypes. These tools include an execution support system, a syntax directed graphical editor, an evolution control system, a change merge facility, automatic
generators for schedule and control code, and facilities to support retrieving reusable components from a software base. The execution support system includes dynamic and static schedulers, a translator, and a debugger.

PSDL is the Prototyping Description Language of CAPS [20]; it is used to specify both prototypes and production software, and has data flow like semantics. PSDL programs have two kinds of objects, corresponding to abstract data types and abstract state machines; they localize the information for analyzing, executing and reusing independent objects. Executable Ada modules can be associated to atomic PSDL objects, and then CAPS can automatically generate "glue" code that composes these modules into a system having the structure described by PSDL. This generated code includes a schedule and tests for all real time constraints that have been declared. The system can then be compiled, executed, and tested. Error messages are produced during execution if constraints are violated. Figure 1 illustrates a prototyping life-cycle. It shows two places where component search can be used in such a life-cycle: in constructing a prototype system and in constructing a production system.
The remainder of this subsection concentrates on work done in the CAPS project on retrieving software components. The use of specifications in retrieving software components was originally suggested by Professor Luqi [19]. This suggestion was refined in later work, including [32] and Steigerwald's Ph.D. thesis [33]. In [33] it is assumed that each component has a fully expanded\(^6\) algebraic specification written in OBJ3 [6, 10, 16], and that the user's query is also a fully expanded algebraic specification that the desired component should satisfy. A Prolog program was written using symbolic representations of signatures to find syntactic matches between the signature of the query and the signatures of components. For each match found, a semantic validation was done by evaluating patterns that represent the functions in the

---

6. This means that the results of any module expressions inside a module are substituted into the module.
signature, first in the query specification\(^7\), and then (after translation) in the specifications of the matched components; the results of these two evaluations are compared to determine the quality of each match. The approach developed in this series of papers is the inspiration for the approach taken in the present dissertation.

The system described in [33] has certain technical limitations. Its semantic basis is not well developed. Also, evaluating patterns with variables gives limited information about the semantic satisfaction of a syntactic match. In addition, since patterns can involve variables that may or may not be eliminated by rewriting, depending on syntactic peculiarities of the equations, it seems possible to have semantically equivalent specifications for which pattern evaluation would give conflicting answers, so that the match in question will appear not to satisfy its semantic requirements even though it really does. In addition, the approach is limited to total syntactic matches and to unparameterized components.

Özdemir's master's thesis [28] describes a component retrieval system for the CAPS software base that uses keyword search and a browser. Both of these use PSDL for queries and for components. Özdemir also provides a graphical user interface and facilities for integrating retrieved components into prototype systems, including techniques for transforming retrieved modules. A better developed version of these ideas appears in the master's thesis of Dolgoff [2]. This work is including retrieval of generic modules and handling of subsort matching.

---

\(^7\) These patterns are terms that involve those functions, plus some variables, constants, and constructors, such that all other functional expressions are instances.
F. DISCUSSION

In comparing the approach of this dissertation with other approaches, the following points may be noted: (1) Our approach focuses on comparing formal specifications of components using ground equation test cases as queries. (2) Users do not need to deal with formal specification notation, but instead can express queries in a standard programming notation, which is automatically translated into algebraic notation. (3) We seek to achieve both efficiency and effectiveness by imposing a series of increasingly stringent filters that use both syntactic and partial semantic information about components. (4) A rank is provided on components in the choice set, measuring how well they fit the user's query. (5) We allow generic modules in the software base. (6) Our approach not only focuses on the problem of retrieving components, but also deals with structuring the software base to facilitate search. (7) Users can give selection criteria to control the search and display of retrieved components. (8) Besides returning the ranked components, we also report information to help the user reformulate the query in case no suitable component was found. (See Figure 2 in Chapter III)
III. ARCHITECTURE OF THE SEARCH PROCESS

This dissertation proposes an approach to the automated retrieval of reusable software components from a software base, continuing work reported in [14, 19, 21, 32, 33, 7]. The approach is based on the following assumptions: (1) the components in the software base are written in a modern programming language, e.g., Ada, that has strong typing, can package together a number of operations over common data representations, and allows generic modules having a number of parameter types and operations; (2) each component has an algebraic specification\(^8\) with equations that are Church-Rosser and terminating\(^9\); and (3) the user's query is a partial algebraic specification, typically consisting of a signature and some ground equations. Assumption (2) is not really limiting, because specifications need not completely characterize the behavior of components, and simple partial specifications are usually Church-Rosser and terminating\(^10\). Similarly, assumption (3) is not limiting, because there is no need for users to be familiar with algebraic specification: query signatures can be expressed as declarations in some familiar programming or specification notation, and the query can be described in terms of the results of executing simple programs. Note that the software base may contain generic modules, and queries may seek to identify a generic module having certain semantic properties.

---

8. [26] describes an approach where components do not need associated algebraic specifications.
9. These terms are explained in Appendix A.
10. Note that any set of ground equations with distinct irreducible left sides is automatically Church-Rosser and terminating.
In our approach, search is organized as a series of increasingly stringent filters on candidate components. We first filter components by comparing their signatures with that of the query. This is accomplished by signature matching, which looks for maps that translate the type and function symbols of the query into corresponding type and function symbols of candidate components. A first stage of signature filtering can compare pre-computed syntactic profiles of components with the profile of the query. These profiles are special data structures that support an efficient approximation of signature matching. Signature matches can be partial, in that only part of the functionality the user seeks may actually be available. Traditional search methods, such as keyword search, could also be used as early filters, if the appropriate information is available\textsuperscript{11}. Profile matching should be followed by full signature matching.

Finally, semantic filters rank components by how well they satisfy the equations in the query. In this process, equations that are logical consequences of the query specification are translated through the signature matches into equations whose proof is attempted in the candidate specifications. For greatest efficiency, it is desirable to restrict queries to be ground equations; these correspond to simple straight line programs. The candidates in the choice set are ranked according to their likelihood of success. If the closest match is partial, the user will need to modify the closest matching component. This whole process can be made iterative.

\textsuperscript{11}\textsuperscript{.}The precision difficulty with keyword search mentioned in Chapter II.1 does not apply in this case, because we are only using it as a filter to reduce the search space.
Our present knowledge indicates that profile and keyword filtering should be applied first in order to eliminate as many components as possible with the lowest possible cost. Therefore syntactic filtering, and keyword filtering, should come before any semantic filtering is attempted. It is also clear that pre-computed catalogues (i.e., indexes) should be used, instead of pulling all the components out of the library for each search. The experiment E in Chapter VIII confirms this conjecture. Figure 2 shows this multi-level filtering architecture; the top line is to indicate user modification of the query in light of the final filtering results.

Figure 2. An Organization Model for Software Component Search
IV. SYNTACTIC FILTERING

Syntactic filtering uses non-behavioral information about components, such as keywords and interface declarations. Our approach involves two levels of syntactic filtering. First, profile filtering computes indexes which partitions the software library in a way that speeds up signature matching. These partitions contain the candidate components. Next, a Profile and Keyword Matching Ratio values are computed for each candidate component. These values will be then combined\(^\text{12}\) with this component Signature Matching Ratio, computed by the signature matching next, to prune components which do not meet the user selection criteria. Secondly, signature matching finds the maps that translate the type and function symbols of the query into the corresponding type and function symbols of the candidate components.

A. KEYWORD MATCHING

Despite its weaknesses, keyword search is still useful, because it is easy to use, inexpensive to implement, and good for indexing components. However, we use keyword filtering cautiously, with a limited number of general keywords that are controlled by a system administrator. Keywords describe categories of components and their relationships to the other components. Sample categories might be data structures, mathematical functions, sort/search routines, and navigation functions.

We use the following function to measure how close the keywords of a query, \(K_{\text{w}_Q}\), are to the keywords of a component, \(K_{\text{w}_M}\):

\[12\text{This combined value is referred as KPS which stands for the product of Keyword, Profile, and Signature matching ratio. Chapter VI formally defines the KPS.} \]
\begin{align*}
\text{KeywordMatchRatio}(Kw_Q, Kw_M) &= \frac{|Kw_Q \cap Kw_M|}{|Kw_Q|}.
\end{align*}

Note that this function measures recall. The numerator represents the number of relevant retrieved keywords while the denominator represents the total number of relevant keywords in a query.

\section*{B. SIGNATURE MATCHING}

This subsection introduces and illustrates the basic concepts of signature matching, under the assumption that there are no subsort\textsuperscript{13} relations; the more complex situation when \( S \) has subsorts (i.e., a partial order containment relation \( \leq \) on data types) is discussed in Sections IV.3 and IV.4 below. We assume\textsuperscript{14} that each component \( M \) in the library has an associated algebraic specification \( T_M \) of the form \((S', \Sigma', E')\), where \((S', \Sigma')\) is a signature with a set \( S' \) of sorts and a set \( \Sigma' \) of functions whose arguments and results have sorts in \( S' \), and where \( E' \) is a set of equations stating properties that the functions in \( \Sigma' \) should satisfy. We also assume that query, \( Q \), is an algebraic specification of the form \((S, \Sigma, E)\) where these symbols mean the same as \( T_M \).

The following illustrates these notions, using the notation of OBJ3; definitions for signature, specification, etc., can be found in Appendix A.

\textbf{Example 1} The algebraic specification for a module defining list of identifiers in our library might have a sort set \( S \) containing the sorts \( \text{Id}, \text{List}, \) and \( \text{Bool} \), and a signature

\textsuperscript{13}Note that algebraic specification theory and OBJ3 use the word “sort” instead of the word “type”.

\textsuperscript{14}We will see how to relax this assumption later.
Σ of functions consisting of the empty list, denoted nil, an append operation *, and a function to test whether an element is in the list, with the following syntax:

\[
\begin{align*}
sorts & \text{Id} \quad \text{Bool} \quad \text{List} \\
op & \text{nill} : \to \text{List} \\
op & _*_ : \text{Id} \ \text{List} \to \text{List} \\
op & \_\text{in} _\_ : \text{Id} \ \text{List} \to \text{Bool}
\end{align*}
\]

The equations in the specification might be:

\[
\begin{align*}
\text{I in nil} &= \text{false} \\
\text{I in (I' * L)} &= \text{if (I == I')} \ \text{then} \ \text{true} \ \text{else} \ \text{I in L} \ \text{fi}
\end{align*}
\]

We also assume that: the library has a set of basic sorts\(^{15}\) for commonly used types like Booleans, identifiers, integers, and floating point numbers; that the names of these basic types and their associated basic operations are identical in the specifications and in the code; and that the library modules and the algebraic specifications for the basic types also have the same names.

**Definition 1** Given two signatures \((S, \Sigma)\) and \((S', \Sigma')\), a permutative signature map \(V: (S, \Sigma) \to (S', \Sigma')\) consists of injective functions \(V: S \to S'\) and \(V: \Sigma \to \Sigma'\) such that for each function symbol \(f: s_1 \ldots s_n \to s\) in \(\Sigma\), there is a permutation \(\pi\) such that \(V(f): V(s_{\pi(1)}) \ldots V(s_{\pi(n)}) \to V(s)\) is a function symbol in \(\Sigma'\). A partial signature match \(V: (S, \Sigma) \to (S', \Sigma')\) is a permutative signature map \(V: (S_o, \Sigma_o) \to (S', \Sigma')\) where \((S_o, \Sigma_o)\) is a subsignature of \((S, \Sigma)\); it is total if \((S_o, \Sigma_o) = (S, \Sigma)\).

---

15. This dissertation will use the words “sort” and “type” interchangeably, and will also use the words “function” and “operation” interchangeably.
The assumption that $V$ is injective on both sorts and operations is reasonable if there is no subsort relations, because otherwise the user would be asking for two or more things that are not actually different.

**Definition 2** Given a library and a query $Q = (S, \Sigma, E)$, the signature choice set for $Q$ consists of all signature matches $V: (S, \Sigma) \rightarrow (S', \Sigma')$ where $T_M = (S', \Sigma', E')$ is the specification of some component $M$, and where each match $V$ is the identity mapping when restriction to the set of basic types and their basic function symbols. Note that the semantic information in the equations is ignored in syntactic matching. To simplify notation and make explicit the module specification associated with a signature match, we may write $V: Q \rightarrow T_M$ for a signature match $V: (S, \Sigma) \rightarrow (S', \Sigma')$ such that $T_M = (S', \Sigma', E')$ is the specification of a component $M$.

The more complex situation when $S$ has subsorts (i.e., a partial order relation $\leq$) is discussed in Sections IV.3 and IV.4 below.

**Example 2** Assume that a user wants to find a module for sets of identifiers, where the sort $\text{Id}$ of identifiers and the sort $\text{Bool}$ of Booleans are basic sorts. Suppose that the signature for such a query includes an empty set, functions for adding and deleting an identifier from a set, and a function to test whether a given identifier is in the set. This can be expressed in OBJ3 notation as follows:
sorts Id Bool Set.
op null : -> Set.
op _+_ : Set Id -> Set.
op _-_ : Set Id -> Set.
op _in_ : Id Set -> Bool.

(Note that the "underscore" characters "_" are place holders, indicating where the arguments should go in "mixfix" functions.)

Suppose that the library, among other things, contains a list of identifiers module whose specification has the signature shown in Example 1. The set of all signature matches from the query to this module has 17 elements. The least defined element $V_1$ is the identity map on Id and Bool and is undefined elsewhere. $V_2$ extends $V_1$ by mapping Set to List. The maps $V_3$-$V_6$ each extend $V_2$ by respectively sending null to nil, sending _+_ to _*_, sending _-_ to _*_, and sending _in_ to _in_. The rest are obtained as unions of $V_3$-$V_6$ satisfying the requirement of being injective, as follows:

- $V_7 = V_3 \cup V_4$;
- $V_8 = V_3 \cup V_5$;
- $V_9 = V_3 \cup V_6$;
- $V_{10} = V_4 \cup V_6$;
- $V_{11} = V_5 \cup V_6$;
- $V_{12} = V_3 \cup V_4 \cup V_6$;
- $V_{13} = V_3 \cup V_5 \cup V_6$;
Our intuitive knowledge of the behavior of sets and lists tells us that the best possible match is $V_{12}$.

For the illustration purpose, a Hasse diagram for the signature maps refinement relation $\subseteq$ is shown next page:

![Figure 3. A Hasse Diagram for the Signature Map Refinement Relation $\subseteq$]

Note that only the maximal elements $V_{12}$ and $V_{13}$ in this partial ordering are of interest - all the others will be strictly less useful. This provides a pruning criterion for signature matching.

**C. PROFILE MATCHING**

The computations for signature matching would be very expensive if it were necessary to try all possible ways of pairing the functions and sorts of queries with those of components. It is therefore highly desirable to cut down the search space. This can be
done. For example, if a query has a function \( f: AAB \rightarrow B \) and a component has a function \( g: ABC \rightarrow D \), then it is obvious that these functions cannot match, because their arguments have different sort patterns; there is no need to compute all possible sort maps to draw this conclusion.

The purpose of profile matching is to speed up signature matching. Profile matching is actually an efficient approximation of signature matching. A profile is a sequence of numbers that describes how the sorts associated with an operation are organized. We can quickly determine whether two given operations could possibly match by comparing their profiles, and hence quickly identify query operations and test cases that will necessarily fail. Profile matching uses pre-computed profile indexes to partition the library, and profile values are used as search keys in seeking components having suitable signatures.

We now introduce some concepts to help us define profiles. The sort groups of an operation are bags (i.e., multisets) consisting of two or more sort occurrences from the rank (i.e., the argument plus value sorts) of the operation that are related under the relation \( = \), which is the transitive-symmetric closure of the ordering \( \leq \) on sorts. The unrelated sort group is the bag (actually a set) of all sort occurrences that are not in any sort group.

**Definition 4** The profile of an operation is a sequence of integers, defined as follows:

1. The first integer is the total number of occurrences of sorts.
2. If the total number of sort groups, \( N \), is greater than 0, then the second to
(1 + N)\(^{th}\) integers are the cardinalities of the sort groups, in descending order.

3. The \((2 + N)\(^{th}\) integer is the cardinality of the unrelated sort group.

4. The \((3 + N)\(^{th}\) integer is:
   - 0 if the value sort is different from any of the argument sorts; and
   - 1 if the value sort belongs to some sort group.

A signature map can relate two operations only if they have the same profile (i.e., the same number of sort occurrences, the same number of sort groups, the same sort group cardinalities, and the same unrelated sort group cardinality).

**Example 3** Some sample profiles are shown in Table 1, where \(A, A', B, C, E, F, G\) are sorts, and \(A'\) is a subsort of \(A\):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow A )</td>
<td>110</td>
</tr>
<tr>
<td>( EF \rightarrow G )</td>
<td>330</td>
</tr>
<tr>
<td>( AA \rightarrow B )</td>
<td>3210</td>
</tr>
<tr>
<td>( ABBCA \rightarrow C )</td>
<td>622201</td>
</tr>
<tr>
<td>( CCBAA \rightarrow B )</td>
<td>622201</td>
</tr>
<tr>
<td>( CCBAA \rightarrow A )</td>
<td>63211</td>
</tr>
<tr>
<td>( CCBAAA' \rightarrow A )</td>
<td>724211</td>
</tr>
</tbody>
</table>

**Table 1: Some Operations and their Profiles**

We are now ready to introduce a theorem which is very useful for the structuring the software library, retrieving candidate components, and performing signature matching. This theorem is stated below and verified through the Experiment C in Chapter VIII. The
formal proof is also provided in Appendix E. Appendix E also shows $\equiv$ as an equivalence relation.

**Theorem 1** Given a query operation and a component operation with their correspond profile values, if these profile values are not equal, then these operations can not be possibly matched.

1. **Software Base Partitioning with Profiles**

Each component $C$ in the software base, $L$, has an implementation part and a specification part. The implementation part is implementation language source code, while the specification part is written in the CAPS prototyping language PSDL with embedded OBJ3 axioms. For each component $C$, let $b(C)$ denote the multiset of profiles of all operations that occur in the signature of $C$, this information may be extracted either from the source code of $C$, or from the PSDL or OBJ3 specification of $C$, and is called the **profile of $C$**.

If $S$ is a multiset of profiles, let $P(S)$ denote the set of all components $C$ in the software base $L$ such that the profile value of $C$ is $S$, i.e., let

$$P(S) = \{ C \in L | S = b(C) \}$$

Let $P$ be the set of all profiles that occur in the software base, and let $\Pi$ denote the set of all subsets $S$ of $P$ such that $P(S)$ is non-empty. Then $\Pi$ is a partially ordered set under set inclusion, and it induces a partition for the set of components in the software base. $\Pi$ can be represented graphically as a so-called **Hasse diagram**, having as its
nodes the elements of $\Pi$, with an edge upward from $S_1$ to $S_2$ iff $S_1 \subset S_2$ and there is no node $S_3$ such that $S_1 \subset S_3 \subset S_2$.

Given a profile $p$ in $P$, we define the **frequency** of $p$ to be the number of profiles $S$ in $\Pi$ that contain $p$, i.e.,

$$Freq(p) = |\{ S \in \Pi \mid p \in S \}|$$

Let us call sets $S'$ such that $|S'| = 1$ **bottom nodes**, and define $\Pi'$ to be $\Pi$ plus all bottom nodes $S'$ not already in $\Pi$, again ordered by set inclusion. This is the structure actually used in the implementation. Note that we can still talk about the frequency of profiles in $\Pi'$, and that $Freq(p)$ can be computed by recursively following edges upward in the Hasse diagram, starting from the bottom node $\{p\}$.

**Example 4** To illustrate the above concepts, assume a small software base such that:

$$
b(C1) = b(C2) = b(C3) = \{p_1, p_2\},
b(C4) = b(C5) = \{p_1, p_2, p_3\},
b(C6) = b(C7) = b(C8) = \{p_1, p_2, p_4, p_6\},
b(C9) = b(C10) = \{p_4, p_5\},
b(C11) = b(C12) = b(C13) = \{p_1, p_2, p_3, p_4\},
b(C14) = b(C15) = \{p_1, p_2, p_4, p_5\}.
$$

Then

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\},$$

and if we assume

$$S_1 = \{p_1, p_2\}, S_2 = \{p_1, p_2, p_3\}, S_3 = \{p_1, p_2, p_4, p_6\}, S_4 = \{p_4, p_5\},$$
then we have that

\[ \Pi = \{S_1, S_2, S_3, S_4, S_5, S_6\} , \]
\[ \Pi' = \{S'_1, S'_2, S'_3, S'_4, S'_5, S'_6, S_1, S_2, S_3, S_4, S_5, S_6\} , \]

and also that

\[ P(S_1) = \{C1, C2, C3\} , P(S_2) = \{C4, C5\} , \]
\[ P(S_3) = \{C6, C7, C8\} , P(S_4) = \{C9, C10\} , \]
\[ P(S_5) = \{C11, C12, C13\} , P(S_6) = \{C15, C16\} . \]

Let us also assume that

\[ Keywords = \{A, B, C, D\} , \]
\[ Kw(C1) = ... = Kw(C5) = \{A, B\} , \]
\[ Kw(C6) = ... = Kw(C10) = \{B, C\} , \]
\[ Kw(C10) = ... = Kw(C15) = \{C, D\} . \]

Figure 4 shows the software base partition for this data.
In our implementation, each block of the partition is described as a set of pointers to indexes in the *ComponentLookupTable*, which has a cell for each component, containing the keywords for the component, and a pointer to physical disk location of the component. These pointers are called *ComponentIDs*. Finally, the software base keeps a separate keyword table for storing keyword identifications (*KeywordIDs*), called the *KeywordTable*. Figure 5 shows the software base index structure for the data in Example 4.
There are two important data that reside on the physical disk. First, the data used for matching. Second, the data used for importation to a user's prototype. The data used for matching are specified by item (a) and (b) below. Once a component is selected by the user, the data used for importation to user's prototype is specified by item (a), (c), and (d). Therefore, we organize the physical file representation of a software base component as follows:

a. A file containing a PSDL specification, for the translation and scheduling for a prototype using the component.
b. A file containing a OBJ3 specification, used for both signature and semantic matching.
c. A file containing an Ada specification, which is imported to become an atomic operator's Ada specification in a prototype, after some modifications.
d. A file containing an Ada body, corresponding to the semantic part. It is imported to a prototype, to become an atomic operator's Ada body, after some modifications.

This organization supports traditional graph search algorithms, such as depth first search, to implement DBMS operations such as initialize, delete, add, and retrieve. There could also be a file for compiled code, which could replace the use of OBJ3 for some queries.

Given a query profile $S_Q$ and component profile $S_C$, we define the ratio ProfileMatchRatio to measure how close the query profile is to the component profile as follows:

$$ProfileMatchRatio(S_Q, S_C) = \frac{|S_Q \cap S_C|}{|S_Q|}.$$
This is again a recall measure. The numerator represents the number of relevant retrieved profiles while the denominator represents the total number of relevant profiles in a query.

2. Retrieving Components with Profile Matching

The following algorithms find the candidate components in a software library:

**Algorithm 8 FindCandidateComponents**

**Input:**

\( Q = (S, \Sigma, E) \).

\( \text{Kw}_Q = \{ \text{Query Keyword Ids} \} \).

\( G = (P, R) \) where \( P = \{ \text{Partitions} \} \) and \( R = \{ \text{Inclusive relations} \} \).

ComponentLookupTable = Array of 2-field records: Component Keywords and Component Pointer.

**Output:**

CandidateTable = Array of 3-field records: Component Id, Keyword Matching Ratio, and Profile Matching Ratio.

Invalid Operations = \{ Invalid Operations \}

1. Compute profile value, \( \bar{p} \), for each operation in \( Q, \Sigma \).
2. For each profile \( p \), verify against the bottom nodes in \( G \). If \( \text{Freq}(p) = 0 \), then store \( p \) in \( \text{Invalid Operations} \).
3. For each ground equation \( i \) in \( Q, E \), identify its associated profiles using the correspond operation’s profile values found in step (1). Store these associated profiles in a variable called \( G_{p_i} \). (Note: \( G_{p_i} \) is a multiset). After that store each of \( G_{p_i} \) into \( G_p \).
4. Let \( N \) be the set of starting nodes contains every node indexed by profile multi-set of size one that is included in the query profile.
5. For each \( n \) in \( N \), call DepthFirstSearchForward.
6. Report invalid operation(s) in \( \text{Invalid Operations} \).
Algorithm 9 *DepthFirstSearchForward*

**Input:**

$G = (P, R)$ where $P = \{\text{Partitions}\}$ and $R = \{\text{Inclusive relations}\}$.  
$v = \text{An element of } P$.  
CandidateTable = Array of 3-field records: Component Id, Keyword Matching Ratio, and Profile Matching Ratio.  
$G_p = \{G_{pi}, \text{where } G_{pi} \text{ is a multiset of profile in a ground equation } i \text{ in Q.E}\}$  
ComponentLookupTable = Array of 2-field records: Component Keywords and Component Pointer.  

**Output:**

CandidateTable = Array of 3-field records: Component Id, Keyword Matching Ratio, and Profile Matching Ratio.  

1. Mark $v$ visited. Assign $P_v$ with Partition $v$'s profile values.  
2. For each $G_p_i$ in $G_p$, If $G_{pi} \subseteq P_v$ then  
   Calculate the *Profile Match Ratio*.  
   For each component pointer at node $v$ do  
   Index to an element in ComponentLookupTable.  
   Calculate the *Keyword Match Ratio*.  
   Store Profile and Keyword Match Ratios and ComponentID in CandidateTable.  
3. For each vertex $n$ adjacent and above $v$ do  
   If $n$ not visited then *DepthFirstSearchForward*.  

**D. SIGNATURE MATCHING ALGORITHM**

The signature matching algorithms seek to find good partial signature maps $V$:  
$(S, \Sigma) \rightarrow (S', \Sigma')$, where $(S, \Sigma)$ is the signature of the query $Q$ and $(S', \Sigma')$ is the signature of a component $M$. Recall that the basic sorts are those common to all modules. The algorithms below take advantage of the following requirements for a signature map $V$, with sort map $V_S: S \rightarrow S'$ and operation map $V_\Sigma: \Sigma \rightarrow \Sigma'$:
1. $V_S$ must be injective.

2. $V_S$ must preserve the subsort relation, i.e., $s_1 \leq s_2$ in $S$ implies $V(s_1) \leq V(s_2)$ in $S'$.

3. $V_S$ must preserve basic sorts.

4. $V_\Sigma$ must be injective.

5. $V_\Sigma$ must preserve basic operations.

6. The profile of each operation $f$ in $\Sigma$ must be the same as the profile of $V_\Sigma(f)$ in $\Sigma'$.

7. If an operation $f$ in $\Sigma$ has argument sorts $s_1, \ldots, s_n$, and if $V_\Sigma(f)$ in $\Sigma'$ has argument sorts $s'_1, \ldots, s'_n$, then there must be a permutation $\pi$ of $\{1, \ldots, n\}$ such that $V(s_{\pi(i)}) = s'_i$ for $i = 1, \ldots, n$, and such that $V(s) = s'$.

This generalizes Definition 2 to the case where there are subsorts. Dolgoff [2] made some initial studies of signature matching with subsorts.

Given a signature match $V$, the measure of how close the signature of the query $Q$ is to a component signature is given by the following:

$$\text{SignatureMatchingRatio}(V,Q) = \frac{|V.\Sigma|}{|Q.\Sigma|},$$

where $Q.\Sigma$ is the signature of the query and $V.\Sigma$ is the subsignature of $Q.\Sigma$ that is actually matched by $V$.

The following three algorithms provide the method for signature matching. The algorithm $\text{SignatureMatch}$ computes $V_\Sigma$, $\text{CalculateSortAssignment}$ computes $V_S$, and $\text{SortAssignable}$ determines whether a sort assignment can be made.

**Algorithm 10 SignatureMatch**

**Input:**

Query signature, $(S,\Sigma)$.
Component signature, $(S',\Sigma')$. 

32
Output:

Signature maps, $V = \{ \text{Signature Maps} \}$.
where Signature map = ( \{Sort Maps\}, \{Op Maps\})
Sort Map = (s $\rightarrow$ s'), s in $S$ and s' in $S'$
and Op Map = (op $\rightarrow$ op'), op in $\Sigma$ and op' in $\Sigma'$.

Variables:

Stack is a FIFO stack for storing/retrieving $\theta$.
i, j are indexes to query and component operations.
Temp Sort Map is a temporary Sort Map

(1) Set $V = \{\}$. 
   Initialize state variable $\theta = (\text{Sort Maps,Op Maps, Operation Mark List} = \{\}, and
   Operation Occupied List = \{\text{Falses}\})$. Set $i = 1$, $m = 1$.
   (2) While True do:
       $j = \text{find_an_available_component_index}(i,\text{Operation Occupied List},$
       \text{Operation Mark List})
       If $j = \text{Invalid}$ and $i /= \text{Invalid}$ then
           If $i = |\Sigma|$ then -- Both i and j are finished
               If Stack is Empty then
                   Exit;
               Else
                   If (Sort Maps,Op Maps) is not a submap in $V$ and preserve
                   subsort relation then
                       $V = V \cup (\text{Sort Maps,Op Maps})$.
                       $\theta = \text{Pop(Stack)}$.
                   End If
               Else
                   $\text{Clear_all_previous_visit_operations}(i,\text{Operation Occupied}$
                   List,\text{Operation Mark List}).
                   $i = i + 1$;
               End If.
       Else -- j is not finished
           Operation Occupied List(j) = True; Operation Mark List(j) = i.
           If Profile(i) = Profile(j) then -- If their profile are the same
               $\text{Clear_previous_visit_operation}(i,j,\text{Operation Occupied List},$
               \text{Operation Mark List})$;
               $\text{Calculate_Sort_Assignments}(\text{Argument Sorts}(i),\text{Argument Sort}(j)$,
               , Range Sort(i), Range Sort(j), Sort Assign Table, Map Num).
               If Map Num > 0 Then -- There is possible sort maps
                   Push($\theta$,Stack);
Op Maps = Op Maps $\cup$ (Symbol(i) $\rightarrow$ Symbol(j)).
For n = 1 to Map_Num do -- Save all possible maps on Stack
    Temp Sort Maps = Sort Maps.
    Sort Maps = Sort Maps $\cup$ (Sort Assign Table(n)).
    Update $\theta$ and Push($\theta$,Stack). -- Save on Stack
    Sort Maps = Temp Sort Maps.
    $\theta = \text{Pop(Stack)}$; -- Now get one
    If i < $|\Sigma|$ then
        i = i + 1.
    Else -- Keep i constant
        If (Sort Maps,Op Maps) is not a submap in V and preserve
            subsort relation then
            V = V $\cup$ (Sort Maps,Op Maps).
            $\theta = \text{Pop(Stack)}$.
        End If
    End If
End If
End If
End If

In discussing the algorithms below, we will use the following terminology: A non-basic sort of the component is \textbf{confined} if it is the image of some sort of the query under $V$, or is related to a confined sort under $\equiv$; a non-basic sort that is not confined is called \textbf{unconfined}. Two sorts $s$, $s'$ that are not related under $V$ are called \textbf{unrelated}, written $s \not= s'$. These algorithms try to map unassigned sorts to appropriate values, and to assign appropriate values to the parameters of generic components.

\textbf{Algorithm 11 Calculate\_Sort\_Assignments}
\textbf{Inputs:}

\begin{itemize}
    \item $S_d$ is the argument sort set of the query operator.
    \item $S'_d$ is the argument sort set of a component operator.
    \item $s_r$ is the value sort of a query operator.
    \item $s_r'$ is the value sort of a component operator.
\end{itemize}
Sort Maps = \{Sort Maps\}, current sort maps.
Sort Map = (s $\rightarrow$ s'), s in S and s' in $S'$
Output:

Sort Maps' = \{Sort Maps\}.

Variables:

Temp Sort Maps is temporary Sort Maps.
SVL is the sort visiting list, for storing pairs of indexes of element in \( S_d \) and \( S'_d \).
i,j are the indexes to argument sort of \( S_d \).
p_{ij} is the previous j index that is being occupied by i.
Lds is a list of flags indicating whether argument sorts in \( S'_d \) are occupied or not.
\( \beta \) is state variable consists of TempSortMaps, SVL, i, Lds.

(1) Initialize \( \beta \) with
\[
\text{TempSortMaps} \leftarrow \text{SortMaps}; \text{SVL}, \text{Lds} \leftarrow \text{Null}; i \leftarrow 1.
\]
Initialize Lds \( \leftarrow \text{False}s \), SAL \( \leftarrow \text{Null} \).
(2) If Sort assignable(Sort(s),Sort(s')) then
\[
\text{TempSortMaps} \leftarrow \text{TempSortMaps} \cup (s, s').
\]
Otherwise, return
(3) Find j such that Lds \( j \neq \text{True} \) and SVL \( j \neq i \). Find a p_{ij} such that SVL \( p_{ij} = i \).
(4) If \( j = \text{Invalid} \), then check the Stack.
If it is \text{Null}, then return
Otherwise, \( \beta \leftarrow \text{pop (Stack)} \) . Return to step (3).
(5) Otherwise, if \( p_{ij} \neq \text{Invalid} \), then Lds \( p_{ij} \leftarrow \text{False} \).
Lds \( j \leftarrow \text{True}, \text{SVL} j \leftarrow i \)
(6) If Sort assignable(sort(i),sort(j)), then Stack \( \leftarrow \text{push} (\beta, \text{Stack}) \).
TempSortMaps \( \leftarrow \text{TempSortMaps} \cup (\text{sort}(i), \text{sort}(j)). \ i = i + 1.
\)
If \( i > |S| \) then
\[
\text{SortMaps'} \leftarrow \text{SortMaps'} \cup \text{TempSortMaps}, \beta \leftarrow \text{pop (Stack)}.
\]
(7) Return to step 3.
Algorithm 12 *SortAssignable*

**Input:**
$q_s$ is a query sort.
$c_s$ is a component sort.

**Output:**
Aflag is a Boolean, true if and only if an assignment can be made.

1. Determine the sort relation between $q_s$ and $c_s$ by consulting a row of Table 2.
2. Determine the sort type of $q_s$ and $c_s$ by consulting a column of Table 2.
3. Look up the result in Table 2 and assign that value to Aflag.
4. If Aflag is *True* then
   Aflag is *True*
   Else
   Aflag is *False*
5. Return Aflag

<table>
<thead>
<tr>
<th>$q_s \rightarrow c_s$</th>
<th>$q_s \not\equiv c_s$</th>
<th>$q_s = c_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic $\rightarrow$ Confined</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Basic $\rightarrow$ Unconfined</td>
<td>T</td>
<td>N/A</td>
</tr>
<tr>
<td>Basic $\rightarrow$ Basic</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Confined $\rightarrow$ Basic</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Confined $\rightarrow$ Unconfined</td>
<td>F</td>
<td>N/A</td>
</tr>
<tr>
<td>Confined $\rightarrow$ Confined</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>Unconfined $\rightarrow$ Basic</td>
<td>T</td>
<td>N/A</td>
</tr>
<tr>
<td>Unconfined $\rightarrow$ Confined</td>
<td>F</td>
<td>N/A</td>
</tr>
<tr>
<td>Unconfined $\rightarrow$ Unconfined</td>
<td>T</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 2: Sort Assignments Table
V. SEMANTIC FILTERING

The choice set of signature matches for a query can be further narrowed by semantic filtering. Under certain very reasonable assumptions, this can be done efficiently by checking satisfaction of certain semantic conditions that are necessary for any correct answer to the query.

We discuss only unparameterized specifications here. In addition, we assume throughout that the specifications associated with components have equations that are Church-Rosser and terminating. This means we can apply the equations from left to right to simplify a term to a unique simplest possible form, called its canonical form, which can be regarded as the result of evaluating the term. For example, the canonical form of the term.

\[ a \text{ in } (b \ast (a \ast \text{nil})) \]

using the equations in Example 1 is true.

Our semantic validation procedure takes ground equations \( t=t' \) from the query specification \( Q \) and tests them for satisfaction in the candidate specifications. Ground equations, that is, equations whose terms have no variables, are particularly useful here, because any ground equation provable from an equational theory \( Q \) is satisfied by the standard model of such a theory, i.e., the initial algebra \( T_Q \) of \( Q \), so that those equations are also satisfied under the initial (standard) interpretation of \( Q \).

This is important because either the query \( Q \) or the component specifications \( T_M \) may have an initial interpretation, so that proving the semantic correctness of a signature
match \( V: Q \rightarrow T_M \) could require complex theorem proving to check inductive consequences. A special property of ground equations is that their translations \( V(t) = V(t') \) must be provable from \( T_M \) in order for \( V \) to be correct, regardless of whether \( Q \) and \( T_M \) are interpreted initially or loosely. Therefore, we can use them under either interpretation to further restrict the search for correct answers to the query \( Q \). The great advantage of ground equation is that we can automatically settle the issue of whether \( V(t) = V(t') \) is provable from \( T_M \) by comparing the irreducible forms of \( V(t) \) and \( V(t') \) after rewriting them with the equations in \( T_M \), since \( T_M \) is assumed to be Church-Rosser and terminating. This results in an efficient decision procedure for behavior queries expressed in this form.

The next subsection explains how to rank members of the choice set based on semantic filtering.

A. ORDERING SEMANTIC MATCHES

Since many signature matches for a query might be found in a large library, it is important to narrow the search by using whatever semantic information is available at each filtering stage, either from the specification \( T_M \), the compiled component \( M \), or both. For this, a sound way of ordering the matches according to their relative degree of semantic correctness is needed. We define below two measures that can be used for this purpose. These measures assign a value to each pair \( (V, I_V) \) consisting of a signature match \( V \) for the query \( Q \), and whatever information \( I_V \) is available at that stage about
equations that have passed or failed the semantic checks. Such measures can be used independently or in combination to define choice sets.

Given a match $V$, the information $I_V$ available for it may be either syntactic or semantic. Syntactic information will include the functions in the query's signature for which the match is defined and their translation under such a match. Semantic information will include the results of checking correctness of ground equations after translating them through $V$. For each such ground equation and match $V$ three things can happen:

- The translated equation is well defined and, after reducing each side to normal form using the equations in the specification of the module matched by $V$, yields an identity; therefore this equation has succeeded for this match;

- The translated equation is well defined and, after reducing each side to normal form using the equations in the specification of the module matched by $V$, yields an equation whose two sides are different; therefore this equation has failed for this match;

- The equation could not be translated, because the match $V$ was undefined for some of the functions appearing in the terms of the equation; this is also a kind of failure.

Note that we can associate each equation with a function symbol, namely the top function symbol of its left side. Therefore, we can assume that the semantic information in $I_V$ is organized by function symbol so that for each such symbol $f$ in the query's signature we have a set of ground equations for it, and information about their success or failure for the match $V$. 

39
The first measure $\mu$ assigns to each pair $(V, J)$ a real-valued function $\mu_{(V, J)}$: 

$\Omega \rightarrow \mathbb{R}$, where $\Omega$ is the signature of the query $Q$, defined as follows:

- If $V(f)$ is undefined, then $\mu_{(V, J)}(f) = 0$.

- $V(f)$ is defined but has no ground equations associated with it, then $\mu_{(V, J)}(f) = 1$.

- Otherwise, $\mu_{(V, J)} = \frac{\text{Success}(f)}{\text{Equation}(f)}$, where $\text{success}(f)$ is the number of successful checks for ground equations $t = t'$ with $t$ having $f$ as its top function symbol reported in $I_V$, $\text{Equations}(f)$ is the total number of such equations.

Consider now two different matches $V$ and $V'$ for the same query $Q$, and suppose that the same ground equations from $Q$ have been tried for $V$ and for $V'$. Then we can compare the degree of success of these matches on an operation by operation basis. If for each operation symbol $f$ we have $\mu_{(V, J)}(f) \geq \mu_{(V', J)}(f)$, then $V$ is an altogether better match than $V'$. The ideal situation is of course when we find a match that is better than all other matches in exactly this sense. However, such an absolutely better match may not exist in the library and we may only get matches that are maximal in their degree of success, that is, no other match is better than them for all functions. This can happen when a query is large enough that two or more parts of the required functionality are available, but their combination is not. In such a case we will find several maximal signature matches that are each best for some fragment of the functionality, but are incomparable among themselves under the $\mu$ ordering. An appropriate environment could use this information to help the user synthesize an optimal combined component.
out of actual components whose corresponding signature matches have maximal $\mu$-measures.

The second measure $SemanticMatchRatio$ is cruder. It is obtained from the first by assigning to each pair $(V, I_V)$ a real number defined by the equation:

$$SemanticMatchRatio(V, I_V) = \sum_{f \in \Omega} \mu^{(V, I_V)}(f),$$

where $\Omega$ is the signature of the query $Q$.

Once we obtain a $SemanticMatchRatio$ of a component, we can compute the overall $ComponentRank$. Since Semantic Matching Ratio is the most significant information and derivable from Profile Matching and Signature Matching, it is important that we order the component base on this. Keyword Match Ratio is the second important piece of information due to how well a user can specify the keywords. The $ComponentRank$ is then a 2-tuple of matching ratios with lexicographic ordering and is defined as follows:

- A component which has a higher $SemanticMatchRatio$ is ordered first. $ComponentRank$ would be included its $SemanticMatchRatio$ and $KeywordMatchRatio$.

- Any components which have the same $SemanticMatchRatios$, then a component with a higher $KeywordMatchRatio$ will be used to order the components. Again $ComponentRank$ would be included its $semanticMatchRatio$ and $KeywordMatchRatio$. 
VI. IMPLEMENTATION

A. CHOICE OF LANGUAGES AND SUPPORT SYSTEMS

This chapter describes the implementation of the Architectural Model for Software Component Search. The prototype of this model is written in Ada programming language. The whole program is about 3280 lines of code. With the intention to improve and port this prototype to the CAPS environment for usage, the author chooses Ada as the primary implementation language. In the future, this work will be also integrated with [36]. This work would include an Ontos data base for storage and retrieval of software components. These data base functions will be written in C++ with Ada bindings to enable them to be used from an Ada main program. All components specifications in the software base are written in the OBJ3 language. All of these processes are executed by a main module called Software Component Search (or SCS). The implementation of each of these processes is described in the coming sections. The component OBJ3 specifications are provided in Appendix B. The Ada source codes are provided in Appendix C. The Unix utilities files to executive OBJ3 environment and support the compile/build of the SCS are provided in Appendix D. Figure 6 next page provides a data flow diagram of the SCS model.
B. QUERY SUBMISSION AND PROFILE GENERATION

Figure 7 (next page) shows the high level structure of Query Submission and Profile Generation processes. These two processes are implemented by an Ada module called GetQuery. This module is executed first by the SCS. It reads in a user’s query specification which consists of a partial specification, keywords, and selection criterion. This partial specification consists of the signature and ground equations. The keywords specify a search path where a user thinks the desired components resided. Finally, a selection criterion is the preference conditions imposed by a user to allow a user to control the search process.
There are two variables. The first one is the product of Keyword, Profile, and Signature Match Ratios (or KPS) which is defined as follows:

\[ KPS = \text{KeywordMatchRatio} \times \text{ProfileMatchRatio} \times \text{SignatureMatchRatio} \]

The purpose of using this KPS value is to allow a user to zoom in a particular range of components which has a particular range of a retrieval syntactic rank's values. This is useful when the number of candidate components is very large. The second one is the block of signature maps that a user is interested in to retrieve. For the current implementation, each block consists of 85 maps. Since the number of maps generated by the SignatureMatch per query can be large, the program needs to establish the map collection points (start and stop points) so that no memory constraint exception would be raised. In this dissertation we refer this as a Mapping Block Number (MBN).

Together, these inputs represent the what (specification), where (keywords), and how (selection criteria) questions for a query.
C. RETRIEVAL OF CANDIDATE SOFTWARE COMPONENTS

Figure 8 shows the high level structure of Retrieval Of Candidate Software Components process. This process is implemented through FindCandidateComponents, InitializeSwb, UpdateCandidateTable, LookupBlockIndex, DetermineProfileIntersection, DepthFirstSearchForward, and InitComponentData.

![Diagram of Retrieval Process]

FindCandidateComponents is the main component of the retrieval of candidate software components. It executes the rest of the modules below it. DepthFirstSearchForward performs a depth-first search on the Hasse diagram to find any whose profiles cover the set of profiles of operation in a ground equation in the query. The module DetermineProfileIntersection determines this condition. If any block satisfies this condition, UpdateCandidateTable is called to include the components of the identified block in a buffer containing the output, called CandidateTable. CalculateKw/Prf-Rank computes KeywordRank and ProfileRank for each CandidateComponent. Finally, InitComponent-
Data sets up a CandidateComponent specification for signature-matching against the user query's specification.

**D. SIGNATURE MATCHING**

Figure 9 shows the high level structure of Signature Matching process. It is implemented by SignatureMatch, FindCompOpIdx, VerifyUpdateRank, ResetPreviousVisit, SortAssignment, Push/Pop Stack operations, UpdateOal modules.

![Figure 9. Structure of the Signature Matching](image)

SignatureMatch is the main executive. It controls all the modules below it. FindCompOpIdx finds an index which points to an available component operation so that its profile can be compared against a query profile. Before storing a signature map into a buffer, VerifyUpdateRank verifies the following conditions prior to storing a map into the SignatureMapTable. They are as follows:

- A Signature map must not be duplicated.
- A Signature map must not be a sub-map of any another signature map in the SignatureMapTable when it is inserted into this table.
- Lastly, the product of Keyword, Profile, and Signature Match Ratios (KPS) of a CandidateComponent must exceed a user specified threshold. This allows a user to control the amount of retrieved components.

ResetPreviousVisit resets the VisitingFlags of a query index operation point to a component index to a free state so that other query operations can be assigned to. SortAssignment determines whether a query sort can be assigned to a component sort. If a sort assignment can be made the SortAssignmentTable will be updated to reflect this. UpdateOal updates the OperationAssignmentTable if the profile and sort of a query operation are compatible with a component operation. Finally, the Push/Pop operations are used as stack operations for storing and retrieving of state variables such as OperationAssignmentTable, SortTableAssignment, VisitingFlags, and component and query indexes during the course of execution of the SignatureMatch.

E. GROUND EQUATION CHECKING

Figure 10 shows the high level structure of Ground Equation Checking. It is implemented by TranslateGroundEquation, ExecuteTestCase, SemanticRank.

![Figure 10 Structure of the Ground Equation Checking](image-url)
TranslateGroundEquation translates query ground equations into component ground equations using all the possible signature maps found by the SignatureMatch module (discussed previously). This module sets the translation flag to indicate whether a ground equation was successfully translated. ExecuteTestCase issues an instantiation statement (OBJ3 Make Statement) if the component is a generic one. It then appends the translated ground equations at the end of the component along with the comments that explain which of the possible signature maps is used along with the original query ground equation. Finally, it invokes OBJ3 to perform term rewriting on these translated ground equations. This invocation is performed through a Unix Script file called Runobj. The result of this execution is then redirected to an output file called Testrun.dat. SemanticsRank reads the result from Testrun.dat and computes the SemanticRank for a component using the equations stated in the Ordering Semantic Matches section. Chapter VIII will provide detailed examples describing these steps.

F. RANKING FORMULATION

Figure 11 shows the high level structure of Ranking Formulation process. This is a simple process. Namely, the ComponentRank is calculated for all the possible maps of the component by evaluate the SemanticMatchRatio and KeywordMatchRatio tuple. The current implementation will keep the highest value along with its map id. The module CalculateTotalRank performs this function.
G. USER INSPECTION AND SELECTION

Figure 12 shows the high level structure of the User Inspection and Selection process. It is implemented by DisplayInvalidOperations and SortDisplayResult. DisplayInvalidOperation displays the invalid operations which do not have the corresponding profile in the current software library. It displays the actual invalid operation names so that a user can be easily isolate the corresponding ground equation in order that a new query formulation can be made. The SortDisplayResult sorts the CandidateComponents in an order high to low and displays them. It displays the map id, Keyword Match Ratio and Semantic Match Ratio for a user evaluation. A log file under module name (with a file extension "tc") is also available for a user to review. It has all the possible maps along with the translated equations.
VII. EXAMPLES

A. SOFTWARE COMPONENT RETRIEVAL EXAMPLES

This chapter provides three examples of the Software Component Search method. The reason for including these examples is to illustrate how the system works and to reinforce the concepts described earlier. Each of the examples presents a query specification, signature maps, results of the runs submitted against the OBJ3 environment, and the final results of the retrieval process. For the purpose of illustration, let us assume that the software base consists of components for generic stack, generic queue, generic bag and list of natural numbers (a non-generic component). The specifications for these components are shown in Appendix B. We also suppose that keywords have been assigned to each component as follows:

Keyword(G-Stack2) = \{Booch, Data-Structure, Stack\},
Keyword(G-Queue) = \{Booch, Data-Structure, Queue\},
Keyword(G-Bag) = \{Booch, Data-Structure, Bag\},
Keyword(N-List) = \{Booch, Data-Structure, List\}

**Example 5. Query for a stack component:**

Suppose that a user submits a query containing the following information:

- **The keywords are:** Booch, Data-Structure, Stack.
- **The partial specification is:**

```plaintext
Package Stack-Of-Nat is Type Stack;
   -- Query operations
   function Empty(Out: Stack).
   function Top(In: Stack; Out: Nat).
   function Push(In: Nat, Stack; Out: Stack).
   function Pop(In: Stack; Out: Stack).
   -- Query ground equations
   Top Push(l,Empty) = Top Pop(Push(6,Push(l,Empty))).
   Pop Push(7,Push(1,Empty)) = Push(1,Empty).
End of Package Stack-Of-Nat;
```
• The selection criterion is to choose a component that has a KPS from 0.50 to 1.0. The MBN is selected to be 0 (first block).

The program scans the query to compute the following profile table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>110</td>
</tr>
<tr>
<td>Top</td>
<td>220</td>
</tr>
<tr>
<td>Push</td>
<td>3211</td>
</tr>
<tr>
<td>Pop</td>
<td>2201</td>
</tr>
</tbody>
</table>

**Table 3: Profile of Operations in Stack-Of-Nat**

We now compute \( G_p \). \( G_p \) is the set that contains set \( S_i \), where \( i \) is a ground equation in \( Q \). Let \( S_i \) contains profiles of all the operations in a ground equation \( i \) in \( Q \). We have

\[
G_p = \{ S_1, S_2 \}, \quad \text{where } S_1 \text{ and } S_2 \text{ are as follows:}
\]

\[
S_1 = \{ 220, 3211, 110, 2201 \}, \quad S_2 = \{ 2201, 3211, 110 \}
\]

Next, the program finds the KeywordIds for the query keywords. The program now executes the FindCandidateComponent and DepthFirstSearch algorithms to search for any nodes whose profile set covers either \( S_1 \) or \( S_2 \). For such nodes, the program uses the ComponentLookupTable to find the corresponding components's KeywordIds and ComponentId. Using this information, the program calculates the ProfileMatchRatio and KeywordMatchRatio, and forwards the components in the node to the signature matching procedure. When the query signature is matched against the signature of G-Stack2, G-Queue, G-Bag, N-list, the following signature maps are obtained:
Query vs. Stack2:

\( V_1: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_1: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow top), (Push \rightarrow push), (Pop \rightarrow pop), (Empty \rightarrow create) \} \)

\( V_2: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_2: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow stacksize), (Push \rightarrow push), (Pop \rightarrow pop), (Empty \rightarrow create) \} \)

\( V_3: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_3: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow top), (Push \rightarrow push), (Pop \rightarrow clear), (Empty \rightarrow create) \} \)

\( V_4: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_4: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow stacksize), (Push \rightarrow push), (Pop \rightarrow clear), (Empty \rightarrow emptyerror) \} \)

\( V_5: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_5: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow top), (Push \rightarrow push), (Pop \rightarrow pop), (Empty \rightarrow emptyerror) \} \)

\( V_6: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_6: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow stacksize), (Push \rightarrow push), (Pop \rightarrow pop), (Empty \rightarrow emptyerror) \} \)

\( V_7: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_7: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow top), (Push \rightarrow push), (Pop \rightarrow clear), (Empty \rightarrow emptyerror) \} \)

\( V_8: Q.S \rightarrow G-Stack2.S' = \{ (Stack \rightarrow Stack), (Nat \rightarrow Elt) \} \)

\( V_8: Q.\Sigma \rightarrow G-Stack2.\Sigma' = \{ (Top \rightarrow stacksize), (Push \rightarrow push), (Pop \rightarrow clear), (Empty \rightarrow emptyerror) \} \)

Query vs. Queue:

\( V_1: Q.S \rightarrow G-Queue.S' = \{ (Stack \rightarrow Queue), (Nat \rightarrow Elt) \} \)

\( V_1: Q.\Sigma \rightarrow G-Queue.\Sigma' = \{ (Top \rightarrow frontof), (Push \rightarrow add.q), (Pop \rightarrow pop.q), (Empty \rightarrow empty) \} \)

\( V_2: Q.S \rightarrow G-Queue.S' = \{ (Stack \rightarrow Queue), (Nat \rightarrow Elt) \} \)

\( V_2: Q.\Sigma \rightarrow G-Queue.\Sigma' = \{ (Top \rightarrow lengthof), (Push \rightarrow add.q), (Pop \rightarrow pop.q), (Empty \rightarrow empty) \} \)

\( V_3: Q.S \rightarrow G-Queue.S' = \{ (Stack \rightarrow Queue), (Nat \rightarrow Elt) \} \)

\( V_3: Q.\Sigma \rightarrow G-Queue.\Sigma' = \{ (Top \rightarrow frontof), (Push \rightarrow add.q), (Pop \rightarrow clear.q), (Empty \rightarrow empty) \} \)

\( V_4: Q.S \rightarrow G-Queue.S' = \{ (Stack \rightarrow Queue), (Nat \rightarrow Elt) \} \)
\[ V_4 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(Top \rightarrow \text{lengthof)}, (\text{Push} \rightarrow \text{add.q}), (\text{Pop} \rightarrow \text{clear.q}), (\text{Empty} \rightarrow \text{empty})\} \]

\[ V_5 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Stack} \rightarrow \text{Queue}), (\text{Nat} \rightarrow \text{Elt})\} \]

\[ V_5 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Top} \rightarrow \text{frontof}), (\text{Push} \rightarrow \text{add.q}), (\text{Pop} \rightarrow \text{pop.q}), (\text{Empty} \rightarrow \text{underflow})\} \]

\[ V_6 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Stack} \rightarrow \text{Queue}), (\text{Nat} \rightarrow \text{Elt})\} \]

\[ V_6 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Top} \rightarrow \text{frontof}), (\text{Push} \rightarrow \text{add.q}), (\text{Pop} \rightarrow \text{pop.q}), (\text{Empty} \rightarrow \text{underflow})\} \]

\[ V_7 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Stack} \rightarrow \text{Queue}), (\text{Nat} \rightarrow \text{Elt})\} \]

\[ V_7 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Top} \rightarrow \text{frontof}), (\text{Push} \rightarrow \text{add.q}), (\text{Pop} \rightarrow \text{clear.q}), (\text{Empty} \rightarrow \text{underflow})\} \]

\[ V_8 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Stack} \rightarrow \text{Queue}), (\text{Nat} \rightarrow \text{Elt})\} \]

\[ V_8 : Q_\Sigma \rightarrow G\text{-}Queue.\Sigma' = \{(\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{add.q}), (\text{Pop} \rightarrow \text{clear.q}), (\text{Empty} \rightarrow \text{underflow})\} \]

**Query vs. N-list:**

\[ V_1 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_1 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{headof}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{tailof}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_2 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_2 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{tailof}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_3 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_3 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{headof}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_4 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_4 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{length}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_5 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_5 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{headof}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{clearhead}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_6 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_6 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{cons}), (\text{Pop} \rightarrow \text{clearhead}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_7 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]

\[ V_7 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Top} \rightarrow \text{headof}), (\text{Push} \rightarrow \text{sethead}), (\text{Pop} \rightarrow \text{tailof}), (\text{Empty} \rightarrow \text{nil})\} \]

\[ V_8 : Q_\Sigma \rightarrow N\text{-}List.\Sigma' = \{(\text{Stack} \rightarrow \text{List}), (\text{Nat} \rightarrow \text{Nat})\} \]
\[ V_8 : Q.\Sigma \rightarrow N-List.\Sigma' = \{ (\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{sethead}), (\text{Pop} \rightarrow \text{tailof}), (\text{Empty} \rightarrow \text{nil}) \} \]

**Query vs. Bag:**

\[ V_1 : Q.S \rightarrow G-Bag.S' = \{ (\text{Stack} \rightarrow \text{Bag}), (\text{Nat} \rightarrow \text{Elt}) \} \]

\[ V_1 : Q.\Sigma \rightarrow G-Bag.\Sigma' = \{ (\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{add}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{empty}) \} \]

\[ V_2 : Q.S \rightarrow G-Bag.S' = \{ (\text{Stack} \rightarrow \text{Bag}), (\text{Nat} \rightarrow \text{Elt}) \} \]

\[ V_2 : Q.\Sigma \rightarrow G-Bag.\Sigma' = \{ (\text{Top} \rightarrow \text{uniqueExtentOf}), (\text{Push} \rightarrow \text{add}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{empty}) \} \]

\[ V_3 : Q.S \rightarrow G-Bag.S' = \{ (\text{Stack} \rightarrow \text{Bag}), (\text{Nat} \rightarrow \text{Elt}) \} \]

\[ V_3 : Q.\Sigma \rightarrow G-Bag.\Sigma' = \{ (\text{Top} \rightarrow \text{lengthof}), (\text{Push} \rightarrow \text{remove}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{empty}) \} \]

\[ V_4 : Q.S \rightarrow G-Bag.S' = \{ (\text{Stack} \rightarrow \text{Bag}), (\text{Nat} \rightarrow \text{Elt}) \} \]

\[ V_4 : Q.\Sigma \rightarrow G-Bag.\Sigma' = \{ (\text{Top} \rightarrow \text{uniqueExtentOf}), (\text{Push} \rightarrow \text{remove}), (\text{Pop} \rightarrow \text{clear}), (\text{Empty} \rightarrow \text{empty}) \} \]

Once the computation of signature maps is complete, the following steps are performed:

- Translate the query ground equations by applying \( V \):

**Query vs. Stack2:**

\[ V_1 : (Q.Eq1 \rightarrow G-Stack2.Eq1) : \]

\[ ((\text{Top} (\text{Push} (1, \text{Empty}))) = \text{Top} (\text{Pop} (\text{Push} (6, \text{Push} (1, \text{Empty})))))) \rightarrow \]

\[ ((\text{top} (\text{push} (1, \text{create}))) = \text{top} (\text{pop} (\text{push} (6, \text{push} (1, \text{create})))))) \]

\[ V_1 : (Q.Eq2 \rightarrow G-Stack2.Eq2) : \]

\[ (\text{Pop} (\text{Push} (7, \text{Push} (1, \text{Empty})))) = \text{Push} (1, \text{Empty})) \rightarrow \]

\[ (\text{pop} (\text{push} (7, \text{push} (1, \text{create})))) = \text{push} (1, \text{create})) \]

**Query vs. Queue:**

\[ V_1 : (Q.Eq1 \rightarrow G-Queue.Eq1) : \]

\[ ((\text{Top} (\text{Push} (1, \text{Empty}))) = \text{Top} (\text{Pop} (\text{Push} (6, \text{Push} (1, \text{Empty})))))) \rightarrow \]

\[ ((\text{frontof} (\text{add} Q (1, \text{empty}))) = \text{frontof} (\text{pop} Q (\text{add} Q (6, \text{add} Q (1, \text{empty}))))) \]
\[ V_1 : (Q.Eq2 \rightarrow G-Queue.Eq2) : \\
(Pop(Push(7, Push(1, Empty)))) = Push(1, Empty)) \rightarrow \\
(pop.Q(\text{add}.Q(7, \text{add}.Q(1, empty)))) = \text{add}.Q(1, empty) \]

**Query vs. List:**

\[ V_1 : (Q.Eq1 \rightarrow N-List.Eq1) : \\
((Top(Push(1, Empty))) = Top(Pop(Push(6, Push(1, Empty)))))) \rightarrow \\
((\text{headof}(\text{cons}(1, nil))) = \text{headof}(\text{tailof}(\text{cons}(6, \text{cons}(1, nil)))))) \]

\[ V_1 : (Q.Eq2 \rightarrow N-List.Eq2) : \\
(Pop(Push(7, Push(1, Empty)))) = Push(1, Empty)) \rightarrow \\
(\text{tailof}(\text{cons}(7, \text{cons}(1, nil)))) = \text{cons}(1, \text{cons}) \]

**Query vs. Bag:**

\[ V_1 : (Q.Eq1 \rightarrow G-Bag.Eq1) : \\
((Top(Push(1, Empty))) = Top(Pop(Push(6, Push(1, Empty)))))) \rightarrow \\
((\text{lengthof}(\text{add}(1, empty))) = \\
\text{lengthof}(\text{clear}(\text{add}(6, \text{add}(1, empty)))))) \]

\[ V_1 : (Q.Eq2 \rightarrow G-Bag.Eq2) : \\
(Pop(Push(7, Push(1, Empty)))) = Push(1, Empty)) \rightarrow \\
(\text{clear}(\text{add}(7, \text{add}(1, empty)))) = \text{add}(1, empty) \]

The translated ground equations are only shown only here for first mapping. In the course of execution, all the translated ground equations for all signature matches will be considered for GroundEquationChecking, which also does the following:

- Instantiate the formal parameter(s) with actual parameter(s) using the OBJ3 make command. This yields a non-generic component. This step is used on G-Queue, G-Stack, G-Bag, but not N-list since it is not a generic component. The sort correspondence in the signature match is used to determine the actual parameter for the instantiation.

- Append the translated ground equations to the end of G-Queue, G-Stack, G-Bag, and N-List OBJ3 code, with the OBJ3 reduce command.

As an example, these transformations are shown below for Nat-Stack2 (G-Stack2 with Nat instantiation). Note that for the purpose of illustration, the author only
shows the first one signature map out of the total of eight: The lines starting with
the symbol ("*") are comment lines.

```
obj Stack2[X :: TRIV] is sort Stack.
  protecting NAT.
op create : -> Stack.
op copy : Stack Stack -> Stack.
op clear : Stack -> Stack.
op push : Elt Stack -> Stack.
op pop : Stack -> Stack.
op empty : -> Stack.
op top : Stack -> Elt.
op depthof : Stack -> Nat.
op isempty : Stack -> Bool.
op isequal : Stack Stack -> Bool.
op StackError : -> Stack.
op StackError : -> Elt.
var S S1 S2 : Stack.
var X : Elt.
eq clear(S) = empty.
eq copy(empty,S) = clear(S).
eq copy(push(X,S1),S2) = push(X,copy(S1,S2)).
eq top(empty) = StackError.
eq top(clear(S)) = StackError.
eq top(push(X,S)) = X.
eq top(push(X,S)) = S.
eq pop(empty) = StackError.
eq pop(clear(S)) = StackError.
eq pop(push(X,S)) = S.
eq pop(create) = StackError.
eq depthof(empty) = 0.
eq depthof(create) = 0.
eq depthof(push(X,S)) = 1 + depthof(S).
eq isempty(S) = if (S = create) or (S = empty) then true else false fi.
eq isequal(S1,S2) = if S1 = S2 then true else false fi.
endo

make testobj is Stack2[NAT] endm
```
This specification now is ready to be given to OBJ3 by a Unix script file called Runobj (see Appendix D). The result of this execution directed to a file called Testrun.dat. Note that the real OBJ3's output file also contains other things, such as the OBJ header. This information is removed. The final result is stored in Testrun.dat. This file contains the following data:

```
Bool: true
Bool: true
Bool: true
Bool: true
Bool: false
Bool: false
Bool: false
Bool: false
Bool: true
Bool: true
```

The first two Bool statements correspond to the first signature map for the two translated ground equations (shown previously). They are term-rewritten to be true. The rest of the Bool statements correspond to the rest of the term-rewritten translated ground equations for the rest of the signature maps. Next the program reads this data to compute the SemanticRank. Finally, the ComponentRank for each component is computed. The result of a retrieval session for the Stack-Of-Nat query is shown below:

The result of your retrieval session is:

Find component: Stack2.obj
Using Map Number: 1
The ComponentRank: (2.0E+00,1.0E+00)

Find component: list.obj
Using Map Number: 1
The ComponentRank: (2.0E+00,6.7E-01)
Find component: queue.obj
Using Map Number: 2
The ComponentRank: (1.0E+00, 6.7E-01)

Find component: bag.obj
Using Map Number: 3
The ComponentRank: (0.0E+00, 6.7E-01)

We note here that the ComponentRank is a 2-tuple value. The first field of the 2-tuple represents the SemanticMatchRatio and the second field is the KeywordMatchRatio. There is only one component that fully meets the query, namely, G-Stack2. N-List is ranked second, since it has a lower KeywordMatchRatio. However, it fully meets the SemanticMatchRatio. G-Queue is ranked third since it has the lowest SemanticMatchRatio. Finally, G-bag is ranked fourth since it completely fails the semantic evaluation. The end result of the retrieval session suggests that we can use G-Stack2 or N-list for the Stack-Of-Nat query.

**Example 6. Query for a set component:**

In this example, we are adding a new generic set component into our library. The specification for this set component is shown in Appendix B. Let us suppose that a user submits a query which consists of the following operations:

- **Union:** Given two sets, form a set containing the items that are members of the first set or the second set.
- **Subset:** Return true if the first set is a subset of the second set.
- **Intersection:** Given two sets, form a set containing the items that are members of the first set and the second set.
- **Insertion:** Insert an item as a member of the set.
- **Cardinality:** Return the current number of items in the set.
• Equal: Return true if the two given sets have the same state and false otherwise.

• Clear: Remove all the items from the set and make the set empty.

• Create: A set constant.

• Copy: Copy the items from one set to another set.

A user further wants to look for a component that has the following semantic samples:

• Eq 1: Clearing an non-empty set is the same as copying an empty set to a set.

• Eq 2: A set that contains elements of another set is a subset of that set.

• Eq 3: An empty set should be a result of applying an intersection operation on two disjoint sets.

• Eq 4: The result of the union of two sets is equal to a set that contains their elements.

• Eq 5: The cardinality of a set is the occurrences of the elements in that set.

Formally, these semantic samples are expressed as ground equations together with their operations in a partial specification below:

Package Set-Of-Nat is Type Set;
-- Operations:
Function: Union(In: Set, Set; Out: Set).
Function: Subset(In: Set, Set; Out: Boolean).
Function: Intersection(In: Set, Set; Out: Set).
Function: Cardinality(In: Set; Out: Boolean).
Function: Equal(In: Set, Set; Out: Boolean).
Function: Create(Out: Set).
Function: Clear(In: Set; Out: Set).
Function: Copy(In: Set, Set; Out: Set).
-- Ground equations:
Eq1: Clear(Insert(1, create)) = Copy(create, Clear(Insert(1, create))).
Eq2: Subset(Insert(1, Insert(2, create)), Insert(1, Insert(3, Insert(2, create)))) = True.
Eq3: Equal(Insert(1, create), Intersection(Insert(1, create), Insert(3, Insert(2, create)))) = True.
Eq4: Equal(Insert(1, Insert(2, create)), Union(Insert(1, create), Insert(2, create))) = True.
Eq5: Cardinality(Insert(1, Insert(2, Insert(3, Insert(4, Insert(5, create)))))) = 5.
End of Package Set-Of-Nat;
A user further chooses the following Keywords: Booch, Data-Structure, and Set. Having a high confidence in finding this component, a user chooses a KPS from 0.9 to 1.0. Finally, a MBN of 0 is also selected for retrieval. The following is the actual output of the SCS program given the query data above:

The result of your retrieval session is:

Find component: set.obj
Using Map Number: 15
The ComponentRank: (4.0E+00,1.0E+00)

This result says that we have found a full match. The KeywordMatchRatio is equal to 1.0 since every thing is matched. Due to the equations 3 and 4 having the same top function “Equal”, the SemanticMatchRatio is 4.0 instead of 5.0 (for 5 ground equations). This is exactly equal to the semantic ranking scheme that we have proposed early in section 5.1. This result is obtained using a signature map number 15 from log file called Set.obj tc. It is as follows:

### Map # 15
### Sort Assignment:
- Set -> Set
- NAT -> Elt
- BOOL -> BOOL
### Operator Assignment:
- Create -> create
- Insert -> add
- Cardinality -> cardinality
- Clear -> clear
- Union -> union
- Equal -> isequal
- Intersection -> intersection
- Subset -> subsetof
- Copy -> copy
If we look closer at the sort and operation assignment, they look almost the same. The result of this retrieval session says that we have successfully located a reusable component call Set.

**Example 7. Query for a Ring component:**

In our final example, a Ring component is added into the library. The specification of this Ring component is shown in Appendix B. Ring data structure is a sequence of zero or more items arranged in a circular fashion. Because this structure wraps around itself, it is highly symmetrical and has many useful applications ranging from manipulation of polynomials to user interface. Suppose now that a user is interested in a Ring Component. The operations requested are:

- **Forward**: Forward Direction (clockwise).
- **Empty**: Ring constant represents an empty string.
- **Adding**: Add an item at the top of the ring.
- **Pop**: Remove an item at the top of the ring.
- **Rotation**: Rotate the ring in a given direction.
- **Cardinality**: Return the current number of items in the ring.
- **TopofRing**: Return the item at the top of the ring.
- **Testing**: This operation is used only for testing an operation that is not valid under a software library.

A user further wants to look for a component which has the following semantic samples:

- **Eq1**: Adding an element to a top of an empty ring and removing it results in an empty ring.
- **Eq2**: The top element of a ring should be the most recently added one.
- **Eq3**: The element in a ring is the number of elements added in an empty ring.
Eq4: The forward rotation of a 3 element ring should move each element two positions clockwise.

Eq5: This equation is used for testing of an unknown operation (invalid profile) in a software library.

The information above can be formalized as a partial specification as follows:

```
Package Ring-Of-Nat is Type Ring;
Import Type Direction;

-- Operations:
Function: Forward(Out: Direction).
Function: Empty(Out: Ring).
Function: Cardinality(In: Set; Out: Boolean).
Function: Adding(In: Nat,Ring; Out: Ring).
Function: Pop(In: Nat,Ring; Out: Ring).
Function: Rotation(In: Direction,Ring; Out: Ring).
Function: TopofRing(In: Ring; Out: Nat).
Function: Testing(In: Ring,Ring,Ring; Out: Ring).

-- Ground equations:
Eq1: Pop(Adding(1,Empty)) == Empty.
Eq2: TopofRing(Adding(1,Adding(2,Adding(3,Empty)))) == 1.
Eq3: Extentof(Adding(2,Adding(3,Adding(1,Empty)))) == 3.
Eq4: Rotation(Forward,Adding(1,Adding(3,Adding(2,Empty)))) == Adding(3,Adding(2,Adding(1,Empty))).
Eq5: Testing(Empty,Empty,Empty,Empty) = Empty.
```

This time a user chooses the following keywords: Booch, Data-Structure, and Ring.

Since a user is also interested in partial matching, a KPS of 0.6 to 1.0 is selected. Two retrieval sessions will be shown for this query due to the number of partial mappings being very large. The result of the first retrieval session is as follows: (we are using a MBN of 0 for the first 85 maps)

The result of your retrieval session is:

Find component: stack1.obj
Using Map Number: 14
The ComponentRank: (3.0E+00,6.7E-01)

Find component: Stack2.obj
Using Map Number: 2
The ComponentRank: (3.0E+00,6.7E-01)
Find component: ring.obj
Using Map Number: 2
The ComponentRank: (2.0E+00,1.0E+00)

The following operation(s) is unknown: Testing

We further review how the results are formulated by looking at maps number 2 and 14 in the log files for Stack1.obj.tc and Stack2.obj.tc. This data shows that ground equations 1, 2, and 3 were term-rewritten to be true. However, equation 4 can not be translated due to some of the operations in this equation being undefined. Even though the Keyword-MatchRatio is high, component Ring ranks second (both Stack1 and Stack2 rank first) due to a low SemanticRank. Next, we go further by looking at the block 1 of the signature maps (the next 85 maps). The retrieval result of the second retrieval session using exactly the same query formulation is as follows:

The result of your retrieval session is:

Find Component: ring.obj
Using Map Number: 65
The ComponentRank: (4.0E+00,1.0E+00)

The following operation(s) is unknown: Testing

This result indicates that Ring component is a better matched component than stack1 or Stack2 since the Ring’s ComponentRank is much higher than both. This is so because the SemanticMatchRatio value is higher than the previous session. Further reviewing of the result in the log file of the Ring component indicates that the first four ground equations are rewritten to be true. The final result also warns a user that operation
“Testing” from the query formulation is not visible in the current software library. This is reported in both retrieval sessions.

In conclusion, the end result of these three examples shows the system can:

- Retrieve reusable software components.
- Discriminate between components in order to provide to a user an indication of which component is a better match. The system can tell the user how close or further apart the retrieved component is relative to a query.
- Provide useful information to explain how the result is formulated at the end of the retrieval session.
- The formulation of the query is clear, friendly enough and not so difficult that a user would not be able to use and comprehend.
- Isolate invalid operations that will help a user to reformulate a new query if needed.
- Retrieve both generic and non-generic components.
 VIII. EXPERIMENTATION

A. INTRODUCTION

This chapter provides the important experimentation results to assess the practical usefulness of the proposed method for retrieving reusable software components. Five different experiments were performed. The first experiment evaluates the user's competence in formulating the ground equations for retrieval of software components. The second and third experiments deal with the performance of the system in relation to the Signature Matching Algorithms. The fourth experiment describes the analysis and performance of the software library using the Hasse Diagram as a model. The fifth experiment provides some results regarding the retrieval performance of the system.

B. EVALUATION USING PARTIAL SPECIFICATIONS AS QUERIES

As proposed in this dissertation, using partial specifications (test cases) as a query for software components is a relatively easy task that can be performed by a typical software engineer. For a software component retrieval task, formal specification is considered by the author as overkill, unfriendly, time consuming, and inflexible for reformation of queries. Because it is easy to comprehend and simple to construct, partial specification as query could help to overcome some of these difficulties. The experiment describes below is a measurement to check the aforementioned conjectures.

An assignment was given to CS-4520 (Advance Software Engineering class at NPS) students to: (1) write two formal specifications for the two software components in the Booch software library, (2) verify the correctness of these specifications by formulat-
ing test cases (ground equations) using only ground terms, (3) evaluate the following assessments:

- Comparing the difficulty in writing formal specification to ground equations.
- Assess whether a software engineer without the knowledge of formal specification could write ground equations for the retrieval of software component task.
- Provide some guide lines to help the user in formulating queries.
- Assess whether OBJ3 can be used for specifying reusable software components.

The result from these assessments are as follows:

- 83% of the students believe that the formulation of ground equations is significantly easier than writing the complete specification for modules. Most students claimed having difficulties in formulating axioms in the beginning of the writing formal specifications due to a change of mind set from procedural languages to rewrite rules. Some others students have some difficulty in formulating axioms due to the involvement with recursiveness in some problems.

- 66% of students believe that a software engineer can formulate such ground equations without the knowledge of algebraic specification. This assumes that he/she knows what are the operations required, and understand the desired behavior of the sought component(s) with respect to these operations.

- The important guidelines to supporting the users in formulating the query through ground equations are as follows:
  - A user should start with the most basic primitive operations and then incrementally test other complex operations. This is like description in a piece-meal fashion. A user learns more about the search components as the search process continues.
  - Per query, a user ought to keep a manageable size of operations and ground equations such that he/she can keep track of the semantics of the search component(s).
  - To further discriminate between components, a user should form ground equations that check on boundary conditions as well as representative samples.
• A user should avoid using too many repeated terms in order to reduce the amount of term rewriting. This will speed up the term rewriting process.

• A user should differentiate between constructors and accessors in order that ground equations can be formulated correctly.

• 83% of students believe that OBJ3 has a real potential usage in specify reusable software components. One big problem is that writing the OBJ3 specification is as error prone as programming. If this scheme is chosen, an OBJ3 specification must be well documented, tested and approved by a committee before insertion into a software library. The second problem with OBJ3 is that its error messages come back to the user are difficult to comprehend and misleading sometimes.

C. SIGNATURE MATCHING ALGORITHMS PERFORMANCE

As described earlier, SignatureMatch algorithm uses the profile definition as a way to eliminate a useless component operation when a query operation is being matched against it. This is indeed an approximate signature matching process. This experiment serves three purposes:

• To verify the correctness of this approximation to assure that no component operation is eliminated wrongly when it is matched against a query operation.

• To measure the efficiency in the reduction of the number of computational operations required when profile definition is used.

• To validate Theorem 1 stated in Chapter IV and proved in Appendix E.

One approach for verifying this experiment is to eliminate the profile equality check in Chapter V (also see item six of the Signature Matching Requirements) of the SignatureMatch algorithm. By doing this, we allow any query operation to pair with any component operations with only the remaining 6 restrictions mentioned in section IV.4. To verify the correctness, two separate executions were made, one with and one without pro-
file approximation. The end results of the two retrieval sessions were compared to see if the retrieved components chosen from the two executions are the same. In addition to this, their corresponding rankings and signature maps were also compared for equality. This experiment is performed using the same query as in Example 7 of Chapter VI. This query was executed together with two separate Signature Matching Algorithms. The first one, SMatch1, uses approximation while the second one, SMatch2, does not. Finally, a counter was inserted into these algorithms to count the number of the pairs of query and component operation that must be evaluated with and without including profile as equality check.

The result of this experiment is summarized as follow:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Stack2</th>
<th>Bag</th>
<th>Queue</th>
<th>Stack1</th>
<th>N-List</th>
<th>Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMatch1</td>
<td>302</td>
<td>494</td>
<td>500</td>
<td>674</td>
<td>1037</td>
<td>1722</td>
</tr>
<tr>
<td>SMatch2</td>
<td>2937</td>
<td>57677</td>
<td>11042</td>
<td>16910</td>
<td>27152</td>
<td>1251649</td>
</tr>
</tbody>
</table>

Table 4: Number of the Pairs of Query and Component Operation Performed for SMatch1 and SMatch2 Using the Query from Example 7

Once this result is obtained, the retrieved component of both SMatch1 and SMatch2 are compared and evaluated. It turns out that both SMatch1 and SMatch2 produce the same result. Only Ring component was retrieved. The Ring's SemanticRanks for both SMatch1 and SMatch2 have the same value. In addition, the signature maps that were used to calculate the ComponentRank for the Ring component were exactly the same. The second part of this experiment was to come up with some timing information in order that we can appreciate the effort in using approximation for signature matching process. This experiment concludes that by using profile definition to approximate the signature match-
ing process, we are able to produce the same result in a very much shorter time (about 56 times faster than using no approximation). This result also confirms Theorem 1 stated in Chapter IV.

D. REDUCTIONS OF USELESS MAPS

A goal of this dissertation is not only to demonstrate a proof of concept but to also try to make it into as practical a tool as possible. A challenge for the SignatureMatch algorithm is trying to retain only the useful maps and to discard the useless ones. This helps to reduce the storage capacity and to improve the speed of the retrieval process. This experiment provides statistical data to demonstrate that this reduction indeed helps with this process. The current prototype uses two different filtering stages to eliminate useless maps. The first stage rejects maps that are either duplicates or sub-maps of the maps that are currently residing in the SignatureMapTable. The second stage eliminates maps that do not have at least one translatable "top" function. For the purpose of efficiency, these map filtering stages are implemented as short-circuit operators (in Ada) to speed up the execution. To assure that no map is eliminated wrongly, this experiment was performed on separate executions using the query in Example 5. The first one, Exec1, included both elimination of redundant/sub-maps and nontranslateable "top" function map. The second one, Exec2, involved only the elimination of redundant/sub-map maps. Finally, the third,
Exec3, involves no elimination of maps. The following table summarizes the results of these executions:

<table>
<thead>
<tr>
<th></th>
<th>Stack1</th>
<th>Stack2</th>
<th>Queue</th>
<th>Bag</th>
<th>N-List</th>
<th>Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec1</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>48</td>
<td>72</td>
</tr>
<tr>
<td>Exec2</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>49</td>
<td>74</td>
</tr>
<tr>
<td>Exec3</td>
<td>168</td>
<td>125</td>
<td>126</td>
<td>81</td>
<td>351</td>
<td>298</td>
</tr>
</tbody>
</table>

Table 5: Number of the Maps Include as Part of the Three Executions Using the Query from Example 5.

In the same fashion as in Experiment 2, the end results of these executions were compared. As expected, the prototype was able to retrieve the same components for the three executions. However, Exec3, took longer to complete than Exec2 and Exec1, because all of the useless maps need to be evaluated before the final result can be produced. This is confirmed with given number of maps shown in row 3 in Table 5 above. Table 5 also indicates that there is some reduction from Exec2 to Exec1, but it is not very much.

This experiment also gives a rough idea of a number of maps that a Signature-Match algorithm produced. It provides a significant improvement for the system in terms of maps storage and execution. However, even with these reduction techniques, some test data indicates that the number of maps can possibly go up to 400 maps when all possible maps were considered. This is the worst case when all the partial maps were chosen.
E. MODEL OF A SOFTWARE LIBRARY

Another issue address by this dissertation is how a software library can be constructed so that it helps with the retrieval process. Without a software library model, the approach proposed in this dissertation would be too costly if every single component from a library must be evaluated through signature and semantic matching before a system can produce the result. There must be a way such that these matching procedures can be directed to only work on the candidate components that are highly desired by a user (and not the useless ones). To satisfy this requirement, an approach has been proposed in Section IV.3. In this chapter, a smaller scale model of a library is constructed to analyze, test and verify the theory of this proposed model. The currently ongoing work of [36] and [37] will implement and carry this idea further.

1. General Discussion of the Software Library Components

For this experiment, a collection consisting of twelve useful data structures from the Booch reusable software library [39] was selected. They are as follows: Array, Bag, Binary Tree version 1, Binary Tree version 2, Deque, Queue, List version 1, List version 2, Set, Stack version 1, Stack version 2, and Ring. These modules were written in OBJ3 language by the students (including the author) from the CS-4520 class as a course project. After that, the author further refined the specification to: (1) unify the syntax so that the prototype can work with, (2) make minor corrections for improvement, (3) include documentation, (4) collect their profile definition, and (5) finally, populate them into the
<table>
<thead>
<tr>
<th>Profile Id</th>
<th>Operation pattern</th>
<th>Stack 1</th>
<th>Stack 2</th>
<th>Bint1</th>
<th>Bint2</th>
<th>Queue</th>
<th>Set</th>
<th>Array</th>
<th>Bag</th>
<th>List1</th>
<th>List2</th>
<th>Ring</th>
<th>Deque</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1=110</td>
<td>( \rightarrow A )</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P2=2201</td>
<td>( A \rightarrow A )</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>P3=220</td>
<td>( A \rightarrow B )</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>P4=3211</td>
<td>( AB \rightarrow A )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P5=330</td>
<td>( AB \rightarrow C )</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P6=3301</td>
<td>( AA \rightarrow A )</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P7=3210</td>
<td>( AA \rightarrow B )</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P8=4311</td>
<td>( ABB \rightarrow B )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9=4221</td>
<td>( ABC \rightarrow B )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6: Profile Definitions for Twelve Software Components
Figure 13. A Hasse Diagram Represents a Software Base Partition
simulated software library (as a Hasse diagram) to integrate them with the prototype. Table 6 summarizes the profile values with respect to the components. Figure 13 models a software library for the twelve components using a Hasse diagram.

2. Analysis of Software Library using Hasse Diagram

Algorithm DepthFirstSearchForward is responsible for the retrieval process of software components. As a depth-first search-based algorithm, its traversal time complexity is \( O(P + R) \) using an adjacency list. \( P \) represents the number of partitions and \( R \) represents the inclusive relations in the Hasse diagram. The time complexity of retrieval of candidate component is \( O(|P| \times \text{MaxProfileSet}) + |\text{MatchedComponents}| \) . The product of \( (|P| \times \text{MaxProfileSet}) \) is the cost for finding partitions that contain candidate components. The cardinality of \( |\text{MatchedComponents}| \) is the cost for outputting the matched components. The space complexity is as follows:

\[
O(|P| + |\text{Profile}| + \text{MaxProfileSetSize} + |\text{Components}| + |R|)
\]

3. A Simulation Study and Evaluation

Suppose, a user enter a following query:

```plaintext
Package Set-Of-Nat is Type Set;
-- Operations:
Function Union(In: Set, Set; Out: Set).
Function Subset(In: Set, Set; Out: Boolean).
Function Intersection(In: Set, Set; Out: Set).
Function Cardinality(In: Set; Out: Boolean).
Function Insert(In: Nat, Set; Out: Set).
Function Equal(In: Set, Set; Out: Boolean).
Function Create(Out: Set).
Function Clear(In: Set; Out: Set).
Function Memberof(In: Nat, Set; Out: Bool).
-- Ground equations:
Eq: Memberof(1, Insert(1, Create)) = True.
End of Package Set-Of-Nat;
```

Given this query, the system computes the profile values for the query operations.
After that it identifies the profile values for the MemberOf, Insert, and Create. They are 330 (P5), 3211 (P4), and 110 (P1) (see Figure 13 and Table 3). Next FindCandidateComponents calls DepthFirstSearchForward to look for any partition which has a set of profiles containing 330, 3211, and 110. There are two partitions that satisfy this condition. These partitions are 3 and 5 (see Figure 13). They include in the following components: Set, Array, Bag, List1, and List2. The rest of the modules are not retrieved since they do not belong to a partition which contains profile 330. The following is the actual result collected from an execution:

The result of your retrieval session is:

Find Component: set.obj
Using Map Number: 1
The ComponentRank: (1.0E+00,1.0E+00)

Find Component: array.obj
Using Map Number: 2
The ComponentRank: (1.0E+00,6.7E-01)

Find Component: list2.obj
Using Map Number: 84
The ComponentRank: (0.0E+00,6.7E-01)

Find Component: list1.obj
Using Map Number: 84
The ComponentRank: (0.0E+00,6.7E-01)

Find Component: bag.obj
Using Map Number: 84
The ComponentRank: (0.0E+00,6.7E-01)

The result from this execution indicates that Set is the best component. It has a full match value for Keyword and Semantic Match Ratios. The component such as Array is partially matched. List1, List2, and Bag are the worst ones because they fail the semantic check. That is why their ComponentRank is 0.0.

To convince ourselves that these components retrieved above are indeed the
correct ones, all twelve components are selected and executed against the SignatureMatching and Ground Equations. The results are correlated with the previous execution to check that we did not discard any potential candidate component. Here is the result of this execution:

The result of your retrieval session is:

Find Component: set.obj
Using Map Number: 1
The ComponentRank: (1.0E+00, 1.0E+00)

Find Component: array.obj
Using Map Number: 37
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: bint1.obj
Using Map Number: 1
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: deque.obj
Using Map Number: 42
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: ring.obj
Using Map Number: 80
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: list2.obj
Using Map Number: 83
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: list1.obj
Using Map Number: 83
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: bag.obj
Using Map Number: 63
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: queue.obj
Using Map Number: 42
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: bint2.obj
Using Map Number: 42
The ComponentRank: (0.0E+00, 6.7E-01)
Find Component: stack1.obj
Using Map Number: 42
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: stack2.obj
Using Map Number: 23
The ComponentRank: (0.0E+00, 6.7E-01)

This result confirms that the components which have not been selected in the previous execution, such as Bint1, Stackt2, Stack1, Bint2, Queue, Ring, and Deque have a SemanticMatchRatio of 0.0 because no translated equation could be produced for them. These components do not have a profile value of 330 or 3211. That is why they are excluded from the first execution. We have eliminated the useless components which would be wasteful and costly (in terms of execution time) if they had been included in the retrieval session. This elimination makes the method useful and practical when the number of software components in the library becomes large.

Our final study with the software library is to see how a partial match of a component can be obtained. So far, for demonstration purposes, we only have one ground equation. Suppose now, a user adds in another ground equation as follows:

Eq: Subset(Insert(1, Insert(2, Create)),
    Insert(1, Insert(3, Insert(2, Create)))) = true.

With this new ground equation, an additional set of profiles is computed. This set has the following elements: 3210, 3211, and 110. We begin to search for the Candidate-Components in the library; this set of value will be checked against the partitions in the library. If any component which belongs to a partition that contains either \{3210, 3211, 110\} or \{330, 3211, 110\} (the previous ground equation) will be considered as a candidate component. In this case, looking at Figure 5, the components Stack1, Queue, Ring, Deque,
and Bint2 will be considered as candidate components in addition to the components selected in the last session. Here is the execution to verify this:

The result of your retrieval session is:

Find Component: set.obj
Using Map Number: 1
The ComponentRank: (2.0E+00, 1.0E+00)

Find Component: array.obj
Using Map Number: 1
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: deque.obj
Using Map Number: 42
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: queue.obj
Using Map Number: 83
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: bint2.obj
Using Map Number: 74
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: stack1.obj
Using Map Number: 83
The ComponentRank: (1.0E+00, 6.7E-01)

Find Component: ring.obj
Using Map Number: 80
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: list2.obj
Using Map Number: 83
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: list1.obj
Using Map Number: 83
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: bag.obj
Using Map Number: 63
The ComponentRank: (0.0E+00, 6.7E-01)

Find Component: stack1.obj
The ComponentRank: (0.0E+00, 6.7E-01)

From this example, we can see that additional components are included due to the
second ground equations. It turns out that these components have a very low ranking and may not contribute much to a user's expectation. However, the system should be able to provide a user with such partial matches in case a user is not satisfied with the better matches. Since a user has control over the search process via the selection criteria, he/she can specify how well the components are matched against his/her query. This would limit the number of matched components to a manageable size so that a user can make a selection with ease. The execution above was executed using a KPS value of 0.0 to 1.0 to allow all components to be evaluated. If a user has chosen a KPS of 0.8 to 1.0, then only the Set component would show up.

F. RETRIEVAL PERFORMANCE

In this section we perform the retrieval measurement for this model. Here we do not try to do an extensive large scale measurement but rather a study to see how the system stands up against two well known critical evaluations [4]. They are Recall and Precision. Recall is defined as:

\[
Recall = \frac{|RelevantComponentsRetrieved|}{|RelevantComponentsInLibrary|}
\]

Precision is defined as:

\[
Precision = \frac{|RelevantComponentsRetrieved|}{|ComponentsRetrieved|}
\]

1. Experimental Description

There are 18 different query sessions performed for this experiment using the same simulated software library discussed in the previous experiment. These sessions consist of
6 different query formulations using 3 different KPS scenarios. The 6 queries are designed to retrieve the following components: Bag, List, Queue, Set, Ring, and Stack. Each query consists of a set of keywords, operations, and a ground equation that characterizes an individual component. The 3 KPS scenarios are chosen with Low (0.0-1.0), Medium (0.6-1.0), and High (0.9-1.0). Once a retrieval session is complete, the ComponentRanks are tabulated in Table 7. For each query, the retrieved components are evaluated to determine their relevance and cross-checked against the expected ones. Finally, the recall and precision values are computed for this query session. They are also tabulated in the same table (column 2). Note that the left most column of Table 7 identifies the a user query. The “-1”, “-2”, and “-3” represents the KPS scenarios: Low (0.0-1.0), Medium (0.6-1.0), and High (0.9-1.0).
<table>
<thead>
<tr>
<th>Query Specification</th>
<th>Recall/Precision</th>
<th>Stack 1</th>
<th>Stack 2</th>
<th>Bint1</th>
<th>Bint2</th>
<th>Queue</th>
<th>Set</th>
<th>Array</th>
<th>Bag</th>
<th>List1</th>
<th>List2</th>
<th>Ring</th>
<th>Deque</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bag-1</td>
<td>1.0 / 0.1</td>
<td>0.53</td>
<td></td>
<td>0.53</td>
<td>0.53</td>
<td>0.53</td>
<td>0.53</td>
<td>2.0</td>
<td>0.53</td>
<td>0.53</td>
<td>0.53</td>
<td>0.4</td>
<td>0.53</td>
</tr>
<tr>
<td>Bag-2</td>
<td>1.0 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bag-3</td>
<td>1.0 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List-1</td>
<td>1.0 / 0.5</td>
<td>1.3</td>
<td>1.3</td>
<td>0.38</td>
<td>1.3</td>
<td></td>
<td>0.5</td>
<td>0.17</td>
<td>2.0</td>
<td>2.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>List-2</td>
<td>1.0 / 0.71</td>
<td>1.3</td>
<td>1.3</td>
<td></td>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td>2.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>List-3</td>
<td>0.4 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td>2.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>Queue-1</td>
<td>1.0 / 0.54</td>
<td>1.0</td>
<td>1.3</td>
<td>0.2</td>
<td>1.3</td>
<td>2.0</td>
<td>1.3</td>
<td>0.5</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Queue-2</td>
<td>1.0 / 0.75</td>
<td></td>
<td>1.3</td>
<td>1.3</td>
<td>2.0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Queue-3</td>
<td>0.16 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ring-1</td>
<td>1.0 / 0.11</td>
<td>0.3</td>
<td>0.53</td>
<td>0.67</td>
<td>0.53</td>
<td>0.4</td>
<td>0.4</td>
<td>0.53</td>
<td>2.0</td>
<td>0.53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ring-2</td>
<td>1.0 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ring-3</td>
<td>1.0 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set-1</td>
<td>1.0 / 0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td>0.44</td>
<td>1.3</td>
<td>1.3</td>
<td>0.44</td>
</tr>
<tr>
<td>Set-2</td>
<td>1.0 / 0.66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td>1.3</td>
<td>1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set-3</td>
<td>0.5 / 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Recall/Precision Measurements for 3 Different KPS scenarios
<table>
<thead>
<tr>
<th>Query Specification</th>
<th>Recall/Precision</th>
<th>Stack 1</th>
<th>Stack 2</th>
<th>Bint1</th>
<th>Bint2</th>
<th>Queue</th>
<th>Set</th>
<th>Array</th>
<th>Bag</th>
<th>List1</th>
<th>List2</th>
<th>Ring</th>
<th>Deque</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack-1</td>
<td>1.0 / 0.5</td>
<td>2.0</td>
<td>2.0</td>
<td>0.38</td>
<td></td>
<td>1.3</td>
<td></td>
<td>0.5</td>
<td>0.17</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>Stack-2</td>
<td>1.0 / 0.71</td>
<td>2.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>Stack-3</td>
<td>0.4 / 1.0</td>
<td>2.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td>1.3</td>
<td></td>
<td></td>
<td></td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 7: Recall/Precision Measurements for 3 Different KPS scenarios (Continue)
2. **Experimental Evaluation**

We showed the recall and precision performance of the 18 retrieval sessions using the 3 different KPS scenarios in Table 7. This table indicates that for a Low scenario KPS, we have a very high recall for all queries, a perfect score of 1.0 for all queries. However, its precision is very low: it starts at 0.1 to 0.5. This is due to the fact that we have stressed coverage. Here we have considered very minimal partial matches as well as full matches. In opposite, for a High KPS scenario, we had a high precision performance but we lost some recall performance at the same time. This is due to the fact that we are have asked the system to look only for full matches. The precision values have a perfect score of 1.0 and the recall measurement is in the range of 0.16 to 1.0. Balance recall and precision is achieved using the Medium KPS scenario. From Table 7, we see that the recall values stays constant at 1.0 while the precision varies from 0.66 to 1.0. This is a better selection criterion since it balances between our recall and precision measurements.

In comparing with a study by [14], for precision, we see that the High KPS scenario is in the same class with Faceted approach\(^\text{16}\). However, it recall performance is outperformed by Faceted approach by a small factor of 0.15 on the low side. Our Medium KPS has a better recall performance but has a lower precision than the Faceted approach. It is about the same class as the Attribute-Value approach.

\(^{16}\text{Faceted approach is ranked highest in precision among Attribute-value, Enumerated, Keyword. Note here that we are not considering the analysis of variance for precision as stated in [14].}\)
We note that the study in [14] is a much bigger scale evaluation. However, we can see that this study gives us some indications about the performance of the system as well as how our selection criteria can effect the precision and recall measurement of the system. In addition to this, we also conclude that, through a conservative selection criterion (Medium KPS scenario), we can possibly obtain a balanced recall and precision retrieval which improve on the recall and precision obtainable by other search techniques. This is due to the fact that we allow a user to have control over the search process through the selection criterion option. This also fits to our assumption that search can be made an iterative process.
IX. SUMMARY AND SUGGESTIONS FOR FUTURE RESEARCH

A. INTRODUCTION

This chapter summarizes the contents of the dissertation, identifying those areas that are contributions to the state of the art, and then offers suggestions for future research. Section B contains the dissertation summary. Section C discusses system modifications to improve its performance. Section D describes the system extensions and suggestions for future research. Section E describe changes to the implementation required to get a production quality realization of the method that can be integrated with the CAPS system.

B. DISSERTATION SUMMARY

This dissertation has described in detail a technique for retrieving reusable software components from a software library using a partial specification. A prototype has been built with an intention to improve and port it to the Computer Aided Prototyping System (CAPS). The goal of CAPS is to provide the software engineer with an environment to support rapid prototyping for hard real embedded systems. CAPS uses a prototype language PSDL to specify both prototypes and production software. The retrieval reusable components can help to improve the prototyping process and produce quality code.

In our approach, the search for software components is organized as a series of increasingly stringent filters on software library components. We first filter components by comparing pre-computed syntactic profiles of components with the profile of the query. The inclusion relation from a Hasse diagram has been used to facilitate this retrieval process. The result of this process is a set of candidate components with their Profile and Key-
Secondly, we filter these components by seeking to find type-consistent translations from the query signature to the component signatures. This is accomplished by signature matching, which looks for maps that translate the sort and operation symbols of the query into corresponding sort and symbols of candidate components. This matching process also uses profiles to reduce the computational effort required. Signature matches calculated can be partial, in that only part of the functionality the user seeks may actually be available. Third, the semantic filtering ranks components by how well they satisfy the ground equations in the query and eliminates the components that do not satisfy any of the ground equations. In this process, equations that are logical consequences of the query specification are translated through the signature matches into equations whose validity is checked with respect to the candidate specifications. Finally, the candidates in the choice set are ranked according to their likelihood of success. The final output represents this choice set as well as how it is computed. Invalid information is also reported to help a user to reformulate a new query if needed. This whole process can be made iterative.

This dissertation makes contributions to the state of the art in reusable software component retrieval.

- A theory of query by partial specification through a multi-level search of software components by comparing between specifications based on syntactic (profile/keyword, signature matching) and semantic information.

- A new method, algorithms, and corresponding implementation that optimizes the signature matching process (for full and partial matches) to produce a set of optimized maps (no duplication or sub-maps). This method is also considers subsort relation for handling component matching.

- A new model for organization of a software library using a Hasse diagram to facilitate of the retrieval process. This model can play an important role in supporting the cataloging of components through semi-automation.
• A new multi-level ranked scheme to order components in terms of their closeness to the user query.

• A new and unique way for classification of program operations in terms of the profiles for approximate signature matching and classification of software components.

• Evidence that a large scale reuse is feasible, avoiding the limitations of informal methods (i.e. using keywords or facets) and complexity of formal methods (having the user query using a complete formal specification). The system proposed also can handle both non-generic as well as generic software components. This is a real practical requirement for a retrieval system. The system can also provide useful data to help the user to formulate a new query if needed.

• The ideas proposed in this dissertation may have implications for software testing. In particular, the techniques for generating test cases to be used in matching queries could also be used to test the correctness of code that is supposed to implement a module.

C. SYSTEM MODIFICATIONS TO ENHANCE PERFORMANCE

This section suggests the modifications that would help the system to improve its performance. These modifications should be relatively easy to implement, and they require only simple verifications.

1. User Interface

As a prototype, the current prototype system has a very limited user interface capability. It would be nice to develop a front-end graphical user interface for the system. On top of this, a syntax directed editor can be used to assist a user with the query formulation process. This would also assure that the partial specification entered by a user is syntactically correct before the retrieval process is started. The output of the prototype should also be improved and automated so that the user can view the desired retrieval information and the selected components with ease. Finally, it would be a good idea to store previous que-
ries in a database so that a user can reuse them in the future. The verification here is to check for the system for the ability to input query via menu. The output of the retrieval process can also be used to verify for input data.

2. **Incremental Retrieval of Software Components**

The current prototype can work with a single selected block of signature maps (85 maps) per execution. For a production system, it should allow a user to backtrack or go forward to any particular block of maps desired. The same approach should be used for allowing the user to view components with any KPS value. This option would add more flexibility for a user in controlling the search process. To verify this process, we can evaluate the output of the retrieval process such as the map ids and the Keyword, Profile, and Signature Match Ratios, and ComponentRank of the retrieved components.

3. **Loading of OBJ3 Environment**

For the semantic filtering process, the current implementation load OBJ3 environment every time a component is being checked for its ground equation. This loading can take some additional time off from the retrieval speed. For the purpose of efficiency, this OBJ3 environment should be loaded only one time during the retrieval process. To verify this, we can look at the OBJ3 process id and status as it executes. The Unix command “ps” would allow us to do that.

**D. SUGGESTIONS FOR FUTURE RESEARCH**

1. **Environment for Evaluation of Test Cases in Queries**

The current prototype uses OBJ3 ‘s term rewriting capability to evaluate the ground equations. A suggestion for future research is to compare the cost these test cases
to the cost of directly in compiled code for the component implementations. Since the number of test cases per query is likely to be small and since new code will have to be generated, compiled, and loaded to invoke the components chosen by semantic matching, this overhead could overwhelm the speed advantage of compiled code over term rewriting, especially since one call to OBJ3 could handle a batch of several specifications with their reductions. The costs involved here should be checked experimentally. The difference between these approaches should be measured to help decide if it is worth the effort of associating formal specifications with each component in the software base.

2. The Choice for Rank Functions

So far all ground equations are assumed to have equal importance. Some equations that may be more significant to a user than the others. For such situations, we can attach weights reflecting the relative importance of different equations. This should be evaluated experimentally. The proper weighting of the importance of the number of equations that match for a given operation relative to the number of operations that are supported by a given number of test cases should also be further explored. Using the relation $=$ in section 4.4 means that some operation matches will be better that others; the definition of match ratio of the signature match could be modified to take account of this, and then it would be interesting to see if this helps with retrieval.

3. Possible Improvements for the Signature Matching Algorithm

The current SignatureMatching algorithm can match a query constant to a component constant as long as their sort assignment is allowable. In the experiment components, there are two kind of constants. They are generator constants and exception error con-
constants. During the course of testing and evaluation of the SignatureMatching algorithm, many useless maps that were created because the system tries to match a query generator constant with an exception constant from the component or vice versa. This kind of matching is unrealistic, and it should not be included in any signature maps. We should only match query generator constants with the generator constants from the component. The same would be true for exception error constants. One possible way to improve the matching process is to have a user explicitly declare exception constants in the specification (the Ada programming language also requires users to declare exceptions in the specifications). Given this information, the SignatureMatching algorithms could know the difference between the two and would not try to cross-assign them with one another.

The second possible improvement is to order the query and component operations in some fashion using their profile values. From here the SignatureMatching process can sequentially step through these ordered operations and match them. This would reduce the number of combinations to be checked. We should note here that this matching process may not be linear due to multiple operations with same profile involved from the query and component (i.e., duplication of operation’s profiles in the query but not in the component). This needs experimentally evaluated and compared with the current approach.

4. Experimentation with other Components from other Libraries

The experiments completed so far indicate that it works well with small subset of the Booch library components, namely Data Structure components. The author believes that additional testing should be performed in a larger setting and include more variety of
components from a different domains such as navigation, business, and mathematical components. Using this setting, more available data can be collected and compiled for recall and precision measurements.

5. Exploring other Specification Languages

In addition to OBJ3, it would be useful to explore using other languages to represent the formal specifications. For example, it would be interesting to see if the techniques suggested here would work for FOOPS, which is an extension of OBJ that handles states, and for Eqlog, which extends OBJ with some features of logic programming. It would be interesting to assess the value of Eqlog for matching, and to explore the trade-offs between expressive power and computational requirements. Eqlog has been implemented by Diaconescu [1] as an extension of the OBJ3 interpreter.

E. SUGGESTION CHANGES TO GET A PRODUCTION QUALITY

To get a production quality of the current implementation, the following points are recommended:

- Replace integer constants in Gldef.a and Swbdef.a with enumeration types for ease of maintenance and debugging purposes. Replace string variables from bounded size to unbound size.
- Incorporate the actual library system to replace the simulated Hasse diagram modules swb.a and swbdef.a.
- Incorporate the user interface procedure Getquery with a more friendly graphical user interface module. Also replace the module SortDisplayResult to output text data in a friendly graphical form. This would provide a better interaction between the system and user.
- Re-implement the current design so that OBJ3 environment can be load only once for each query session. This would save some execution time.
- The MBN option could be removed. In this way the system would require to
search through all possible maps for the best possible maps instead stopping at current limit which is 85 maps. This would be a better solution to current implementation since a user is provide less input with the query.

- Include an option to allow a user to view all the signature maps from the log files (component name with extension “tc”) for evaluation if he/she desires so. These signature maps can be also incorporated as inputs to a wrapper program for the CAPS prototype.
REFERENCES


[10] Joseph Goguen, Claude Kirchner, Helene Kirchner, Aristide Megrelis, and Jose Meseguer. An introduction to OBJ3. In Jean-Pierre Jouannaud and Stephane Kaplan,


APPENDIX A - BACKGROUND INFORMATION

In this chapter we provide some background information and necessary definitions. These materials are not intended as a tutorial, but rather to indicate what the reader needs to know in understanding the technical aspects of our work, especially in Chapter IV. Interested readers can refer to [10,15,16,17] for more information.

A. OBJ3 AND ALGEBRAIC SPECIFICATION

OBJ3 is a functional programming language rigorously based on order sorted logic and can be used to describe the syntactic and semantic properties of sequential processes. The rigorous semantics of the language also allows specifications to be written as programs that are declarative in style and mirror the structure of an algebraic specification. This property makes it possible for OBJ3 to be used as a theorem prover for validating and implementing algebraic specifications. Finally, OBJ3 supports parameterized programming which is one of the powerful features that support our work in handling generic modules.

In OBJ3, an algebraic specification for objects consists of two parts: a signature and a set of axioms. The signature defines the sorts (or types) being specified, the operation symbols, and their functionality in an object. It is denoted as \((S, \Sigma)\) where \(S\) and \(\Sigma\) are a sort set and an operation symbol set, respectively. The axioms are expressed as equations describing the semantics (or behavior) of an object.

B. GROUND EQUATION

In our user query specification, an equation which contains no variables is referred to as a ground equation. It illustrates an example of an object’s behavior by describing
how the operations are interacted with one another using only appropriate ground terms, we assume that the set of ground equations in the query has distinct left-hand sides and is complete, in the sense of being terminating and Church-Rosser; it does not need to be “complete” in the sense of completely describing the desired component.

C. TERM REWRITING

In order that a specification can be executed to determine that the intended properties follow the stated axioms, a technique called Term Rewriting is used. In term rewriting, each axiom is interpreted as a left-to-right rewrite rule that states the left hand side (LHS) can be rewritten to its corresponding right hand side (RHS). Given terms $t_i$, $t_j$, and $t_k$, a set $\Psi$ of rewrite rules is Church-Rosser if whenever

$$t_i \Rightarrow * t_j \text{ and } t_i \Rightarrow * t_k$$

then there is a term $t_l$ such that $t_i \Rightarrow * t_l$ and $t_k \Rightarrow * t_l$, where $\Rightarrow *$ denotes successive application of rewrite rules. $\Psi$ is terminating if there is no infinite chain of rewrite applications. $\Psi$ is canonical if it is Church-Rosser and terminating. Given term $t$ and $t'$, if $(t \Rightarrow * t')$ and $t'$ can not be further reduced through any of rewrite rules, then $t'$ is referred to as the normal form of $t$. For canonical specification, each ground term has a unique normal form, called its canonical form. Term rewriting is implemented in OBJ3 in terms of a feature called “Reduce” that reduces an expression to its normal form.

D. OPERATIONS

Let $A$ be a nonempty set, then the $n$-ary operation $op$ is a function from $A^n$ to $A$ and is written $op$: $A^n \rightarrow A$. A nullary operation $op: \rightarrow A$, corresponds to a constant
value that is a member of the set A while a unary operation is a function from A into A, that is \( \text{op}: A \rightarrow A \). Operations are also referred to as Functions. [38]

E. ACCESSOR AND CONSTRUCTOR OPERATIONS

An operation whose range sort that is not a principle sort is defined to be an accessor operator. For example, in our stack example, the Top operation is an accessor since it does not return a result of sort stack (the main sort). An operation whose range sort that is a principal sort is defined to be a constructor operation. The values of an abstract data type are built up using constructor operations. Constructors can further break down into atomic and nonatomic operators. For example, in a Stack component, operations create and push are atomic constructors while the pop operator is a nonatomic constructor operator. The rationale here is that whatever the value constructed by the pop can also be performed by using push and create operators.
APPENDIX B - OBJ3 COMPONENTS

This appendix contains the OBJ3 source code for the twelve Booch data structure Components. They are as follows:

************************************************************************
*** This is a list1 obj3 (Generic Version)
*** Keywords: Booch, Data-Structure, List
*** Function: To perform list data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototype.
************************************************************************

obj listl [X :: TRIV] is sort List.

  protecting NAT.
  op nil : -> List.
  op cons : Elt List -> List.
  op headof : List -> Elt.
  op tailof : List -> List.
  op clear : List -> List.
  op clearhead : List -> List.
  op contains : List Elt -> Bool.
  op lengthof : List -> Nat.
  op sethead : Elt List -> List.
  op isnull : List -> Bool.
  op isequal : List List -> Bool.
  op listerror : -> List.
  op elterror : -> Elt.
  op copy : List List -> List.

  var I, J : Elt.
  var L : List.
  var Nl : List.

  eq copy(nil,nil) = nil.
  eq copy(L,nil) = L.
  eq copy(nil,L) = nil.
  eq copy(cons(J,L),cons(I,Nl)) = cons(J,copy(L,Nl)).

  eq lengthof(nil) = 0.
  eq lengthof(cons(I,L)) = 1 + lengthof(L).

  eq isequal(nil,nil) = true.
  eq isequal(L,nil) = if not lengthof(L) == 0 and lengthof(nil) == 0
    then false else true fi.
  eq isequal(nil,Nl) = if lengthof(nil) == 0 and not lengthof(Nl) == 0
    then false else true fi.
  eq isequal(cons(J,L),cons(I,Nl)) = if J == I and isequal(L,Nl)
    then true else false fi.
eq clear(L) = nil.

eq clearhead(nil) = nil.
eq clearhead(cons(I,L)) = L.

eq isnull(L) = if lengthof(L) == 0 then true else false fi.

eq headof(nil) = elterror.
eq headof(cons(I,L)) = I.

eq tailof(nil) = nil.
eq tailof(cons(I,L)) = L.

eq sethead(I,nil) = listerror.
eq sethead(I,L) = cons(I,clearhead(L)).

eq contains=nil(I) = false.
eq contains(cons(J,L),I) = if J == I then true else contains(L,J) fi.

endo

************************************************************************
*** This is a list2 obj3 (List of Natural number)
*** Keywords: Booch, Data-Structure, List
*** Function: To perform list data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototpe.
************************************************************************

obj list is sort List.

protecting NAT.
op nil : -> List.
op cons : Nat List -> List.
op headof : List -> Nat.
op tailof : List -> List.
op clear : List -> List.
op clearhead : List -> List.
op contains : List Nat -> Bool.
op lengthof : List -> Nat.
op sethead : Nat List -> List.
op isnull : List -> Bool.
op isequal : List List -> Bool.
op listerror : -> List.
op elterror : -> Nat.
op copy : List List -> List.

var I, J : Nat.
var L, N : List.

eq copy(nil,nil) = nil.

104
eq copy(L,nil) = L.
eq copy(nil,L) = nil.
eq copy(cons(J,L),cons(I,Nl)) = cons(J,copy(L,Nl)).
eq lengthof(nil) = 0.
eq lengthof(cons(I,L)) = 1 + lengthof(L).

eq isequal(nil,nil) = true.
eq isequal(L,nil) = if not lengthof(L) == 0 and lengthof(nil) == 0
then false else true fi.
eq isequal(nil,Nl) = if lengthof(nil) == 0 and not lengthof(Nl) == 0
then false else true fi.
eq isequal(cons(J,L),cons(I,Nl)) = if J == I and isequal(L,Nl)
then true else false fi.
eq clear(L) = nil.
eq clearhead(nil) = nil.
eq clearhead(cons(I,L)) = L.

eq isnull(L) = if lengthof(L) == 0 then true else false fi.
eq headof(nil) = elterror.
eq headof(cons(I,L)) = I.

eq tailof(nil) = nil.
eq tailof(cons(I,L)) = L.

eq sethead(I,nil) = listerror.
eq sethead(I,L) = cons(I,clearhead(L)).

eq contains(nil,I) = false.
eq contains(cons(J,L),I) = if J == I and isequal(L,Nl)
then true else contains(L,I) fi.

end

***********************************************************************
*** This is a bintl obj3 (Binary Tree Data structure version 1)
*** Keywords: Booch, Data-Structure, Binary-Tree
*** Function: To perform binary tree data structure operations.
*** Component History: Components from CS4520 Project
*** Modify syntax to work with SCS prototpe.
***********************************************************************

object bintl[X :: TRIV] is sort Bintree .
  protecting NAT .
  protecting INT .
  op empty : -> Bintree .
  op left : Bintree -> Bintree .
  op right : Bintree -> Bintree .
  op isempty : Bintree -> Bool .
  op node : Bintree -> Elt .
  op isin : Elt Bintree -> Bool .
  op naterror : -> Elt .

105
op make : Elt Bintree Bintree -> Bintree.
op max : Nat Nat -> Nat.
op depth : Bintree -> Nat.
op height : Bintree -> Nat.

var m : Elt.
var l,r : Bintree.
var n : Elt.
var i : Nat.
var j : Nat.

eq left(empty) = empty.
eq left(make(n,l,r)) = 1.

eq right(empty) = empty.
eq right(make(n,l,r)) = r.

eq node(empty) = naterror.
eq node(make(n,l,r)) = n.

eq isempty(empty) = true.
eq isempty(make(n,l,r)) = false.

eq isin(n,empty) = false.
eq isin(m,make(n,l,r)) = if n == m then true else isin(m,l) or isin(m,r) fi.

eq max(i,j) = if i > j then i else j fi.

eq depth(empty) = 0.
eq depth(make(n,l,r)) = 1 + max(depth(l), depth(r)).

eq height(make(n,l,r)) = depth(make(n,l,r)) - 1.

endo

objbint2[X::TRIV] is
sorts Tree Child Node .
protecting NAT .
subsort Node < Tree .

op null -> Tree .
op left : -> Child .
op right : -> Child .
op node : Elt Tree Tree -> Tree .
op copy : Tree Tree -> Tree.
op clear : Tree -> Tree.
op construct : Elt Tree Child -> Tree.
op setitem : Elt Tree -> Tree.
op swapchild : Child Tree Tree -> Tree.
op isequal : Tree Tree -> Bool.
op isnull : Tree -> Bool.
op itemof : Tree -> Elt.
op childof : Child Tree -> Tree.
op treeisnull : -> Tree.
op treeisnull : -> Elt.

var T T1 T2 T3 T4 : Tree.
var C C1 C2 : Child.
var I I1 I2 : Elt.

eq copy(T2(T1, T2)) = T1.
eq clear(T) = null.
eq construct(I,T,C) = if C == right then node(I,null,T)
  else node(I,T,null) fi.
eq setitem(I,null) = treeisnull.
eq setitem(I2,node(I,T2)) = node(I2,T1,T2).
eq swapchild(C,node(I,T1,T2),T3) =
  if C == left then node(I,T3,T2) 
  else node(I,T1,T3) fi.
eq swapchild(C,node(I,T1,T2),T3) =
  if C == left then T1 
  else T2 fi.
eq isequal(T1 ,T2) = if T1 == T2 then true else false fi.
eq isnull(T) = if T == null then true else false fi.
eq itemof(null) = treeisnull.
eq itemof(node(I,T1,T2)) = I.
eq childof(C,null) = treeisnull.
eq childof(C,node(I,T1,T2)) = if C == left then T1 else T2 fi.

obj LOCATION is
  sort Location.
  ops Front Back : -> Location.
endo

obj deque[X :: TRIV] is
  sort Deque.

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

*** This is a deque obj3 (Deque Data structure)
*** Keywords: Booch, Data-Structure, Deque
*** Function: To perform deque data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototype.

obj LOCATION is
  sort Location.
  ops Front Back : -> Location.
endo

obj deque[X :: TRIV] is
  sort Deque.
*** Import predefined types
  protecting LOCATION.
  protecting NAT.

*** Generator
  op empty : -> Deque.

*** Other constructors/properties
  op copy : Deque Deque -> Deque.
  op clear : Deque -> Deque.
  op add : Elt Location Deque -> Deque.
  op pop : Location Deque -> Deque.
  op isequal : Deque Deque -> Bool.
  op lengthof : Deque -> Nat.
  op isempty : Deque -> Bool.
  op frontof : Deque -> Elt.
  op backof : Deque -> Elt.

*** Exception
  op underflow : -> Deque.
  op underflow : -> Elt.

*** Variable declarations
  var D D2 : Deque.
  var L : Location.
  var E : Elt.

*** Equations
  eq copy(empty,D) = empty.
  eq copy(add(E,Front,D),D2) = add(E,Front,copy(D,D2)).
  eq clear(D) = empty.
  eq add(E,Back,D) = if D == empty then add(E,Front,empty) else
                          add(frontof(D),Front,add(E,Back,pop(Front,D))) fi.
  eq pop(L,empty) = underflow.
  eq pop(Front,add(E,Front,D)) = D.
  eq pop(Back,add(E,Front,D)) = if D == empty then D else
                               add(E,Front,pop(Back,D)) fi.
  eq isequal(D,D2) = if D == D2 then true else false fi.
  eq lengthof(empty) = 0.
  eq lengthof(add(E,L,D)) = 1 + lengthof(D).
  eq isempty(D) = if D == empty then true else false fi.
  eq frontof(empty) = underflow.
  eq frontof(add(E,Front,D)) = E.
  eq backof(empty) = underflow.
eq backof(add(E,Front,D)) = if D == empty then E else
backof(pop(Front,D)) fi.

endo
************************************************************************
*** This is a Set obj3 (Set Data structure)
*** Keywords: Booch, Data-Structure, Set
*** Function: To perform Set data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototpe.
************************************************************************

obj set[X :: TRIV] is sort Set.
  protecting NAT.

*** basic set operations

*** constructors

  op create : -> Set.
  op clear : Set -> Set.
  op copy : Set Set -> Set.
  op add : Elt Set -> Set.
  op remove : Elt Set -> Set.
  op union : Set Set -> Set.
  op intersection : Set Set -> Set.

*** assessors

  op cardinality : Set -> Nat.
  op isequal : Set Set -> Bool.
  op isempty : Set -> Bool.
  op memberof : Elt Set -> Bool.
  op subsetof : Set Set -> Bool.
  op prossubsetof : Set Set -> Bool.

*** exception

  op seterror : -> Set.

*** variables declaration

  var S S1 S2 S3 S4 : Set.
  var E E1 E2 : Elt.

*** axioms

  eq clear(S) = create.
eq cardinality(create) = 0.
eq cardinality(add(E,S)) = if memberof(E,S) then cardinality(S) else 1 + cardinality(S) fi.

eq remove(E,create) = seterror.
eq remove(E,add(E_1,S_1)) = if E == E_1 then S_1 else union(add(E,create),remove(E,Sl)) fi.

eq subsetof(create,S_1) = true.

*** eq subsetof(add(E_1,create),add(E_2,create)) = if E_1 == E_2 then true else false fi.
eq subsetof(add(E_1,S_1),add(E_2,S_2)) = if memberof(E_1,add(E_2,S_2)) then subsetof(S_1,add(E_2,S_2)) else false fi.

eq pros subsetof(S_1,S_2) = if subsetof(S_1,S_2) and cardinality(S_2) > cardinality(S_1) then true else false fi.

eq memberof(E,create) = false.
eq memberof(E,add(E_1,S_1)) = if E == E_1 then true else memberof(E,S_1) fi.

eq isequal(create,create) = true.
eq isequal(add(E,S_1),add(E,S_2)) = if S_1 == S_2 then true else false fi.

eq copy(S_1,S_2) = S_1.

eq isempty(create) = true.
eq isempty(add(E,S_1)) = false.

eq union(create,S) = S.
eq union(S,create) = S.
eq union(add(E_1,S_1),S_2) = if memberof(E_1,S_2) then union(S_1,S_2) else add(E_1,union(S_1,S_2)) fi.

eq intersection(create,S) = create.
eq intersection(S,create) = create.
eq intersection(add(E_1,S_1),S_2) = if memberof(E_1,S_2) then add(E_1,intersection(S_1,S_2)) else intersection(S_1,S_2) fi.

endo
This is a bag obj3 (bag Data structure)

Keywords: Booch, Data-Structure, Bag

Function: To perform bag data structure operations.

Component History: Components from CS4520 Project.

Modify syntax to work with SCS prototype.

obj bag[X :: TRIV] is sort Bag .
protecting NAT .

*** CONSTRUCTORS:

op copy   : Bag Bag -> Bag .

*** Empty the bag:
op clear : Bag -> Bag .

*** Empty bag (constant)
op empty : -> Bag .

*** Represents a singleton bag:
op singleton : Elt -> Bag .

*** Adds an element to a bag:
op add : Elt Bag -> Bag .

*** Removes an element from a bag:
op remove : Elt Bag -> Bag .

*** The following 3 ops represent the bags union, intersection, and difference
*** respectively:
op union : Bag Bag -> Bag .
op and : Bag Bag -> Bag .
op diff : Bag Bag -> Bag .

***

SELECTORS:

*** The following operation returns the total number of elements in a bag.
*** More than one occurrences of an element is allowed in a bag.
op lengthof : Bag -> Nat .

*** The uniqueExtentOf operation returns the number of elements of a bag
*** counting only one occurrence of each element if there are more than
*** one (The bag is treated like a set by this operation):
op uniqueExtentOf : Bag -> Nat .

*** The numberOf operation gives the number of occurrences of a given
*** element in the bag:
op numberof : Elt Bag -> Nat.

*** The following operations are self explanatory:
op in : Elt Bag -> Bool.
op isequal : Bag Bag -> Bool.
op subset : Bag Bag -> Bool.
op propersubset : Bag Bag -> Bool.
op isempty : Bag -> Bool.

***

*** Exception:
op bagerror : -> Bag.

***

*** SUPPORTING OPERATIONS:

*** A bag supporting operation used by most other bag operations:
op _+_ : Bag Bag -> Bag.

*** A bag supporting operation used to remove only the first occurrence of
*** an element that belongs to the bag:
op removeOne : Elt Bag -> Bag.

***

*** VARIABLES:

var B B1 B2 : Bag.
vars E E1 : Elt.
var X : Elt.

***

*** AXIOMS:
eq empty + B = B.
eq B + empty = B.
eq copy(B1,B2) = B1.
eq clear(B) = empty.
eq add(X,B) = singleton(X) + B.
eq \text{in}(X, \text{empty}) = \text{false}.
eq \text{in}(X, \text{singleton}(E)) = (X == E).
eq \text{in}(X, (\text{singleton}(E) + B)) = \text{if } X == E \text{ then true else } \text{in}(X, B) \text{ fi}.

eq \text{and}(\text{empty}, B) = \text{empty}.
eq \text{and}(\text{singleton}(E), B) = \text{if } \text{in}(E, B) \text{ then singleton}(E) \text{ else } \text{empty} \text{ fi}.
eq \text{and}(\text{singleton}(E) + B, B1) = \text{if } \text{in}(E, B1) \text{ then singleton}(E) + \text{and}(\text{removeOne}(E, B), \text{removeOne}(E, B1)) \text{ else } (B, B1) \text{ fi}.
eq \text{union}(\text{empty}, B) = B.
eq \text{union}(\text{singleton}(E), B) = (\text{singleton}(E) + B).
eq \text{union}(\text{singleton}(E) + B, B1) = \text{singleton}(E) + \text{union}(B, B1).

eq \text{diff}(\text{empty}, B) = \text{empty}.
eq \text{diff}(\text{singleton}(E), B) = \text{if } \text{in}(E, B) \text{ then } \text{singleton}(E) \text{ else } \text{empty} \text{ fi}.
eq \text{diff}(\text{singleton}(E) + B, B1) = \text{if } \text{numberof}(E, B) < \text{numberof}(E, B1) \text{ then } \text{diff}(B, B1) \text{ else } \text{singleton}(E) + \text{diff}(B, B1) \text{ fi}.

eq \text{isempty}(B) = (B == \text{empty}).
eq \text{isequal}(\text{empty}, B) = \text{false}.
eq \text{isequal}(B, \text{empty}) = \text{false}.
eq \text{isequal}(\text{singleton}(E), \text{singleton}(E1)) = \text{if } E == E1 \text{ then true else false fi}.
eq \text{isequal}(\text{singleton}(E), B) = \text{if } \text{in}(E, B) \text{ and } (\text{lengthof}(B) == 1) \text{ then true else false fi}.
eq \text{isequal}(\text{singleton}(E) + B, B1) = \text{if } \text{numberof}(E, (\text{singleton}(E) + B)) == \text{numberof}(E, B1) \text{ then } \text{isequal}(\text{remove}(E, B), \text{remove}(E, B1)) \text{ else false fi}.

eq \text{subset}(\text{empty}, B) = \text{true}.
eq \text{subset}(\text{singleton}(E), B) = \text{in}(E, B).
eq \text{subset}(\text{singleton}(E) + B, B1) = \text{if}
\quad \text{numberof}(E, (\text{singleton}(E) + B)) \leq \text{numberof}(E, B1) \text{ then } \text{subset}(B, B1) \text{ else false fi}.
eq \text{propersubset}(B, B1) = \text{subset}(B, B1) \text{ and not isequal}(B, B1).

eq \text{lengthof}(\text{empty}) = 0.
eq \text{lengthof}(\text{singleton}(E)) = 1.
eq \text{lengthof}(\text{singleton}(X) + B) = 1 + \text{lengthof}(B).

eq \text{remove}(E, \text{empty}) = \text{bagerror}.
eq \text{remove}(E, \text{singleton}(E1)) = \text{if } E == E1 \text{ then empty else singleton}(E1) \text{ fi}.
eq \text{remove}(X, \text{singleton}(E1) + B) = \text{if } X == E1 \text{ then } \text{remove}(X, B) \text{ else } (\text{singleton}(E1) + \text{remove}(X, B)) \text{ fi}.

eq \text{uniqueExtentOf}(\text{empty}) = 0.
eq \text{uniqueExtentOf}(\text{singleton}(E)) = 1.
eq \text{uniqueExtentOf}(\text{singleton}(X) + B) = \text{if } \text{in}(X, B) \text{ then } \text{uniqueExtentOf}(B) \text{ else } 1 + \text{uniqueExtentOf}(B) \text{ fi}.

eq \text{numberof}(E, \text{empty}) = 0.
eq numberof(E,singleton(E1)) = if E == E1 then 1 else 0 fi.
eq numberof(E,(singleton(E1) + B)) = if E == E1 then
    1 + numberof(E,B) else numberof(E,B) fi.
endo
************************************************************************
*** This is a Array obj3 (Array Data structure)
*** Keywords: Booch, Data-Structure, Array
*** Function: To perform Array data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototype.
************************************************************************
obj array[X :: TRIV] is sort Array .
    protecting NAT .
    protecting INT .
*** basic string operations
*** constructors
    op succ _ : Nat -> Nat .
    op pred _ : Nat -> Nat .
    op unitArray : Elt -> Array .
    op abuttedTo : Array Array -> Array .
    op emptyArray : -> Array .
    op copy : Array Array -> Array .
    op clear : Array -> Array .
    op delete : Nat Array -> Array .
*** assessors
    op isEqual : Array Array -> Bool .
    op isEmpty : Array -> Bool .
    op componentOf : Nat Array -> Elt .
    op sizeOf : Array -> Nat .
*** exceptions
    op eltUnderflow : -> Elt .
    op natUnderflow : -> Nat .
    op arrayError : -> Array .
*** Support Operations
    op headof : Nat Array -> Array .
    op tailof : Nat Array -> Array .
*** variable declaration
    vars A A1 A2 : Array .
    vars C1 C2 : Elt .
    vars I : Nat .
*** axioms
114
eq succ I = I + 1.
eq pred I = if (I > 1) then (I - 1) else natUnderflow fi.

eq headof(0,A) = A.
eq headof(I,abuttedTo(A1,A2)) =
    if sizeOf(A1) == I then A1 else arrayError fi.

eq tailof(0,abuttedTo(unitArray(C1),A)) = A.
eq tailof(I,abuttedTo(A1,A2)) =
    if sizeOf(A1) == I then A2 else arrayError fi.

eq delete(0,abuttedTo(unitArray(C1),A)) = A.
eq delete(I,A) = abuttedTo(headof(I,A),tailof(I,A)).

eq abuttedTo(abuttedTo(A,A1),A2) = abuttedTo(A,abuttedTo(A1,A2)).

eq componentOf(0,unitArray(C1)) = C1.
eq componentOf(0,abuttedTo(unitArray(C1),A)) = C1.
eq componentOf(I,abuttedTo(unitArray(C1),A)) =
    if I > 0 then componentOf(pred I,A) else eltUnderflow fi.

eq sizeOf(unitArray(C1)) = succ 0.
eq sizeOf(abuttedTo(unitArray(C1),A)) = succ (sizeOf(A)).

eq copy(emptyArray,A) = emptyArray.
eq copy(A1,A2) = if sizeOf(A1) == sizeOf(A2) then A1 else arrayError fi.

eq clear(A) = emptyArray.
eq isEqual(A1,A2) = if A1 == A2 then true else false fi.
eq isEmpty(clear(A)) = true.
eq isEmpty(emptyArray) = true.
endo

************************************************************************
*** This is a ring obj3 (ring Data structure)
*** Keywords: Booch, Data-Structure, Ring
*** Function: To perform Ring data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototpe.
************************************************************************

obj DIRECTION is
  sort Direction.
  ops Forward Backward : -> Direction.
endo

obj ring[X :: TRIV] is
  sorts Ring ERing.
  subsort Ring < ERing.

*** Import predefined type
protecting NAT.
protecting DIRECTION.

*** Generator
op empty : -> Ring.
op {_,_} : Nat Ring -> ERing.
op {_,_} : Nat Ring -> Ring.
***  °P {_,rotateerror} : Nat Ring -> Ring.

*** Other constructor/properties
op copy : Ring Ring -> Ring .
op copy : ERing ERing -> ERing .
op clear : ELt Ring -> ERing .
op insert : ELt Ring -> Ring .
op insert : ELt ERing -> ERing .
op pop : Ring -> Ring .
op pop : ERing -> ERing .
op rotate : Direction Ring -> Ring .
op rotate : Direction ERing -> ERing .
op mark : ERing -> ERing .
op rotatetomark : ERing -> ERing .
op isequal : ERing ERing -> Bool .
op extentof : Ring -> Nat .
op extentof : ERing -> Nat .
op isempty : ERing -> Bool .
op topof : Ring -> ELt .
op topof : ERing -> ELt .
op atmark : ERing -> Bool .

*** Exceptions
op underflow : -> Ring .
op underflow : -> ELt .
op rotateerror : -> Ring .

*** Variable declarations
var R R2 : Ring .
var A A2 : ERing .
var E E2 : ELt .
var M M2 : Nat .

*** Equations
eq {0,underflow} = underflow .
eq {M,rotateerror} = rotateerror .

eq copy(empty,R) = empty .
eq copy(insert(E,R),R2) = insert(E,copy(R,R2)) .
eq copy({M,R},{M2,R2}) = {M,copy(R,R2)} .
eq clear({M,R}) = {0,empty} .
eq insert(E, (M, R)) = if R == empty then \{0, insert(E, R)\} else \
\{M + 1, insert(E, R)\} fi.

eq pop(empty) = underflow.
eq pop(insert(E, R)) = R.
eq pop((M, R)) = if M == 0 then \{0, pop(R)\} else \{p(M), pop(R)\} fi.

eq rotate(Forward, empty) = rotateerror.
eq rotate(Backward, empty) = rotateerror.
eq rotate(Forward, insert(E, R)) = if R == empty then insert(E, empty) \
else insert(topof(R), rotate(Forward, insert(E, pop(R)))) fi.
eq rotate(Backward, insert(E, R)) = if R == empty then insert(E, empty) \
else insert(topof(rotate(Backward)), insert(E, pop(rotate(Backward, R)))) \
fi.
eq rotate(Forward, (M, R)) = if M = 0 \
then \{p(extentof(R)), rotate(Forward, R)\} else \n\{p(M), rotate(Forward, R)\} fi.
eq rotate(Backward, (M, R)) = if M == p(extentof(R)) then \
\{0, rotate(Backward, R)\} else \
\{M + 1, rotate(Backward, R)\} fi.

eq mark((M, R)) = \{0, R\}.
eq rotatetomark((M, R)) = if M == 0 then \{M, R\} else \
rotatetomark(rotate(Forward, (M, R))) fi.

eq isequal(A, A2) = if A == A2 then true else false fi.

eq extentof(empty) = 0.
eq extentof(insert(E, R)) = 1 + extentof(R).
eq extentof((M, R)) = extentof(R).

eq isempty((M, R)) = if R == empty then true else false fi.

eq topof(empty) = underflow.
eq topof(insert(E, R)) = E.
eq topof((M, R)) = topof(R).

eq atmark((M, R)) = if M == 0 then true else false fi.

endo

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

*** This is a Stack obj3 (Stack Data structure, version 1)  
*** Keywords: Booch, Data-Structure, Stack 
*** Function: To perform Stack data structure operations. 
*** Component History: Components from CS4520 Project. 
*** Modify syntax to work with SCS prototype. 
************************************************************************

obj stack l[X :: TRIV] is sort Stack.

117
protecting NAT.

*** CONSTRUCTORS:

op create : -> Stack .
op copy : Stack Stack -> Stack .
op clear : Stack -> Stack .
op push : Elt Stack -> Stack .
op pop : Stack -> Stack .
op empty : -> Stack .

*** SELECTORS:

op top : Stack -> Elt .
op depthof : Stack -> Nat .
op isempty : Stack -> Bool .
op isequal : Stack Stack -> Bool .

*** EXCEPTIONS:

op StackError : -> Stack .
op StackError : -> Elt .

*** VARIABLES:

var S S1 S2 : Stack .
var X : Elt .

*** AXIOMS:

eq clear(S) = empty .
eq copy(empty,S) = clear(S) .
eq copy(push(X,S1),S2) = push(X,copy(S1,S2)) .
eq top(empty) = StackError .
eq top(clear(S)) = StackError .
eq top(push(X,S)) = X .
eq pop(empty) = StackError .
eq pop(clear(S)) = StackError .
eq pop(push(X,S)) = S .
eq pop(create) = StackError .
eq depthof(empty) = 0 .
eq depthof(create) = 0 .
eq $\text{depthof}(\text{push}(X, S)) = 1 + \text{depthof}(S)$.
eq \text{isempty}(S) = \text{if } (S == \text{create}) \text{ or } (S == \text{empty}) \text{ then true else false fi}.
eq \text{isequal}(S1, S2) = \text{if } S1 == S2 \text{ then true else false fi}.
end

************************************************************************
*** This is a Stack obj3 (Stack Data structure, version 2)
*** Keywords: Booch, Data-Structure, Stack
*** Function: To perform Stack data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototpe.
************************************************************************

obj stack2[X :: TRIV] is sort Stack.
   protecting NAT.

*** basic stack operations
*** constructors
   op create : -> Stack.
   op clear : Stack -> Stack.
   op push : Elt Stack -> Stack.
   op pop : Stack -> Stack.

*** assessors
   op top : Stack -> Elt.
   op isempty : Stack -> Bool.
   op stacksize : Stack -> Nat.

*** exception
   op emptyerror : -> Stack.

*** variables declaration
   var S : Stack.
   var X : Elt.

*** axioms
   eq clear(S) = create.
   eq top(push(X, S)) = X.
   eq isempty(S) = \text{if } S == \text{create} \text{ then true else false fi}.
   eq stacksize(S) = \text{if } S == \text{create} \text{ then 0 else } 1 + \text{stacksize(pop(S)) fi}.
   eq pop(create) = emptyerror.
   eq pop(push(X, S)) = S.
endo

************************************************************************
*** This is a Queue obj3 (Queue Data structure)
*** Keywords: Booch, Data-Structure, Queue
*** Function: To perform Queue data structure operations.
*** Component History: Components from CS4520 Project.
*** Modify syntax to work with SCS prototpe.
************************************************************************

obj queue[X :: TRIV] is sort Queue.
   protecting NAT.

   op empty : -> Queue.
op copy.Q2 : Queue Queue -> Queue.
op clear.Q : Queue -> Queue.
op add.Q : Elt Queue -> Queue.
op pop.Q : Queue -> Queue.
op isequal : Queue Queue -> Bool.
op lengthof : Queue -> Nat.
op isempty : Queue -> Bool.
op frontof : Queue -> Elt.
op underflow : -> Queue.
op underflow : -> Elt.

var Q Ql Q2 : Queue.
var E : Elt.

eq copy.Q2(Ql,Q2) = Ql.
eq clear.Q(Q) = empty.
eq pop.Q(empty) = underflow.
eq pop.Q(add.Q(E,Q)) = if Q == empty then Q else add.Q(E,pop.Q(Q)) fi.
eq isequal(Ql,Q2) = if Ql == Q2 then true else false fi.
eq lengthof(empty) = 0.
eq lengthof(add.Q(E,Q)) = 1 + lengthof(Q).
eq isempty(Q) = if Q == empty then true else false fi.
eq frontof(empty) = underflow.
eq frontof(add.Q(E,Q)) = if Q == empty then E else frontof(Q) fi.
APPENDIX C - ADA SOURCE CODE

-- Module Name: gldef.a
-- Description: This is the global definition for variables of the
-- software component search prototype
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

Package Global_def is

Nill : constant := 999;
Max_maps: constant := 85;
Sort_id_range: constant := 30;
Void : Constant := 1;
Unvoid : Constant := 2;
F : constant := 0;
T : constant := 1;
Basic: constant := 1;
Confined: constant := 2;
Unconfined: constant := 3;
Rat_type : Constant := 1;
Int_type : Constant := 2;
Nat_type : Constant := 3;
Bool_type : Constant := 4;
Char_type : Constant := 5;
Elt_type : Constant := 12;
Unrelated: Constant := 1;
Related: Constant := 2;
Bquery_sort : Constant := 10;
Equery_sort : Constant := 20;

Type Test_case_str is new string(1..200);
Type Children_ids_type is array (1..3) of Natural;
Type Parent_ids_type is array (1..3) of Natural;
Type Ovl_type is array (1..25) of Natural;
Type Svl_type is array (1 ..25) of Natural;
Type Lopctype is array (1..25) of Boolean;
Type Ldsjype is array (1..25) of Boolean;
Type Symbol_name is new string(l ..20);
Type Sortjype is array (1..7) of Natural;
Type Matrix is array(Positive range <> ,Positive range <> ) of Natural;

Type Sort_Rank is
Record
    MaxoverallRank : Float;
    ModuleName : Symbol_name;
    SelSemanticRank : Float;
    SelSignatureRank : Float;
    SelKeywordRank : Float;
    SelProfileRank : Float;
    Mapnum: Natural;
End Record;

Type Asort_Array is array(1..12) of Sort_rank;

Type Testcase_Status_type is
    Record
        Complete : Boolean;
        Mul_value : Float;
        Top_function: Natural;
        Translate : Boolean;
    End Record;

Type Testcases_status_types is array(1..5) of Testcase_status_type;

Type Testequation_type is
    Record
        evalue: Testcase_str;
    End Record;

Type Testequations_types is array(1..5) of Testequation_type;

Type A_sort_table_type is
    Record
        Sort_id: Natural;
        Sort_symbol: Symbol_name;
        Main_sort: Natural;
        Sort_type: Natural;
        Sort_rank: Natural;
        Csortid: Natural;
        Children_ids: Children_ids_type;
        Parent_ids: Parent_ids_type;
        Ground_term: Symbol_name;
    End Record;
Type Sort_table_types is array(1..Sort_Id_Range) of A_sort_table_type;

Type Sort_table_def is
    Record
        Num_of_Sort: Natural;
        Sort_table: Sort_table_types;
    End Record;
Type Tstables is array(1..20) of Sort_table_def;

Type Q2csort is
Record
  Qsort : Natural;
  Csort : Natural;
End Record;
Type Q2csorts is array (1..10) of Q2csort;

Type Sal_type is
  Record
    Sort_account : Natural;
    Sort_asgn : Q2csorts;
  End Record;
Type Sal_types is array (1..10) of Sal_type;

Type Q2cop is
  Record
    Qop : Symbol_name;
    Cop : Symbol_name;
  End Record;
Type Q2cops is array (1..10) of Q2cop;

Type Oal_type is
  Record
    Op_account : Natural;
    Op_asgn : Q2cops;
  End Record;

Type Stack1 is
  Record
    Sal : Sal_type;
    Ovl : Ovl_type;
    Oal : Oal_type;
    Lope : Lopc_type;
    fi : Natural;
    Stable : Sort_table_def;
  End Record;
Type Stack1_type is array (1..10) of Stack1;

Type Stack2 is
  Record
    Sal : Sal_type;
    Stable : Sort_table_def;
    Svl : Svl_type;
    Lds : Lds_type;
    di : Natural;
  End Record;
Type Stack2_type is array (1..5) of Stack2;

Type Asig is
  Record
    Sal : Sal_type;
    Oal : Oal_type;

SignatureRank : Float;
SemanticRank : Float;
Testcase: Testcases_status_types;
Testequations: Testequations_types;
End Record;

Type Signature_map is array(1..Max_maps) of Asig;

Type Op is
  Record
    Rs : Natural;
    Dms : Sort_type;
    Symbol : Symbol_name;
    Profile : Natural;
  End Record;
Type COp_type is array(1..25) of Op;
Type QOp_type is array(1..10) of Op;

Type Symbol_type is
  Record
    Name : Symbol_name;
    Symboltype : Natural;
    Length : Natural;
  End Record;

Type KeywordList_def is array(1..5) of Natural;
Type ProfileList_def is array(1..11) of Natural;

Type SWC is
  Record
    Ops : COp_type;
    Num_ops : Natural;
    Obj_filename : Symbol_name;
    KeywordList : KeywordList_def;
    ProfileList : ProfileList_def;
  End Record;

Type Ground_equation is
  Record
    Eqtext : Test_case_str;
    ProfileList : ProfileList_def;
    Status : Natural;
  End Record;

Type QGround_equations is array(1..5) of Ground_equation;

Type QC is
  Record
    Ops : QOp_type;
    Num_ops : Natural;
    Num_tcases: Natural;
    Geq: QGround_equations;
    ProfileList : ProfileList_def;
    KeywordList : KeywordList_def;
  End Record;
End Record;

-- For Lookup table
Sort_assignment_table: Constant Matrix:=

Type CandidateTable_Def is
  Record
    ComponentId : Natural;
    ProfileKeywordRank : Float;
  End Record;
Type CandidatesTable is Array (1..14) of CandidateTable_Def;
Type Profile_err_list is Array (1..5) of Natural;

End Global_def;

-----------------------------------------------------------------
-- Module Name: scs.a
-- Description: This is the main driver for the software
-- component search prototype
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped
-----------------------------------------------------------------

With Op_Util_Mods; Use Op_Util_Mods;
With Global_def; Use Global_def;
With Query_processing_pkg; Use Query_processing_pkg;
With Swb_pkg; Use Swb_pkg;
With Swb_def_pkg; Use Swb_def_pkg;
With Signature_match_pkg; Use Signature_match_pkg;
With Semantic_match_pkg; Use Semantic_match_pkg;
With Init_pkg; Use Init_pkg;
With Formulate_Result_Output_pkg; Use Formulate_Result_Output_pkg;
With Float_io; Use Float_io;
With Integer_io; Use Integer_io;
With Text_io; Use Text_io;
With Integer_io; Use Integer_io;
Procedure Main is
  Candidates : CandidatesTable;
  Profile_err : Profile_err_list := (Others => Nill);
  Q: Qc; C: SWC;
  V: Signature_map;
  i : Natural := 1;
  c1 : natural := 0;
  sn : natural;
  Num_vmaps : Natural;
  Stable : Sort_table_def;
  UI,LI : Float;
  KeywordRank, ProfileRank : Float;
  SortArray : Asort_Array;
Begin

-- Get Query input from user
put("Enter Ll, Ul: ");
get(Ll); get(Ul);

Put("Enter Sn: ");
get(sn);

Get_Query(Q, Stable);
-- Retrieve components from Library using profile from test cases
FindCandidateComponents(Q, Candidates, Profile_err);
While True
Loop
If Candidates(i).ComponentId /= Nill then
    Get_Query(Q, Stable);
    Initialize_vmaps(V, Num_vmaps, Stable);
    InitComponentData(Candidates(i).ComponentId, C, Stable, Q, KeywordRank, ProfileRank);
    -- Signature Match
    put("Working on : "); put(String(C.Obj_filename)); New_line;
    c1 := 0;
    Signature_Match(Q, C, V, Stable, Num_vmaps, c1, sn, Ll, Ul, KeywordRank, ProfileRank);
    put("Count number of unfilter map"); put(c1); new_line;
    -- Semantic Match
    Semantic_Match(Q, C, V, Stable, Num_vmaps);
    -- Formulate Result and output to user
    Calculate_total_rank(V, Q, C, i, Num_vmaps, SortArray, KeywordRank, ProfileRank);
    i := i + 1;
Else
    exit;
End If;
End Loop;
Sort_Display_Result(SortArray, i-1);
Display_invalid_operations(Q, Profile_err);

End Main; -- End of main

-- Module Name: sigmat.a
-- Description: This is the signature matching procedure for the
-- software component search prototype (Algorithms 10)
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Op_Util_Mods; Use Op_Util_Mods;
With Global_def; Use Global_def;
With Sort_assignment_pkg; Use Sort_assignment_pkg;
With textio; Use textio;
With integer_io; Use integer_io;
Package Signature_match_pkg is
Procedure Signature_match (Q: in QC; C: In SWC; V: In Out Signature_Map;
    Stable: In Out Sort_table_def; Num_vmaps: In Out Natural;
    C1,Sn: In Out Natural; L1,U1,KwRank,PrfRank: Float);
End Signature_match_pkg;

Package body Signature_match_pkg is

Procedure Signature_match (Q: in QC; C: In SWC; V: In Out Signature_Map;
    Stable: In Out Sort_table_def; Num_vmaps: In Out Natural;
    C1,Sn: In Out Natural; L1,U1,KwRank,PrfRank: Float) is
    Tsal: Sal_types;
    Tstable: Tstables;
    Sal: Sal_type;
    Oal: Oal_type;
    Ovl: Ovl_type;
    Lopc: Lopc_type;
    Stack: Stack 1 type;
    stk_idx : Natural := 0;
    fi, pfj, fj, fip : Natural;
    Num_maps: Natural;
    Begin
        Initialize_state_variables(Sal,Oal,Ovl,Lopc,fi);
        While True -- Loop until all maps are covered
            Loop
                fj := Find_comp_op_idx(C.Num_ops,fi,Lopc,Ovl);
                If fj = Nil and fi /= Nil then
                    If fi = Q.Num_ops then -- Both fj and fi finished
                        If stk_idx = 0 then -- If stack is empty, it's done
                            -- Show_V(V,Num_vmaps,Stable); -- Display result;
                            Exit;
                        Else
                            Verify_Update_Rank(Sal,Oal,Stable,V,Num_vmaps,Q,C1,Sn,L1,U1,
                                KwRank,PrfRank);
                            Pop(Stack,Stk_idx,Sal,Stable,Oal,Ovl,Lopc,fi);
                            If Num_vmaps = Max_maps then
                                Return;
                            End If;
                        End If;
                    End if;
                    Else -- fi is not finish yet
                        Reset_all_previous_visit(C.Num_ops,fi,Lopc,Ovl);
                        fi := fi + 1; -- Get next fi
                    End If;
                Else -- fj not finish
                    Lopc(fj) := True; Ovl(fj) := fi;
                    Num_maps := 0;
                    If C.Ops(fj).Profile = Q.Ops(fi).profile then
                        Reset_previous_visit(C.Num_ops,fi,fj,Lopc,Ovl);
                        Sort_assignment(Q.Ops(fi).Dms,C.Ops(fj).Dms,Q.Ops(fj).Rs,
                            C.Ops(fj).Rs,Sal,Stable,Tsal,Tstable,Num_maps);
                    End If;
            End If;
        End While;

        Initialize_state_variables(Sal,Oal,Ovl,Lopc,fi);
        While True -- Loop until all maps are covered
            Loop
                fj := Find_comp_op_idx(C.Num_ops,fi,Lopc,Ovl);
                If fj = Nil and fi /= Nil then
                    If fi = Q.Num_ops then -- Both fj and fi finished
                        If stk_idx = 0 then -- If stack is empty, it's done
                            -- Show_V(V,Num_vmaps,Stable); -- Display result;
                            Exit;
                        Else
                            Verify_Update_Rank(Sal,Oal,Stable,V,Num_vmaps,Q,C1,Sn,L1,U1,
                                KwRank,PrfRank);
                            Pop(Stack,Stk_idx,Sal,Stable,Oal,Ovl,Lopc,fi);
                            If Num_vmaps = Max_maps then
                                Return;
                            End If;
                        End If;
                    End if;
                    Else -- fi is not finish yet
                        Reset_all_previous_visit(C.Num_ops,fi,Lopc,Ovl);
                        fi := fi + 1; -- Get next fi
                    End If;
                Else -- fj not finish
                    Lopc(fj) := True; Ovl(fj) := fi;
                    Num_maps := 0;
                    If C.Ops(fj).Profile = Q.Ops(fi).profile then
                        Reset_previous_visit(C.Num_ops,fi,fj,Lopc,Ovl);
                        Sort_assignment(Q.Ops(fi).Dms,C.Ops(fj).Dms,Q.Ops(fj).Rs,
                            C.Ops(fj).Rs,Sal,Stable,Tsal,Tstable,Num_maps);
                    End If;
            End If;
        End While;
    End Procedure;
End Package body Signature_match_pkg;
If Num_maps > 0 then -- There's possible sort maps
  Push(Sal, Stable, Oal, Ovl, Lopc, fi, Stk_idx, Stack);
  Update_Oal(Q.Ops(f, Symbol, C.Ops(fj).Symbol, Oal);
  Push_Individual_Sals(Num_maps, Tsal, Tstable, Sal, Stable, Oal, Ovl, Lopc, fi, Stkidx, Stack);
  Pop(Stack, Stk_idx, Sal, Stable, Oal, Ovl, Lopc, fi);
  If fi < Q.Num.ops then
    fi := fi + 1;
  Else -- Keep fi constant
    Verify_Update_Rank(Sal, Oal, Stable, V, Num_vmaps, Q.C1, Sn, L1, Ul, KwRank, PrfRank);
    Pop(Stack, Stk_idx, Sal, Stable, Oal, Ovl, Lopc, fi);
    If Num_vmaps = Max_maps then
      Return;
    End If;
  End If;
End If;
End Loop;
End Signature_Match;
End Signature_Match_pkg; -- End of signature match

-- Module Name: nsa.a
-- Description: This module performs the sort assignment for the
Signature Matching Algorithm.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With So_Util_Mods; Use So_Util_Mods;
With Sort_assignable_pkg; Use Sort_assignable_pkg;
With Text_io; Use Text_io;
With Integer_io; Use Integer_io;

Package Sort_assignment_pkg is

  Procedure Sort_assignment(Qsd, Csd: Sort_type; Qsr, Csr: Natural;
    Sal: Sal_type; Sort_table: Sort_table_def; Tsal: out Sal_types; Tstable: out Tstables; Num_maps: in out Integer);

End Sort_assignment_pkg;

Package body Sort_assignment_pkg is

  Procedure Sort_assignment(Qsd, Csd: Sort_type; Qsr, Csr: Natural;
    Sal: Sal_type; Sort_table: Sort_table_def; Tsal: out Sal_types; Tstable: out Tstables; Num_maps: in out Integer) is
    Svl: Svl_type;

128
Lds : Lds_type;
di : Natural;
dj,pdj : Natural;
Salp : Sal_type := Sal;
Stable : Sort_table_def := Sort_table;
Stk_idx : Natural := 0;
Stack : Stack2_type;
Sort_exist : Boolean;

Begin
Initialize_state_variables(Svl,Lds,di);
-- First, check range sort
If Assignable(Qsr,Csr,Stable) = T then
  Sort_exist := False;
For i in 1..Sal.Sort_account
  Loop
    If Qsr = Sal.Sort_asgn(i).Qsort then
      Sort_exist := True; -- Sort assignment existed
      Exit;
    End If;
  End Loop;
If Sort_exist = False then
  Update_Sort(Qsr,Csr,Salp,Stable);
End If;
If Num_sort(Csd) = 0 then -- operator is constant
  If Sort_exist = False then
    Num_maps := Num_maps + 1;
    Tstable(Num_maps) := Stable;
    Tsal(Num_maps) := Salp;
    Return;
  End If;
End If;
Else
  Return;
End If;
-- Second, check domain sorts
While True
  Loop
    dj := Find_comp_so_idx(Num_sort(Csd),di,Svl,Lds);
    pdj := Find_previous_comp_so_idx(Num_sort(Csd),di,Svl);
    If dj = Nil or di = Nil then
      If stk_idx = 0 then -- If stack is empty
        Exit; -- Done
      Else
        Pop(Stack,Stk_idx,Salp,Svl,Lds,Stable,di);
      End if;
    Else
      If pdj /= Nil then
        Lds(pdj) := False;
      End if;
      Lds(dj) := True; Svl(dj) := di;
      If Assignable(Qsd(di),Csd(dj),Stable) = T then
Sort_exist := False;
For i in 1..Sal.Sort_count
Loop
If Qsd(di) = Sal.Sort_asgn(i).Qsort then
  Sort_exist := True; -- Sort assignment existed
  Exit;
End If;
End Loop;
If Sort_exist = False then
  Push(Salp,Svl,Lds,di,Stable,Stk_idx,Stack);
  Update_Sort(Qsd(di),Csd(dj),Salp,Stable);
End If;
di := di + l;
If di > Num_sort(Qsd) then
  Num_maps := Num_maps + 1;
  Tstable(NumMaps) := Stable;
  Tsal(NumMaps) := Salp;
End if;
End if;
End Loop;
End Sort_assignment;
End Sort_assignment_pkg;

-- Module Name: asable.a
-- Description: This module determines whether two given sorts can be assigned or not.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With Text_io; Use Text_io;
With Integer_io; Use Integer_io;

Package Sort_Assignable_pkg is
  Function Determine_sort_types(Sorti,Sortj: Natural; Sortable: Sort_table_def) Return Natural;
  Function Determine_sort_relation(Sorti,Sortj: Natural; Sortable: Sort_table_def) Return Natural;
  Function Assignable(Sorti,Sortj: Natural; Stable: Sort_table_def) Return Natural;

End Sort_assignable_pkg;

Package body Sort_assignable_pkg is
  Function Determine_sort_relation(Sorti,Sortj: Natural; Sortable: Sort_table_def) Return Natural is
Begin

-- Column 1, rows 7,8,9 or Column 1, rows 2,5
If Sorttable.Sort_table(Sorti).Sort_type = Unconfined or
Sorttable.Sort_table(Sortj).Sort_type = Unconfined then
  Return Unrelated;
-- Columns 1,2 rows 4,6
Elsif Sorttable.Sort_table(Sorti).Sort_type = Confined then
  If Sorttable.Sort_table(Sorttable.Sort_table(Sorti).Csortid).Main_sort =
    Sorttable.Sort_table(Sortj).Main_sort then
    Return Related;
  Else
    Return Unrelated;
  End If;
-- Column 1,2 row 3
Elsif Sorttable.Sort_table(Sorti).Sort_type = Basic and
Sorttable.Sort_table(Sortj).Sort_type = Basic then
  If Sorttable.Sort_table(Sorti).Main_sort =
    Sorttable.Sort_table(Sortj).Main_sort then
    Return Related;
  Else
    Return Unrelated;
  End If;
-- Column 1,2 row 1
Else
  If Sorttable.Sort_table(Sorttable.Sort_table(Sortj).Csortid).Main_sort =
    Sorttable.Sort_table(Sorti).Main_sort then
    Return Related;
  Else
    Return Unrelated;
  End If;
End If;
End Determine_sort_relation;

Function Determine_Sort_types(Sorti,Sortj: Natural;Sorttable:
  Sort_table_def) Return Natural is
Sort_type_i, Sort_type_j : Natural;
Begin
  Sort_type_i := Sorttable.Sort_table(Sorti).Sort_type;
  Sort_type_j := Sorttable.Sort_table(Sortj).Sort_type;
  -- Look for row number
  If Sort_type_i = Basic and Sort_type_j = Confined then
    Return 1;
  Elsif Sort_type_i = Basic and Sort_type_j = Unconfined then
    Return 2;
  Elsif Sort_type_i = Basic and Sort_type_j = Basic then
    Return 3;
  Elsif Sort_type_i = Confined and Sort_type_j = Basic then
    Return 4;
  Elsif Sort_type_i = Confined and Sort_type_j = Unconfined then
    Return 5;
Elsif Sort_type_i = Confined and Sort_type_j = Confined then
    Return 6;
Elsif Sort_type_i = Unconfined and Sort_type_j = Basic then
    Return 7;
Elsif Sort_type_i = Unconfined and Sort_type_j = Confined then
    Return 8;
Else
    Return 9;
End If;
End Determine_sort_types;

Function Assignable(Sorti, Sortj: Natural; Stable: Sort_table_def) Return Natural is
    Row, Column: Natural;
    Begin
    If Sorti = Nill or Sortj = Nill then
        Return F;
    Else
        Row := Determine_sort_types(Sorti, Sortj, Stable);
        Column := Determine_sort_relation(Sorti, Sortj, Stable);
        Return(Sort_assignment_table(Row, Column));
    End If;
    End Assignable;
End Sort_assignable_pkg;

-- Module Name: Utilop.a
-- Description: This module does the utilities functions for the main
-- Signature Matching algorithm which is Sigmat.a.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With Text_io; Use Text_io;
With Integer_io; Use Integer_io;
With Float_io; Use Float_io;
With Semops_match_pkg; Use Semops_match_pkg;
Package Op_UtilMods is
    Procedure Push(Salp: In Sal_type; Stable: Sort_table_def; Oalp: In Oal_type;
        Ovlp: In Ovl_type; Lopc: Lopc_type; Stk_idx: In Out Natural; Stk_idx:
        In Out Natural; Stack: Out Stack1_type);
    Function Submapof(Oal : Oal_type; V: Signature_map; i: Natural) Return Boolean;
    Function Find_comp_op_idx(Num_ops, fi: Natural; Lopc: Lopc_type;
        Ovl: Ovl_type) return Natural;
    Procedure Reset_all_previous_visit(Num_ops, fi: Natural, Lopc: Lopc_type;
        Ovl: in Ovl_type) return Natural;
    Procedure Reset_previous_visit(Num_ops, fi, fj: Natural;
        Lopc: out Lopc_type; Ovl: in Ovl_type);
    Procedure Pop(Stack: In Stack1_type; Stk_idx: In Out Natural;
Salp: Out Sal_type; Stable: out Sort_table_def; Oalp:
Out Oal_type; Ølp: Out Ølv_type; Lopc: Out Lopc_type;
fi: Out Natural;
Procedure Update_Oal(Qsymbol,Csymbol: In Symbol_name;
Oal: In Out Oal_type);
Procedure Verify_Update_Rank(Sal: Sal_type;Oal: Oal_type;Stable:
Sort_table_def; V: In Out Signature_map; Num_vmaps:
In Out Natural;Q : Qc;cl,sn: In Out Natural;L1,U1,KwRank,
PrfRank: Float);
Procedure Initialize_state_variables(Sal: Out Sal_type; Oal: Out Oal_type;
Ølv: out Ølv_type;
Lopc: Out Lopc_type; fi: Out Natural);
Function CheckTopValue(Oal: Oal_type; Q: Qc) Return Boolean;
Procedure Push_Individual_Sals(Num_maps: In Natural;Tsal: In Sal_types;
Tstable: in Tstables; Sal: In Out Sal_type; Stable: In out
Sort_table_def; Oal: In Oal_type; Ølv: In Ølv_type;
Lopc: Lopc_type; fi: In Natural; Stk_idx : In Out Natural;
Stack: Out Stack LJype);
Procedure Show_V(V: Signature_map; Num_vmaps: Natural;Stable:
Sort_table_def);

End Op_UtiI_Mods;

Package body Op_UtiI_Mods is

Procedure Show_V(V: Signature_map; Num_vmaps: Natural;Stable:
Sort_table_def) is
Begin
For i in 1..Num_vmaps
Loop
Put(""); New_line;
Put("Map "); Put(i); New_line;
Put("""); New_line;
Put("Sort Assignment."); New_line;
For j in 1..V(i).Sal.Sort_acount
Loop
Put(String(Stable.Sort_table(V(i).Sal.Sort_asgn(j).Qsort).Sort_symbol));
Put("->");
Put(String(Stable.Sort_table(V(i).Sal.Sort_asgn(j).Csort).Sort_symbol));
New_Liine;
End Loop;

Put("Operator Assignment."); New_line;
For j in 1..V(i).Oal.Op_acount
Loop
Put(String(V(i).Oal.Op_asgn(j).Qop));
Put("->");
Put(String(V(i).Oal.Op_asgn(j).Cop));
New_Liine;
End Loop;
Put("SignatureRank:");

133
Procedure Push_Individual_Sals(Num_maps: In Natural; Tsal: In Sal_types;
  Tstable: in Tstables; Sal: In Out Sal_type; Stable: In out
  Sort_table_def; Oal: In Oal_type; Ovl: In Ovl_type;
  Lope: Lope_type; fi: In Natural; Stk_idx : In Out Natural;
  Stack: Out Stack1_type) is

Begin
  For n in 1.. Num_maps
    Loop
      Stable := Tstable(n);
      Sal.Sort_asgn := Tsal(n).Sort_asgn,
      Sal.Sort_acount := Tsal(n).Sort_acount;
      Push(Sal,Stable,Oal,Ovl,Lope,fi,Stk_idx,Stack);
    End Loop;
  End Push_Individual_Sals;

Procedure Verify_Update_Rank(Sal: Sal_type; Oal: Oal_type; Stable:
  Sort_table_def; V: In Out Signature_map; Num_vmaps:
  In Out Natural; Q : Qc; c1,Sn: In Out Natural; L1,U1,KwRank,
  PrfRank: Float) is

CofSuccess: Float;
  j : Natural := 1;
  Flag : Boolean := True;
  Ref_sort,Child_ref_sort,Count : Natural;

Begin
  — First verify subsort relation;
  For i in Bquery_sort.. Equery_sort
    Loop
      If Stable.Sort_table(i).Sort_id /= Nill and
        Stable.Sort_table(i).Csortid /= Nill then
        Ref_sort := Stable.Sort_table(i).Csortid;
        While Stable.Sort_table(i).Children_ids(j) /= Nill
          Loop
            If Stable.Sort_table(Stable.Sort_table(i).Children_ids(j)).Csortid
              /= Nill then
              Child_ref_sort :=
                Stable.Sort_table(Stable.Sort_table(i).Children_ids(j)).Csortid;
                If Stable.Sort_table(Child_ref_sort).Sort_rank <
                  Stable.Sort_table(Child_ref_sort).Main_sort /=
                    Stable.Sort_table(Child_ref_sort).Main_sort then
                      Flag := False; -- Sorts are not preserved in ordered.
                      Exit;
                    End If;
            End If;
            j := j + 1;
          End Loop;
        End While;
      End If;
    End Loop;
End If;
End Loop;
If Flag = True then -- Sort are presevered in ordered.
-- Now check for uniqueness
Count := 0;
For i in 1..Num_vmaps
Loop
If not Submapof(Oal,V,i) then
  Count := Count + 1;
End If;
End Loop;
If Count = Num_vmaps then
  CofSuccess :=
  KwRank * PrfRank * Float(Oal.Op_acount) / Float(Q.Num_ops);
If CheckTopValue(Oal,Q)
and then CofSuccess > L1 and then
  CofSuccess <= Ul then
  cl := cl + 1;
If cl > sn * Max_maps then
  If cl mod Max_maps = 0 then
    put("Warning Max maps reach ");new_line;
    Num_vmaps := Max_maps;
  Else
    Num_vmaps := cl mod Max_maps;
  V(Num_vmaps).Sal.Sort_asgn := Sal.Sort_asgn;
  V(Num_vmaps).Sal.Sort_acount := Sal.Sort_acount;
  V(Num_vmaps).SignatureRank :=
  Float(Oal.Op_acount) / Float(Q.Num_ops);
End If;
End If;
End If;
End If;
End If;
End If;
End Verify_update_Rank;

Function CheckTopValue(Oal: Oal_type; Q: Qc) Return Boolean is
  Tempsymbol : Test_case_str;
Begin
For i in 1..Oal.Op_acount
Loop
  For j in 1..Q.Num_tcases
  Loop
    Tempsymbol := (others => '');
    For m in 1..Test_case_str'Length
    Loop
      If Oal.Op_Asgn(i).Qop(m) = '' then
        Tempsymbol(m) := '(';
        Exit;
      End If;
    End Loop;
    Tempsymbol(m) := Oal.Op_Asgn(i).Qop(m);
  End Loop;
End Loop;
Return Tempsymbol = Q.tcases;
End CheckTopValue;
End Loop;
If Top_value(Q.Geq(j).Eqtext, Tempsymbol) then
  Return True;
End If;
End Loop;
End Loop;
Return False;
End CheckTopValue;

Function Submapof(Oal : Oal_type; V: Signature_map; i: Natural)
  Return Boolean is
    Flag: Boolean;
    Count: Natural := 0;
    Begin
      For m in 1..Oal.Op_count
        Loop
          Flag := False;
          For k in 1..V(i).Oal.Op_count
            Loop
              If V(i).Oal.Op_asgn(k).Qop = Oal.Op_asgn(m).Qop and
                Flag := True;
                Count := Count + 1;
                Exit;
              End If;
            End Loop;
          End Loop;
        If Count = Oal.Op_count then
          Return True;
        Else
          Return False;
        End If;
      End Loop;
    End Submapof;

Procedure Initialize_state_variables(Sal: Out Sal_type; Oal: Out Oal_type;
                                      Ovl: out Ovl_type; Lope: Out Lopc_type; fi: Out Natural) is
    Begin
      Sal.Sort_Acount := 0; — Note: there's no need for initialize Qsort, Csort.
      Oal.OpAcount := 0; — Same here.
      For j in 1..Ovl'Length
        Loop
          Ovl(j) := Nill;
          Lopc(j) := False;
        End loop;
      fi := l;
    End Initialize_state_variables;

Function Find_comp_op_idx(Num_ops,fi: Natural; Lope: Lopc_type; Ovl: Ovl_type) return Natural is
    Begin

For i in 1..Num_ops
  Loop
    If Ovl(i) /= fi and Lopc(i) /= True then
      Return i;
    End If;
  End Loop;
End Find_comp_op_idx;

Procedure Reset_all_previous_visit(Num_ops, fi: Natural;
  Lopc: out Lopc_type; Ovl: in Ovl_type) is
  Begin
    For i in 1..Num_ops
      Loop
        If Ovl(i) = fi then
          Lopc(i) := False;
        End If;
      End Loop;
    End Reset_all_previous_visit;

Procedure Reset_previous_visit(Num_ops, fi, fj: Natural;
  Lopc: out Lopc_type; Ovl: in out Ovl_type) is
  Begin
    For i in 1..Num_ops
      Loop
        If Ovl(i) = fi and i /= fj then
          Lopc(i) := False;
        End If;
      End Loop;
    End Reset_previous_visit;

Procedure Push(Salp: In Sal_type; Stable: Sort_table_def; Oalp: In Oal_type;
  Ovlp: In Ovl_type; Lopc: Lopc_type; fip: In Natural; Stk_idx:
  In Out Natural; Stack: Out Stack1_type) is
  Begin
    Stk_idx := Stk_idx + 1;
    Stack(Stk_idx).Sal.Sort_asgn := Salp.Sort_asgn;
    Stack(Stk_idx).Sal.Sort_acount := Salp.Sort_acount;
    Stack(Stk_idx).Stable := Stable;
    Stack(Stk_idx).Lopc := Lopc;
    Stack(Stk_idx).Ovl := Ovlp;
    Stack(Stk_idx).fi := fip;
  End Push;

Procedure Pop(Stack: In Stack1_type; Stk_idx: In Out Natural;
  Salp: Out Sal_type; Stable: out Sort_table_def; Oalp:
  Out Oal_type; Ovlp: Out Ovl_type; Lopc: Out Lopc_type;
  fip: Out Natural) is
  Begin

137
Package body So_Util_Mods is

End So_Util_Mods;

138
Function Num_sort(Sort_arr: Sort_type) return Natural is
Begin
For i in 1..Sort_type'Length
Loop
If Sort_arr(i) = Nil then
Return (i-1);
End If;
End Loop;
Return Nil; -- Return Error if get here.
End Num_sort;

Procedure Initialize_state_variables(Svl: Out Svl_type;
Lds: Out Lds_type;di: Out Natural) is
Begin
For j in L.Lds'Length
Loop
Lds(j) := False;
Svl(j) := Nil;
End loop;
di := 1;
End Initialize_state_variables;

Function Find_comp_so_idx(Num_sos,di: Natural;Svl: Svl_type; Lds: Lds_type)
return Natural is
Begin
For i in 1..Num_sos
Loop
If Svl(i) /= di and Lds(i) /= True then
Return i;
End If;
End Loop;
return Nil;
End Find_comp_so_idx;

Function Find_previous_comp_so_idx(Num_sos,di: Natural; Svl: Svl_type)
return Natural is
Begin
For i in 1..Num_sos
Loop
If Svl(i) = di then
Return i;
End If;
End Loop;
return Nil;
End Find_previous_comp_so_idx;

Procedure Push(Salp: Salp_type; Svl: Svl_type; Lds: Lds_type; di: Natural;
Stable: Sort_table_def;Stk_idx: In Out Natural; Stack:
Out Stack2_type) is
Begin
Stk_idx := Stk_idx + 1;
Stack(Stk_idx).Sal.Sort_asgn := Salp.Sort_asgn;
Stack(Stk_idx).Sal.Sort_acount := Salp.Sort_acount;
Stack(Stk_idx).Lds := Lds;
Stack(Stk_idx).Svl := Svl;
Stack(Stk_idx).di := di;
Stack(Stk_idx).Stable := Stable;
End Push;

Procedure Pop(Stack: In Stack2_type; Stk_idx: In Out Natural;
                Salp: Out Sal_type; Svl: Out Svl_type; Lds: Out Lds_type;
                Stable: Out Sort_table_def; di: Out Natural) is
Begin
    Salp.Sort_asgn := Stack(Stk_idx).Sal.Sort_asgn;
    Salp.Sort_acount := Stack(Stk_idx).Sal.Sort_acount;
    Svl := Stack(Stk_idx).Svl;
    Lds := Stack(Stk_idx).Lds;
    di := Stack(Stk_idx).di;
    Stable := Stack(Stk_idx).Stable;
    Stk_idx := Stk_idx - 1;
End Pop;

Procedure Update_Sort(Sorti, Sortj: Natural; Sal: In Out Sal_type;
                      Sorttable: In Out Sort_table_def) is
Sort_exist: Boolean;
Begin
    Sort_exist := False;
    For i in 1..Sal.Sort_acount
    Loop
        If Sorti = Sal.Sort_asgn(i).Qsort then
            Sort_exist := True; -- Sort assignment existed
            Exit;
        End If;
    End Loop;
    If Sort_exist = False then -- New sort assignment
        Sal.Sort_acount := Sal.Sort_acount + 1;
        Sal.Sort_asgn(Sal.Sort_acount).Qsort := Sorti;
        Sal.Sort_asgn(Sal.Sort_acount).Csort := Sortj;
    End If;
    If Sorttable.Sort_table(Sorti).Csortid = Nill then
        If Sorttable.Sort_table(Sorti).Sort_type = Unconfined then
            Sorttable.Sort_table(Sorti).Sort_type := Confined,
        End If;
        Sorttable.Sort_table(Sorti).Csortid := Sortj;
    End If;
    If Sorttable.Sort_table(Sortj).Csortid = Nill then
        If Sorttable.Sort_table(Sortj).Sort_type = Unconfined then
            Sorttable.Sort_table(Sortj).Sort_type := Confined,
        End If;
        Sorttable.Sort_table(Sortj).Csortid := Sorti;
    End If;
End Update_sort;
-- Module Name: init.a
-- Description: This module initializes the Map table and component
-- variables for preparing for signature matching process.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With text_io; Use text_io;
With integer_io; Use integer_io;
Package Init_pkg is
  Procedure Initialize_variables(V: Out Signature_map;
      Num_vmaps : Out Natural; Stable: Out Sort_table_def);
End Init_pkg;

Package body Init_pkg is

Procedure Initialize_variables(V: Out Signature_map;
      Num_vmaps : Out Natural; Stable: Out Sort_table_def) is
  Begin

    -- Initialize Signature Map variables
    Num_vmaps := 0;
    For i in 1..Signature_map'Length
    Loop
      For j in 1..Testequations_types'Length
      Loop
        V(i).Testequations(j).evalue := (Others => '');
        End Loop;
      End Loop;

    -- Initialize Sort table for component
    For i in 10..20
    Loop
      Stable.Sort_table(i).Sort_id := Nill;
      Stable.Sort_table(i).Sort_Symbol := (Others => ' ');
      Stable.Sort_table(i).Main_sort := Nill;
      Stable.Sort_table(i).Sort_rank := Nill;
      Stable.Sort_table(i).Sort_type := Unconfined;
      Stable.Sort_table(i).Csortid := Nill;
      Stable.Sort_table(i).Children_ids := (Others => Nill);
      Stable.Sort_table(i).Parent_ids := (Others => Nill);
      End Loop;
    End Initialize_variables;
  End Init_pkg; -- End of signature match
-- Module Name: semat.a
-- Description: This module does the semantic matching operation.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

with Global_def; use Global_def;
with Semops_match_pkg; use Semops_match_pkg;
with text_io; use text_io;
with integer_io; use integer_io;

Package Semantic_match_pkg is
  Procedure Semantic_match (Q: in QC; C: In SWC; V: In Out Signature_Map;
                           Stable: Sort_table_def; Num_vmaps: In Natural);
  End Semantic_match_pkg;

Package body Semantic_match_pkg is

  Procedure Semantic_match (Q: in QC; C: In SWC; V: In Out Signature_Map;
                           Stable: Sort_table_def; Num_vmaps: In Natural) is

  Begin
    Translate_ground_equations(Q, C, V, Stable, Num_vmaps);
    Execute_test_cases(V, Num_vmaps, Q, C.Obj_filename, Stable);
    Semantic_rank(V, Num_vmaps, Q);
  End Semantic_Match;
  End Semantic_Match_pkg; -- End of Semantic match

-- Module Name: semat.a
-- Description: This module does the semantic matching operations for
---semat.a
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

with TEXT_IO, SYSTEM;
use TEXT_IO, SYSTEM;
with Global_def; use Global_def;
with integer_io; use integer_io;
with Float_io; use Float_io;

Package Semops_match_pkg is

  Dot: Constant := 1;
  Groundsort: Constant := 2;
  Op: Constant := 3;
  Undefined: Constant := 4;
  Defined: Constant := 5;

  Procedure System_call(command: String);
Procedure Convert_ground_sort(Tsymbol, Psymbol: Symbol_name; SymbolName: In out Symbol_name; V: Signature_map; Num_arg: Natural; Num_map: Natural; Q: QC; Stable: Sort_table_def);

Procedure Getasymbol(Q: QC; Stable: Sort_table_def; V: Signature_map;
Tcase_str: Test_case_str; Str_ptr: In Out Natural; Num_map: Natural;
Num_arg: In Out Natural; Psymbol: In Out Symbol_name;
Symbol: In Out Symbol_type; Ext_stuff: In Out Symbol_name;
Length_Ext: Out Natural);

Procedure Translate_ground_equations(Q: in QC; C: In SWC; V: In Out Signature_map;
Stable: Sort_table_def; Num_vmaps: In Natural);

Procedure Get_make_statement(V: Signature_map; Mapnum: Natural; Makestm: In Out Test_case_str; Filename: Symbol_name; Stable: Sort_table_def; Flag: Out Boolean);

Procedure Execute_test_cases(V: In Out Signature_map; Num_vmaps: Natural; Q: QC;
Filename: Symbol_name; Stable: Sort_table_def);

Procedure Semantic_rank(V: In Out Signature_map; Num_vmaps: Natural; Q: QC);

Function Top_value(Equ1, Equ2: Test_case_str) Return Boolean;

End Semops_match_pkg;

Package body Semops_match_pkg is

Function Top_value(Equ1, Equ2: Test_case_str) Return Boolean is
i: Natural := 1;
Begin
While True
Loop
If i = Test_case_str'Length then
Return True;
Elsif Equ1(i) = Equ2(i) and Equ1(i) /= '(' then
i := i + 1; -- Normal path
Elsif Equ1(i) = Equ2(i) and Equ1(i) = '(' then
Return True;
Else
Return False;
End If;
End Loop;
End Top_Value;

Procedure Semantic_rank(V: In Out Signature_map; Num_vmaps: Natural; Q: QC) is
Input_File : Text_IO.File_Type;
Input_Line : String(1..13);

143
Len : Integer;
equationf : Natural;
Success : Natural;
Begin

Open(Input_File,In_File,"testrun.dat"); -- Note, it can recreate using objfile

-- Initialize Complete flags to be false, success = 0
For i in 1..Num_vmaps Loop
  For j in 1..Q.Num_tcases Loop
    V(i).Testcase(j).Complete := False;
  End Loop;
End Loop;

-- Now, Calculate mu
Reset(Input_File);
For i in 1..Num_vmaps Loop
  For j in 1..Q.Num_tcases Loop
    If V(i).Testcase(j).Complete = False then
      If V(i).Testcase(j).Top_function = Undefined then
        V(i).Testcase(j).Complete := True;
        V(i).Testcase(j).Mul_value := 0.0;
      Elsif V(i).Testcase(j).Top_function = Defined and
        V(i).Testcase(j).Translate = False then
        V(i).Testcase(j).Complete := True;
        V(i).Testcase(j).Mul_value := 1.0;
      Elsif V(i).Testcase(j).Top_function = Defined and
        V(i).Testcase(j).Translate = True then
        -- Calculate equation(f)
        Equationf := 0;
        Form in 1..Q.Num_tcases Loop
          If Top_value(V(i).Testequations(m).evalue,
            V(i).Testequations(j).evalue) then
            equationf := equationf + 1;
          End if;
        End Loop;

        -- Count success(f)
        Success := 0;
        For m in 1..Q.Num_tcases Loop
          If Top_value(V(i).Testequations(m).evalue,
            V(i).Testequations(j).evalue) and
            V(i).Testcase(m).Translate = True then
            Input_line := (others => '');
            If not End_Of_file(Input_File) then
              Get_line(Input_File,Input_Line,len);
            End if;
          End if;
        End Loop;
    End if;
  End Loop;
End Loop;

End;
If Input_Line(7) = 't' OR Input_Line(7) = 'T' THEN
    Success := Success + 1;
End If;
End If;
End If;
End Loop;
V(i).Testcase(j).Mul_value :=
1.0 + Float(Success) / Float(equationf) -
(Float(equationf) - Float(success)) / Float(equationf);

For m in 1..Q.Num_testcases -- Null out the rest of testcases
Loop
    If Top_value(V(i).Testequations(m).evalue,
        V(i).Testequations(j).evalue) then
        V(i).Testcase(m).Complete := True;
    End If;
    End Loop;
Else Null;
    End If;
End Loop;
End Loop;
Close(Input_File); -- Done, close inputfile

-- Initialize Complete flags to be True
For i in 1..Num_vmaps
Loop
    For j in 1..Q.Num_testcases
        Loop
            V(i).Testcase(j).Complete := True;
        End Loop;
    End Loop;
End Loop;

-- Now, Calculate SemanticRank
For i in 1..Num_vmaps
Loop
    V(i).SemanticRank := 0.0;
    For j in 1..Q.Num_testcases
        Loop
            If V(i).Testcase(j).Complete = True then
            End if;
        End Loop;
    End Loop;
End Loop;
End Semantic_rank;

Procedure Get_make_statement(V: Signature_map; Mapnum: Natural; Makestm: In Out
                   Test_case_str; Filename: Symbol_name; Stable: Sort_table_def; Flag:
                   Out Boolean) is
  i, m, k : Natural;
  Begin
    Flag := True;
    For j in 1..V(Mapnum).Sal.Sort_acount
      Loop
        If Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Csort).Sort_symbol(1..3) = 
          "Elt" then
          If Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Qsort).Sort_symbol(1..3) 
            = "NAT" or 
            Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Qsort).Sort_symbol(1..3) 
            = "INT" or 
            Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Qsort).Sort_symbol(1..3) 
            = "RAT" then
            Flag := True;
            Makestm(1..16) := "make testobj is ";
            i := 1;
            While True
              Loop
                If Filename(i) = '.' then
                  exit;
                Else
                  Makestm(i+17) := Filename(i);
                  i := i + 1;
                  End If;
              End Loop;
            End If;
        End If;
      End Loop;
      i := i + 17;
      Makestm(i..i) := "[";
      m := 1;
      While True
        Loop
          If Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Qsort).Sort_symbol(m) 
            = '_' then
            Exit;
          Else
            Makestm(i+m) :=
            Stable.Sort_table(V(Mapnum).Sal.Sort_asgn(j).Qsort).Sort_symbol(m);
            m := m + 1;
            End If;
        End Loop;
      k := i + m;
      Makestm(k..k+5) := "; endm";
    exit; -- We assume there is only 1 sort assigned to Elt
    Else
      Flag := False;
    End If;
  End If;
End Loop;
End Get_make_statement;

Procedure System_call(Command : String) is

procedure system_c (command : ADDRESS);
pragma INTERFACE(C, SYSTEM_C);
pragma INTERFACE_NAME(SYSTEM_C, "_system");
TEMP : constant STRING := command&ASCII.NUL;
ERROR : INTEGER;

Begin
    SYSTEM_C(TEMP ADDRESS);
End System_call;

Procedure Execute_test_cases(V: In Out Signature_map;Num_vmaps: Natural;Q: QC;
    Filename : Symbol_name;Stable: Sorttabledef) is

the_file: Text_io.file_type;
str : String(1..70) := (Others => '');
m : Natural;
makeflag,Display_flag : Boolean;
Makestm : Test_case_str;
Flag : Boolean;

Begin

Create(the_file,out_file,"testcase.dat");

-- First, create testcase.dat.
For i in 1..Num_vmaps
Loop
    makeflag := False;
    Makestm := (others => ' ');

    Display_flag := True;
    For j in 1..Q.Num_cases
    Loop
        If V(i).Testcase(j).Translate = True then
            If makeflag = False then
                Get_make_statement(V,i,Makestm,Filename,Stable,Flag);
                If Flag = False then
                    For k in 1..Q.Num_cases
                    Loop
                        V(i).Testcase(k).Translate := False;
                    End Loop;
                    Exit;
                End If;
                new_line(the_file);
                Put(the_file,string(Makestm));
                makeflag := True;
            End If;
        End If;
    End Loop;
End If;
End If;
If Display_flag = True then
Display_flag := False;
new_line(the_file);
Put(the_file,"**** *");
Put(the_file,"++++++++++++++++++++++"); New_line(the_file);
Put(the_file,"**** *");
Put(the_file,"Map #"); Put(the_file,i); New_line(the_file);
Put(the_file,"**** *");
Put(the_file,"++++++++++++++++++++++"); New_line(the_file);
Put(the_file,"**** *");
Put(the_file,"SignatureRank");
For jl in 1..V(i).Sal.Sort_acount
Loop
  Put(the_file,"**** *");
  For k in 1..Symbol_name'Length
    Loop
      Put(the_file,
        Stable.Sort_table(V(i).Sal.Sort_asgn(jl).Qsort).Sort_symbol(k));
    End Loop;
    Put(the_file,"->");
  End Loop;
  New_line(the_file);
End Loop;
Put(the_file,"**** *");
Put(the_file,"Operator Assignment:");New_line(the_file);
For jl in 1..V(i).Oal.Op_acount
Loop
  Put(the_file,"**** *");
  For k in 1..Symbol_name'Length
    Loop
      Put(the_file,V(i).Oal.Op_asgn(jl).Qop(k));
    End Loop;
  End Loop;
  New_line(the_file);
End Loop;
Put(the_file,"**** *");
Put(the_file,"SignatureRank");
Put(the_file,Float(V(i).SignatureRank),4,1);New_line(the_file);
End If;
New_line(the_file);
Put(the_file, "*** Grd Eq.");
Put(the_file, String(Q,Geq(j).Eqtext));
New_line(the_file);
Put(the_file, "*** Testcase #"); Put(the_file, j);
New_line(the_file);
Put(the_file, "reduce ");
Put(the_file, String(V(i).Testequations(j).evalue));
New_line(the_file);
End If;
End Loop;
End Loop;
close(the_file);

-- second, cp filename testfile.obj
str(1..4) := "cp ",
m := 1;
While true
Loop
If Filename(m) = '' or m = Symbol_name'Length then
exit;
Else
str(m+4) := Filename(m);
m := m + 1;
End If;
End Loop;
str(16..27) := "tempfile.obj";
system_call(str);

-- Sed to remove control char
system_call("cat testcase.dat >> tempfile.obj");
system_call("cat tempfile.obj | sed -e '' > testfile.obj");

-- put("I run obj now"); new_line;
system_call("runobj");

-- Save a copy for the wrapper usage later
str := (Others => '');
str(1..15) := "mv testfile.obj";
str(16..17) := " ",
For i in 1..m
Loop
str(18+i) := Filename(i);
End Loop;
Str(18+m..20+m) := ".tc";
system_call(str);
End Execute_test_cases;

Procedure Convert_ground_sort(Tsymbol,Psymbol: Symbol_name;SymbolName: In out
Symbol_name;V : Signature_map;Num_arg: Natural;Num_map: Natural;
Q:QC;Stable: Sort_table_def) is
Csort_type,Qsort_type : Natural := Nil;
Sort_id,Op_index,index : Natural;
Begin
If Psymbol(l) /= '' then -- There exist previous symbol
   -- Get Op index
For i in 1..Q.Num_ops
   Loop
      If Q.Ops(i).Symbol = Psymbol then
         Op_index := i;
         Exit;
      End If;
   End Loop;
   -- Find sort type for query and component
   Sort_id := Q.Ops(Op_index).dms(Num_arg);
For j in 1..V(Num_map).Sal.Sort_acount
   Loop
      If V(Num_map).Sal.Sort_asgn(j).Qsort = Sortjd then
         QSort_type := Stable.Sort_table(Sort_id).Sort_id;
         Csort_type :=
         Exit;
      End If;
   End Loop;
Else -- Constant stand by itself
   If Tsymbol(l) = V or Tsymbol(l) = 'f or
      Tsymbol(l) = T or Tsymbol(l) = '*F' then
      QsortType := Booltype;
   Else
      Qsort_type:= Nat_type;
   End If;
   For i in 1..Symbol_name'Length
      Loop
         If Tsymbol(i) = '-' then
            Qsort_type := Int_type;
            exit;
         Elsif Tsymbol(i) = '.' then
            Qsort_type := Rat_type;
            exit;
         Else
            Null;
         End If;
      End Loop;
   End If;
   -- Compute correspondence mapped sort for component sort
   For j in 1..V(Num_map).Sal.Sort_acount
      Loop
         If Stable.Sort_table(V(Num_map).Sal.Sort_asgn(j).Qsort).Sort_Id =
            Qsort_type then
            Csort_type :=
            Exit;
         End If;
      End Loop;
   End If;
End if;
End If;
End loop;
End If;
-- Determine SymbolName
If Qsort_type = Bool_type then
If Csort_Type > 10 and Csort_Type /= Elt_type then
  SymbolName := Stable.Sort_table(Csort_type).Ground_term;
Else
  SymbolName := Tsymbol;
End If;
Elsif Qsort_type = Nat_type then
If Csort_Type > 10 and Csort_Type /= Elt_type then
  SymbolName := Stable.Sort_table(Csort_type).Ground_term;
Else -- Csort_type = Elt,Int,Float
  SymbolName := Tsymbol;
End If;
Elsif Qsort_type = Int_type then
If Csort_Type > 10 and Csort_Type /= Elt_type then
  SymbolName := Stable.Sort_table(Csort_type).Ground_term;
ElIsIf Csort_type = Nat_type and Tsymbol(1) = '-' then
  SymbolName(1..Symbol_name'Length) :=
  Tsymbol(2..Symbol_name'Length) & ' ';
else -- Csort_type = Elt,Int,Float
  SymbolName := Tsymbol;
End If;
Else
if Csort_Type > 10 and Csort_Type /= Elt_type then
  SymbolName := Stable.Sort_table(Csort_type).Ground_term;
ElseIf Csort_type = Int_type or Csort_type = Nat_type then -- Truncate it
  For i in 1..Symbol_name'Length
    Loop
      If Tsymbol(i) /= ' ' then
        SymbolName(i) := Tsymbol(i);
      Else
        index := i;
        For m in i+1..Symbol_name'Length -- Found ' ' here
          Loop
            If Tsymbol(m) not in '0'..'9' then
              SymbolName(index) := Tsymbol(m);
              Index := Index + 1;
            End If;
          End Loop;
        Exit;
      End If;
    End Loop;
  End If;
End Else -- Csort_type = Float,Elt
  SymbolName := Tsymbol;
End If;
End If;
End Convert_ground_sort;

Procedure Getasymbol(Qc:Qc; Stable: Sort_table_def; V: Signature_map;
  Tcase_str: Test_case_str; Str_ptr: In Out Natural; Num_map: Natural;
  Num_arg: In Out Natural; Psymbol: In Out Symbol_name;
  Symbol: In Out Symbol_type; Ext_stuff: In Out Symbol_name;
  Length_Ext: Out Natural) is
  Tsymbol: Symbol_name := (Others =>'');
  Temp_ptr: Natural := Str_ptr;
  Ex_ptr: Natural;
  Psymflag: Boolean;
Begin
  -- If first char is char
  If Tcase_str(Str_ptr) in 'A'..'Z' or Tcase_str(Str_ptr) in 'a'..'z' then
    While True
      Loop
        If Tcase_str(Temp_ptr) = ')' or Tcase_str(Temp_ptr) = '(' or
          Tcase_str(Temp_ptr) = ' ' or Tcase_str(Temp_ptr) = ',' or
          Tcase_str(Temp_ptr) = '=' then
          Ex_ptr := 0;
          Psymflag := False;
          While True -- Also get extra stuff & move ptr to
            Loop
              If Tcase_str(Temp_ptr+Ex_ptr) in 'A'..'Z'
              or Tcase_str(Temp_ptr+Ex_ptr) in 'a'..'z'
              or Tcase_str(Temp_ptr+Ex_ptr) in '0'..'9'
              or Tcase_str(Temp_ptr+Ex_ptr) = '.' then
                Exit;
              Else
                If Tcase_str(Temp_ptr+Ex_ptr) = '' then
                  Psymflag := True;
                  End If;
                Ex_ptr := Ex_ptr + 1;
                End If;
            End Loop;
          End While;
        For i in 1..Temp_ptr-Str_ptr
            Loop
              Tsymbol(i) := Tcase_str(Str_ptr+i-1);
            End Loop;
        End For;
        For i in 1..Ex_ptr
          Loop
            Ext_stuff(i) := Tcase_str(Temp_ptr+i-1);
          End Loop;
        Exit; -- One symbol found
      Else
        Temp_ptr := Temp_ptr + 1; -- Get pass 1 of char above to new char
End If;
End Loop;

-- If first char is numeric
Elsif Tcase_str(Str_ptr) in '0'..'9' then
While True
Loop
If Tcase_str(Temp_ptr) = ')' or Tcase_str(Temp_ptr) = '(' or
Tcase_str(Temp_ptr) = ' or Tcase_str(Temp_ptr) = ',' or
Tcase_str(Temp_ptr) = '=' then
Ex_ptr := 0;
Psymflag := False;
While True -- Also get extra stuff & move ptr to
Loop
If Tcase_str(Temp_ptr+Ex_ptr) in 'A'..'Z'
or Tcase_str(Temp_ptr+Ex_ptr) in 'a'..'z'
or Tcase_str(Temp_ptr+Ex_ptr) in '0'..'9'
or Tcase_str(Temp_ptr+Ex_ptr) = '.' then
Exit;
Else
If Tcase_str(Temp_ptr+Ex_ptr) = '' then
Psymflag := True;
End If;
Ex_ptr := Ex_ptr + 1;
End If;
End Loop;
End Loop;

For i in 1..Temp_ptr-Str_ptr
Loop
Tsymbol(i) := Tcase_str(Str_ptr+i-l);
End Loop;
For i in 1..Ex_ptr
Loop
Ext_stuff(i) := Tcase_str(Temp_ptr+i-l);
End Loop;
exit; -- One symbol found

Else
Temp_ptr := Temp_ptr+1; -- Get pass 1 of char above to new char
End If;
End Loop;
Else -- Must be a dot since space is skip from above
Symbol.Symboltype := Dot;
End If;

-- Check for valid symbol
If Symbol.Symboltype /= Dot then
If Tcase_str(Str_ptr) in '0'..'9' or -- ground term
Tcase_str(Str_ptr) = 't' or
Tcase_str(Str_ptr) = '.' or
Tcase_str(Str_ptr..Str_ptr+3) = "true" or
Tcase_str(Str_ptr..Str_ptr+3) = "True" or
Tcase_str(Str_ptr..Str_ptr+4) = "false" or
Tcase_str(Str_ptr..Str_ptr+4) = "false"
If Psymflag = True then
  Psymbol := (Others => ''); -- Set flag for constant evaluation
End if;
Symbol.Symboltype := Groundsort;
Convert_ground_sort(Tsymbol,Psymbol,Symbol.Name,V,Num_arg,Num_map,
Q,Stable);
If Psymbol(1) /= '' then
  Num_arg := Num_arg + 1;
End If;
Length_ext := Ex_ptr;
Str_ptr := Temp_ptr;
For i in 1..Symbol_name'Length
  Loop
    If Symbol.Name(i) = '' then
      Symbol.Length := i-1;
      Exit;
    End If;
  End Loop;
Else
  Symbol.Symboltype := Null;
  For j in 1..V(Num_map).Oal.Op_account
    Loop
      If V(Num_map).Oal.Op_asgn(j).Qop(1..(Temp_ptr-Str_ptr)) =
        Tsymbol(1..(Temp_ptr-Str_ptr)) then
        If Psymflag = False then
          Psymbol := V(Num_map).Oal.Op_asgn(j).Qop;
        Else
          Psymbol := (Others => ''); -- Set flag for constant evaluation
        End if;
        Num_arg := 1;
      Symbol.Symboltype := Op;
      For i in 1..Symbol_name'Length
        Loop
          If Symbol.Name(i) = '' then
            Symbol.Length := i-1;
            Exit;
          End If;
        End Loop;
        Length_ext := Ex_ptr;
        Str_ptr := Temp_ptr;
        Exit;
      End If;
    End Loop;
  End Loop;
  If Symbol.Symboltype /= Op then
    Symbol.Symboltype := Undefined;
  End If;
End If;
End If;
End Getasymbol;

Procedure Translate_ground_equations(Q: in QC; C: In SWC; V: In Out
Signature_Map; Stable: Sort_table_def; Num_vmaps: In Natural) is
Save_ptr, Str_ptr, Num_arg : Natural;
Symbol: Symbol_type;
Psymbol : Symbol_name := (Others => ' ');
Tflag : Boolean;
Ext_stuff: Symbol_name := (Others => ' ');
Length_Ext : Natural;
i : Natural;
Begin
For Num_map in 1..Num_vmaps
Loop
For tnum in 1..Q.Num_cases
Loop
  Num_arg := 1;
  Psymbol := (Others => ' ');
  Str_ptr := 1;
  Save_ptr := Str_ptr;
  Symbol.Symboltype := Nil;
  Tflag := False;
i := 0;
While True
Loop
  Symbol.Name := (Others => ' ');
  Getasymbol(Q, Stable, V, Q.Geq(tnum).Eqtext, Str_ptr, Num_map, Num_arg,
Psymbol, Symbol, Ext_stuff, Length_Ext);
i := i + 1;
If Symbol.SymbolType = Dot then -- Done
  V(Num_map).testequations(tnum).evalue(Save_ptr + 2) := '.';
  Tflag := True;
  Exit;
Elsif Symbol.Symboltype = Undefined then
  If i = 1 then
    V(Num_map).Testcase(tnum).Top_function := Undefined;
    End If;
    Tflag := False;
    Exit;
  Else
    If i = 1 then
      V(Num_map).Testcase(tnum).Top_function := Defined;
      End If;
      Tflag := True;
      For j in 1..Symbol.Length
      Loop
        V(Num_map).testequations(tnum).evalue(Save_ptr + j) :=
        Symbol.Name(j);
      End Loop;
    Else
      V(Num_map).testequations(tnum).evalue(Save_ptr + j) :=
      Symbol.Name(j);
      End Loop;
  End If;
End Loop;

For j in 1..Length_Ext
Loop
  V(Num_map).testequations(tnum).evalue(Save_ptr+(Symbol.Length)+j) := Ext_stuff(j);
End Loop;
Str_ptr := Str_ptr + Length_ext;
save_ptr := Save_ptr+(Symbol.Length)+Length_ext;
End If;
End Loop;
If Tflag = True then
  V(Num_map).Testcase(tnum).Translate := True;
Else
  V(Num_map).Testcase(tnum).Translate := False;
End If;
End Loop;
End Translate_ground_equations;

End Semops_Match_pkg; -- End of Semops match

-- Module Name: swb_def.a
-- Description: This module defines definition for variables for
-- software library.
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped
Package Swb_def_pkg is
Include : Constant := 1;
Max_node_range : Constant := 30;
RootNode : Constant := 0;

-- Keyword id
Kw_Booch : Constant := 1;
Kw_DataStructure : Constant := 2;
Kw_Stack : Constant := 3;
Kw_Binary_tree : Constant := 4;
Kw_Queue : Constant := 5;
Kw_Set : Constant := 6;
Kw_Array : Constant := 7;
Kw_Bag : Constant := 8;
Kw_String : Constant := 9;
Kw_List : Constant := 10;
Kw_Ring : Constant := 11;
Kw_Deque : Constant := 12;

-- Obj module id
Stack1_obj : Constant := 1;
Stack2_obj : Constant := 2;
Bint1_obj : Constant := 3;
Bint2_obj : Constant := 4;
Queue_obj : Constant := 5;
Set_obj : Constant := 6;
Array_obj : Constant := 7;
Bag_obj : Constant := 8;
List1_obj : Constant := 9;
List2_obj : Constant := 10;
Ring_obj : Constant := 11;
Deque_obj : Constant := 12;

Type Children_id_list is array(1..11) of Natural;
Type Component_id_list is array(1..11) of Natural;
Type Profile_id_list is array(1..11) of Natural;

Type Hasse_node_type is
  Record
    Visit : Boolean;
    Pipro : Boolean;
    Childrenlist : Children_id_list;
    Componentlist : Component_id_list;
    ProfileList : Profile_id_list;
  End Record;

Type Hasse_diagram_type is array(0..Max_node_range) of Hasse_node_type;

End Swb_def_pkg;

-- Module Name: swb.a
-- Description: This module initialize components data for the
-- software library
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With Swb_def_pkg; Use Swb_def_pkg;
With text_io; Use text_io;
With integer_io; Use integer_io;

Package Swb_pkg is
  Function DetermineProfileIntersection(GeqProfile:Profilelist_def; Pprofile:
    Profile_id_list) Return Boolean;
  Procedure FindCandidateComponents(Q: In Out Qc; Candidates: In Out
    CandidatesTable; Profileerr: Out Profileerrlist);
  Procedure IntializeSwb(H : In Out Hasse_diagram_type);
  Procedure IntializeSwb(H : In Out Hasse_diagram_type);
  Function LookupBlockIndex(Pindex: Natural) return Integer;
Procedure PeformDfsfw(H: In Out Hasse_diagram_type;Q: Qc; node: Integer; 
Candidates: Out CandidatesTable; Eoa: In Out Natural);
Procedure InitComponentData(ComponentId: Natural;C: In Out SWC; Stable:
In Out Sort_table_def;Q : Qc; KwRank, PrfRank :
Out Float);
Function ProfileRank(Q: Qc; C : SWC) return Float;
Function KeywordRank(Q: Qc; C : SWC) return Float;
End Swb_pkg;

Package body Swb_pkg is

Function KeywordRank(Q: Qc; C :SWC) return Float is
Qkeywcard,Intersect : Natural := 0;
Begin
For i in l..KeywordList_def'Length
Loop
If Q.KeywordList(i) = Nill then
Exit;
Else
Qkeywcard := Qkeywcard + 1;
For j in 1..KeywordList_def'Length
Loop
If Q.KeywordList(i) = C.KeywordList(j) then
Intersect := Intersect + 1;
End If;
End Loop;
End if;
End Loop;
If Intersect = 0 then
Put("Warning: Incorrect Keyword ranking");New_Line;
Else
Return(Float(Intersect)/Float(Qkeywcard));
End If;
End KeywordRank;

Function ProfileRank(Q: Qc; C :SWC) return Float is
Qprofcard,Intersect: Natural := 0;
Begin
For i in 1..Profilelist_def'Length
Loop
If Q.ProfileList(i) = Nill then
Exit;
Else
Qprofcard := Qprofcard + 1;
For j in 1..Profilelist_def'Length
Loop
If Q.ProfileList(i) = C.ProfileList(j) then
Intersect := Intersect + 1;
End If;
End Loop;
End If;
End Loop;
End If;
End ProfileRank;

158
End Loop;
If Intersect = 0 then
  Put("Warning: Incorrect Profile ranking"); New_line;
Else
  Return(Float(Intersect)/Float(Qprofcard));
End If;
End ProfileRank;

Procedure InitComponentData(ComponentId: Natural; C: In Out SWC; Stable;
  In Out Sort_table_def; Q: Qc; KwRank, PrfRank: Out Float) is
  Begin

  Case ComponentId is
  "ring.obj"
  Begin

    When ring_obj =>
    Begin
      -- put("Working on ring_obj"); New_line;
      -- op empty: Ring obj
      C.Ops(1).Profile := 110;
      C.Ops(1).Symbol := "empty",
      C.Ops(1).rs := 11;
      C.Ops(1).dms(1) := Nill;

      -- op copy: Ring Ring -> Ring
      C.Ops(2).Profile := 3301;
      C.Ops(2).Symbol := "copy",
      C.Ops(2).rs := 11;
      C.Ops(2).dms(1) := 11;
      C.Ops(2).dms(2) := 11;
      C.Ops(2).dms(3) := Nill;

      -- op copy: ERing ERing -> ERing
      C.Ops(3).Profile := 3301;
      C.Ops(3).Symbol := "copy",
      C.Ops(3).rs := 16;
      C.Ops(3).dms(1) := 16;
      C.Ops(3).dms(2) := 16;
      C.Ops(3).dms(3) := Nill;

      -- op clear: ERing -> ERing
      C.Ops(4).Profile := 2201;
      C.Ops(4).Symbol := "clear",
      C.Ops(4).rs := 16;
      C.Ops(4).dms(1) := 16;
      C.Ops(4).dms(2) := Nill;

      -- op insert: Elt Ring -> Ring
      C.Ops(5).Profile := 3211;

  End Case;

End InitComponentData;
C.Ops(5).Symbol := "insert";
C.Ops(5).rs := 11;
C.Ops(5).dms(1) := 12;
C.Ops(5).dms(2) := 11;
C.Ops(5).dms(3) := Nill;

-- op insert: Elt ERing -> ERing
C.Ops(6).Profile := 3211;
C.Ops(6).Symbol := "insert";
C.Ops(6).rs := 16;
C.Ops(6).dms(1) := 12;
C.Ops(6).dms(2) := 16;
C.Ops(6).dms(3) := Nill;

-- op pop: Ring -> Ring
C.Ops(7).Profile := 2201;
C.Ops(7).Symbol := "pop";
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := Nill;

-- op pop: ERing -> ERing
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "pop";
C.Ops(8).rs := 16;
C.Ops(8).dms(1) := 16;
C.Ops(8).dms(2) := Nill;

-- op rotate: Direction Ring -> Ring
C.Ops(9).Profile := 3211;
C.Ops(9).Symbol := "rotate";
C.Ops(9).rs := 11;
C.Ops(9).dms(1) := 15;
C.Ops(9).dms(2) := 11;
C.Ops(9).dms(3) := Nill;

-- op rotate: Direction ERing -> ERing
C.Ops(10).Profile := 3211;
C.Ops(10).Symbol := "rotate";
C.Ops(10).rs := 16;
C.Ops(10).dms(1) := 15;
C.Ops(10).dms(2) := 16;
C.Ops(10).dms(3) := Nill;

-- op mark: ERing -> ERing
C.Ops(11).Profile := 2201;
C.Ops(11).Symbol := "mark";
C.Ops(11).rs := 16;
C.Ops(11).dms(1) := 16;
C.Ops(11).dms(2) := Nill;
-- op rotatetomark: ERing -> ERing
C.Ops(12).Profile := 2201;
C.Ops(12).Symbol := "rotatetomark ";
C.Ops(12).rs := 16;
C.Ops(12).dms(1) := 16;
C.Ops(12).dms(2) := Nil;

-- op isequal: ERing Ering -> Bool
C.Ops(13).Profile := 3210;
C.Ops(13).Symbol := "isequal ";
C.Ops(13).rs := 13;
C.Ops(13).dms(1) := 16;
C.Ops(13).dms(2) := 16;
C.Ops(13).dms(3) := Nil;

-- op extentof: Ring -> Nat
C.Ops(14).Profile := 220;
C.Ops(14).Symbol := "extentof ";
C.Ops(14).rs := 14;
C.Ops(14).dms(1) := 11;
C.Ops(14).dms(2) := Nil;

-- op extentof: ERing -> Nat
C.Ops(15).Profile := 220;
C.Ops(15).Symbol := "extentof ";
C.Ops(15).rs := 14;
C.Ops(15).dms(1) := 16;
C.Ops(15).dms(2) := Nil;

-- op isempty : ERing -> Bool
C.Ops(16).Profile := 220;
C.Ops(16).Symbol := "isempty ";
C.Ops(16).rs := 13;
C.Ops(16).dms(1) := 16;
C.Ops(16).dms(2) := Nil;

-- op atmark: ERing -> Bool
C.Ops(17).Profile := 220;
C.Ops(17).Symbol := "atmark ";
C.Ops(17).rs := 13;
C.Ops(17).dms(1) := 16;
C.Ops(17).dms(2) := Nil;

-- op topof: Ring -> Bool
C.Ops(18).Profile := 220;
C.Ops(18).Symbol := "topof ";
C.Ops(18).rs := 12;
C.Ops(18).dms(1) := 11;
C.Ops(18).dms(2) := Nil;

-- op topof: ERing -> Bool
C.Ops(19).Profile := 220;
C.Ops(19).Symbol := "topof ";
C.Ops(19).rs := 12;
C.Ops(19).dms(1) := 16;
C.Ops(19).dms(2) := Nill;

-- op underflow: -> ring
C.Ops(20).Profile := 110;
C.Ops(20).Symbol := "underflow ";
C.Ops(20).rs := 11;
C.Ops(20).dms(1) := Nill;

-- op rotaterror: -> ring
C.Ops(21).Profile := 110;
C.Ops(21).Symbol := "rotaterror ";
C.Ops(21).rs := 11;
C.Ops(21).dms(1) := Nill;

-- op underflow: -> elt
C.Ops(22).Profile := 110;
C.Ops(22).Symbol := "underflow ";
C.Ops(22).rs := 12;
C.Ops(22).dms(1) := Nill;

-- op Forward: -> Direction
C.Ops(23).Profile := 110;
C.Ops(23).Symbol := "Forward ";
C.Ops(23).rs := 15;
C.Ops(23).dms(1) := Nill;

-- op Backward: -> Direction
C.Ops(24).Profile := 110;
C.Ops(24).Symbol := "Backward ";
C.Ops(24).rs := 15;
C.Ops(24).dms(1) := Nill;

C.Num_ops := 24;
C.Obj_filename := "ring.obj ";
C.Profile_list := (110,2201,220,3211,3301,3210,Others => Nill);
C.Keyword_list := (Kw_Booch,Kw_DataStructure,Kw_Ring,Others => Nill);
Stable.Sort_table(ll).Sort_Symbol := "Ring ";
Stable.Sort_table(11).Groundterm := "empty ";

Stable.Sort_table(15).Sort_Symbol := "Direction ";
Stable.Sort_table(15).Ground_term := "forward ";
Stable.Sort_table(15).Main_sort := 15;
Stable.Sort_table(15).Sort_rank := 1;
Stable.Sort_table(15).Sort_type := Unconfined;
Stable.Sort_table(15).Csortid := Nill;
Stable.Sort_table(16).Sort_id := 16;
Stable.Sort_table(16).Sort_Symbol := "ERing"
Stable.Sort_table(16).Ground_term := "empty"
Stable.Sort_table(16).Main_sort := 11;
Stable.Sort_table(16).Sort_rank := 2;
Stable.Sort_table(16).Sort_type := Unconfined;
Stable.Sort_table(16).Csortid := Nill;

-- Bint2_obj Binary Tree version two

When Bint2_obj =>
  -- put("Working on bint2_obj"); New_line;
  -- op null: -> Tree
  C.Ops(1).Profile := 110;
  C.Ops(1).Symbol := "null"
  C.Ops(1).rs := 11;
  C.Ops(1).dms(1) := Nill;

  -- op isnull: Tree -> Bool
  C.Ops(2).Profile := 220;
  C.Ops(2).Symbol := "isnull"
  C.Ops(2).rs := 13;
  C.Ops(2).dms(1) := 11;
  C.Ops(2).dms(2) := Nill;

  -- op clear.T: Tree -> Tree
  C.Ops(3).Profile := 2201;
  C.Ops(3).Symbol := "clear.T"
  C.Ops(3).rs := 11;
  C.Ops(3).dms(1) := 11;
  C.Ops(3).dms(2) := Nill;

  -- op left: -> Child
  C.Ops(4).Profile := 110;
  C.Ops(4).Symbol := "left"
  C.Ops(4).rs := 15;
  C.Ops(4).dms(1) := Nill;

  -- op right: -> Child
  C.Ops(5).Profile := 110;
  C.Ops(5).Symbol := "right"
  C.Ops(5).rs := 15;
  C.Ops(5).dms(1) := Nill;

  -- op copy.T2: Tree Tree -> Tree
  C.Ops(6).Profile := 3301;
  C.Ops(6).Symbol := "copy.T2"
  C.Ops(6).rs := 11;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := 11;
C.Ops(6).dms(3) := Nill;

-- op itemof: Tree -> Elt
C.Ops(7).Profile := 220;
C.Ops(7).Symbol := "itemof ";
C.Ops(7).rs := 12;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := Nill;

-- op treeisnull: -> Tree
C.Ops(8).Profile := 110;
C.Ops(8).Symbol := "treeisnull ";
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := Nill;

-- op treeisnull: -> elt
C.Ops(9).Profile := 110;
C.Ops(9).Symbol := "treeisnull ";
C.Ops(9).rs := 12;
C.Ops(9).dms(1) := Nill;

-- op isequal: Tree Tree -> Bool
C.Ops(10).Profile := 3210;
C.Ops(10).Symbol := "isequal ";
C.Ops(10).rs := 13;
C.Ops(10).dms(1) := 11;
C.Ops(10).dms(2) := 11;
C.Ops(10).dms(3) := Nill;

-- op setitem.T: Elt Tree -> Tree
C.Ops(11).Profile := 3211;
C.Ops(11).Symbol := "setitem.T ";
C.Ops(11).rs := 11;
C.Ops(11).dms(1) := 12;
C.Ops(11).dms(2) := 11;
C.Ops(11).dms(3) := Nill;

-- op childof: Child Tree -> Tree
C.Ops(12).Profile := 3211;
C.Ops(12).Symbol := "childof ";
C.Ops(12).rs := 11;
C.Ops(12).dms(1) := 15;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

-- op swapchild.T1: Child Tree Tree -> Tree
C.Ops(13).Profile := 4311;
C.Ops(13).Symbol := "swapchild.T1 ";
C.Ops(13).rs := 11;
C.Ops(13).dms(1) := 15;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := 11;
C.Ops(13).dms(4) := Nil;

-- op swapchild.T3: Child Tree Tree -> Tree
C.Ops(14).Profile := 4311;
C.Ops(14).Symbol := "swapchild.T3 ";
C.Ops(14).rs := 11;
C.Ops(14).dms(1) := 15;
C.Ops(14).dms(2) := 11;
C.Ops(14).dms(3) := 11;
C.Ops(14).dms(4) := Nil;

-- op construct.T: Elt Child Tree -> Tree
C.Ops(15).Profile := 4221;
C.Ops(15).Symbol := "construct.T ";
C.Ops(15).rs := 11;
C.Ops(15).dms(1) := 12;
C.Ops(15).dms(2) := 15;
C.Ops(15).dms(3) := 11;
C.Ops(15).dms(4) := Nil;

C.Num_ops := 15;
C.Obj_filename := "bint2.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Binary_tree,Others => Nil);
C.ProfileList := (110,2201,220,3211,3301,3210,4311,4221,Others => Nil);
Stable.Sort_table(11).Sort_Symbol := "Tree ";
Stable.Sort_table(11).Ground_term := "null ";

Stable.Sort_table(15).Sort_id := 15;
Stable.Sort_table(15).Sort_Symbol := "Child ";
Stable.Sort_table(15).Ground_term := "left ";
Stable.Sort_table(15).Main_sort := 15;
Stable.Sort_table(15).Sort_rank := 1;
Stable.Sort_table(15).Sort_type := Unconfined;
Stable.Sort_table(15).Csortid := Nil;

---------------------------------------------------------------------
-- Bag.obj
---------------------------------------------------------------------

When Bag_obj =>
-- put("Working on bag_obj");new_line;
-- op emptybag : -> Bag
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "empty ";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nil;

-- op isempty : Bag -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isempty",
C.Ops(2).rs := 13;
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Null;

-- op lengthof: Bag -> Nat
C.Ops(3).Profile := 220;
C.Ops(3).Symbol := "lengthof",
C.Ops(3).rs := 14;
C.Ops(3).dms(1) := 11;
C.Ops(3).dms(2) := Null;

-- op in : Elt Bag -> Bool
C.Ops(4).Profile := 330;
C.Ops(4).Symbol := "in",
C.Ops(4).rs := 13;
C.Ops(4).dms(1) := 12;
C.Ops(4).dms(2) := 11;
C.Ops(4).dms(3) := Null;

-- op clear : Bag -> Bag
C.Ops(5).Profile := 2201;
C.Ops(5).Symbol := "clear",
C.Ops(5).rs := 11;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Null;

-- op copy: Bag Bag -> Bag
C.Ops(6).Profile := 3301;
C.Ops(6).Symbol := "copy",
C.Ops(6).rs := 11;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := 11;
C.Ops(6).dms(3) := Null;

-- op union: Bag Bag -> Bag
C.Ops(7).Profile := 3301;
C.Ops(7).Symbol := "union",
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := 11;
C.Ops(7).dms(3) := Null;

-- op and: Bag Bag -> Bag
C.Ops(8).Profile := 3301;
C.Ops(8).Symbol := "and",
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := 11;
C.Ops(8).dms(3) := Null;

166
-- op diff: Bag Bag -> Bag
C.Ops(9).Profile := 3301;
C.Ops(9).Symbol := "diff ";
C.Ops(9).rs := 11;
C.Ops(9).dms(1) := 11;
C.Ops(9).dms(2) := 11;
C.Ops(9).dms(3) := Nill;

-- op UniqueExtentOf: Bag -> Nat
C.Ops(10).Profile := 220;
C.Ops(10).Symbol := "uniqueExtentOf ";
C.Ops(10).rs := 14;
C.Ops(10).dms(1) := 11;
C.Ops(10).dms(2) := Nill;

-- op add: Elt Bag -> Bag
C.Ops(11).Profile := 3211;
C.Ops(11).Symbol := "add ";
C.Ops(11).rs := 11;
C.Ops(11).dms(1) := 12;
C.Ops(11).dms(2) := 11;
C.Ops(11).dms(3) := Nill;

-- op remove: Elt Bag -> Bag
C.Ops(12).Profile := 3211;
C.Ops(12).Symbol := "remove ";
C.Ops(12).rs := 11;
C.Ops(12).dms(1) := 12;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

-- op NumberOf: Elt Bag -> Nat
C.Ops(13).Profile := 330;
C.Ops(13).Symbol := "numberOf ";
C.Ops(13).rs := 14;
C.Ops(13).dms(1) := 12;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := Nill;

-- op singlton: Elt -> Bag
C.Ops(14).Profile := 220;
C.Ops(14).Symbol := "singlton ";
C.Ops(14).rs := 11;
C.Ops(14).dms(1) := 12;
C.Ops(14).dms(2) := Nill;

-- op isequal: Bag Bag -> Bool
C.Ops(15).Profile := 3210;
C.Ops(15).Symbol := "isequal ";
C.Ops(15).rs := 13;
C.Ops(15).dms(1) := 11;
C.Ops(15).dms(2) := 11;
C.Ops(15).dms(3) := Nil;

-- op subset: Bag Bag -> Bool
C.Ops(16).Profile := 3210;
C.Ops(16).Symbol := "subset ";
C.Ops(16).rs := 13;
C.Ops(16).dms(1) := 11;
C.Ops(16).dms(2) := 11;
C.Ops(16).dms(3) := Nil;

-- op propsubset: Bag Bag -> Bool
C.Ops(17).Profile := 3210;
C.Ops(17).Symbol := "propsubset ";
C.Ops(17).rs := 13;
C.Ops(17).dms(1) := 11;
C.Ops(17).dms(2) := 11;
C.Ops(17).dms(3) := Nil;

-- op bagerror : -> Bag
C.Ops(18).Profile := 110;
C.Ops(18).Symbol := "bagerror ";
C.Ops(18).rs := 11;
C.Ops(18).dms(1) := Nil;

C.Num_ops := 18;
C.Obj_filename := "bag.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Bag,Others => Nil);
C.ProfileList := (110,2201,220,3211,330,3301,3210,Others => Nil);
Stable.Sort_table(11).Sort_Symbol := "Bag ";
Stable.Sort_table(11).Ground_term := "empty ";

-- Array.obj

When array_obj =>
-- put("Working with array_obj"),New_line;
-- op EmptyArray : -> Array
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "emptyArray ";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nil;

-- op UnitArray : Elt -> Array
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "unitArray ";
C.Ops(2).rs := 11;
C.Ops(2).dms(1) := 12;
C.Ops(2).dms(2) := Nil;

-- op AbuttedTo: Array Array -> Array
C.Ops(3).Profile := 3301;
C.Ops(3).Symbol := "abuttedTo ";
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 11;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nill;

-- op Copy: Array Array -> Array
C.Ops(4).Profile := 3301;
C.Ops(4).Symbol := "copy ",
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := 11;
C.Ops(4).dms(3) := Mill;

-- op IsEqual: Array Array -> Bool
C.Ops(5).Profile := 3210;
C.Ops(5).Symbol := "isEqual ",
C.Ops(5).rs := 13;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := 11;
C.Ops(5).dms(3) := Nill;

-- op Isempty: Array Array -> Bool
C.Ops(6).Profile := 3210;
C.Ops(6).Symbol := "isempty ",
C.Ops(6).rs := 13;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := 11;
C.Ops(6).dms(3) := Nill;

-- op Sizeof: Array -> Nat
C.Ops(7).Profile := 220;
C.Ops(7).Symbol := "sizeof ",
C.Ops(7).rs := 14;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := Nill;

-- op Clear: Array -> Array
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "clear ",
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nill;

-- op delete: Nat Array -> Bool
C.Ops(9).Profile := 3211;
C.Ops(9).Symbol := "delete ",
C.Ops(9).rs := 13;
C.Ops(9).dms(1) := 14;
C.Ops(9).dms(2) := 11;
C.Ops(9).dms(3) := Nill;

-- op EltUnderflow: -> Elt
C.Ops(10).Profile := 110;
C.Ops(10).Symbol := "eltUnderflow";
C.Ops(10).rs := 12;
C.Ops(10).dms(1) := Nill;

C.Ops(10).Profile := 110;
C.Ops(10).Symbol := "eltUnderflow";
C.Ops(10).rs := 12;
C.Ops(10).dms(1) := Nill;

-- op NatUnderflow: -> Nat
C.Ops(11).Profile := 110;
C.Ops(11).Symbol := "natUnderflow";
C.Ops(11).rs := 13;
C.Ops(11).dms(1) := Nill;

-- op CopyError: -> Array
C.Ops(12).Profile := 110;
C.Ops(12).Symbol := "arrayError";
C.Ops(12).rs := 11;
C.Ops(12).dms(1) := Nill;

-- op ComponentOf: Nat Array -> Elt
C.Ops(13).Profile := 330;
C.Ops(13).Symbol := "componentOf";
C.Ops(13).rs := 12;
C.Ops(13).dms(1) := 14;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := Nill;

C.Num_ops := 13;
C.Obj_filename := "array.obj";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Array,Others => Nill);
C.ProfileList := (110,2201,220,3211,330,3301,3210,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "Array";
Stable.Sort_table(11).Ground_term := "emptyarray";

When list1.obj =>
-- put("Working on list1.obj");New_line;
-- op nil: -> List
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "nil";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nill;

-- op isnull: List -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isnull";
C.Ops(2).rs := 13; -- Alias to Bool sortid 4
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Nill;
-- op cons: Elt List -> List
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "cons",
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 12;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nill;

-- op tailof: List -> List
C.Ops(4).Profile := 2201;
C.Ops(4).Symbol := "tailof",
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nill;

-- op headof: List -> Elt
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "headof",
C.Ops(5).rs := 12;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nill;

-- op lengthof: List -> Nat
C.Ops(6).Profile := 220;
C.Ops(6).Symbol := "lengthof",
C.Ops(6).rs := 14;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := Nill;

-- op Listerror: -> List
C.Ops(7).Profile := 110;
C.Ops(7).Symbol := "listerror",
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := Nill;

-- op clear: List -> List
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "clear",
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nill;

-- op elterror : -> Elt
C.Ops(9).Profile := 110;
C.Ops(9).Symbol := "elterror",
C.Ops(9).rs := 12;
C.Ops(9).dms(1) := Nill;

-- op copy: List List -> List
C.Ops(10).Profile := 3301;
C.Ops(10).Symbol := "copy ";
C.Ops(10).rs := 11;
C.Ops(10).dms(1) := 11;
C.Ops(10).dms(2) := 11;
C.Ops(10).dms(3) := Nill;

-- op clearhead: List -> List
C.Ops(11).Profile := 2201;
C.Ops(11).Symbol := "clearhead ";
C.Ops(11).rs := 11;
C.Ops(11).dms(1) := 11;
C.Ops(11).dms(2) := Nill;

-- op contains: Elt List -> Bool
C.Ops(12).Profile := 330;
C.Ops(12).Symbol := "contains ";
C.Ops(12).rs := 13;
C.Ops(12).dms(1) := 12;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

-- op sethead: Elt List -> List
C.Ops(13).Profile := 3211;
C.Ops(13).Symbol := "sethead ";
C.Ops(13).rs := 11;
C.Ops(13).dms(1) := 12;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := Nill;

-- op isequal: List List -> Bool
C.Ops(14).Profile := 3210;
C.Ops(14).Symbol := "isequal ";
C.Ops(14).rs := 13;
C.Ops(14).dms(1) := 11;
C.Ops(14).dms(2) := 11;
C.Ops(14).dms(3) := Nill;

C.Num_ops := 14;
C.Obj_filename := "list1.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_List,Others => Nill);
C.ProfileList := (110,2201,220,3211,330,3301,3210,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "List ";
Stable.Sort_table(11).Ground_term := "nil ";

-----------------------------------------------------------------------
-- list2.obj Natural List
-----------------------------------------------------------------------

When List2_obj =>
   -- put("Working on List2. obj");New_line;
   -- op nil : -> List
   C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "nil 
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nill;

-- op isnull: List -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isnull 
C.Ops(2).rs := 13; -- Alias to Bool sortid 4
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Nill;

-- op cons: Nat List -> List
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "cons 
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 14;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nill;

-- op tailof: List -> List
C.Ops(4).Profile := 2201;
C.Ops(4).Symbol := "tailof 
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nill;

-- op headof: List -> Nat
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "headof 
C.Ops(5).rs := 14;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nill;

-- op lengthof: List -> Nat
C.Ops(6).Profile := 220;
C.Ops(6).Symbol := "lengthof 
C.Ops(6).rs := 14;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := Nill;

-- op Listerror: -> List
C.Ops(7).Profile := 110;
C.Ops(7).Symbol := "Listerror 
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := Nill;

-- op clear: List -> List
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "clear 
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nill;

-- op elterror : -> Nat
C.Ops(9).Profile := 110;
C.Ops(9).Symbol := "elterror 
C.Ops(9).rs := 14;
C.Ops(9).dms(1) := Nill;

-- op copy: List List -> List
C.Ops(10).Profile := 3301;
C.Ops(10).Symbol := "copy 
C.Ops(10).rs := 11;
C.Ops(10).dms(1) := 11;
C.Ops(10).dms(2) := 11;
C.Ops(10).dms(3) := Nill;

-- op clearhead: List -> List
C.Ops(11).Profile := 2201;
C.Ops(11).Symbol := "clearhead 
C.Ops(11).rs := 11;
C.Ops(11).dms(1) := 11;
C.Ops(11).dms(2) := Nill;

-- op contains: Nat List -> Bool
C.Ops(12).Profile := 330;
C.Ops(12).Symbol := "contains 
C.Ops(12).rs := 13;
C.Ops(12).dms(1) := 14;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

-- op sethead: Nat List -> List
C.Ops(13).Profile := 3211;
C.Ops(13).Symbol := "sethead 
C.Ops(13).rs := 11;
C.Ops(13).dms(1) := 14;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := Nill;

-- op isequal: List List -> Bool
C.Ops(14).Profile := 3210;
C.Ops(14).Symbol := "isequal 
C.Ops(14).rs := 13;
C.Ops(14).dms(1) := 11;
C.Ops(14).dms(2) := 11;
C.Ops(14).dms(3) := Nill;

C.Num_ops := 14;
C.Obj_filename := "list2.obj 
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_List,Others => Nill);
C.ProfileList := (110,2201,2211,330,3301,3210,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "List ";  
Stable.Sort_table(11).Ground_term := "nil ";  

-- deque obj

When deque_obj =>
-- put("Working with deque.obj"), New_line;
-- op empty : -> deque
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "empty ";
C.Ops(1).ds := 11;
C.Ops(1).dms := (Others => Nill);

-- op underflow : -> deque
C.Ops(2).Profile := 110;
C.Ops(2).Symbol := "underflow ";
C.Ops(2).ds := 11;
C.Ops(2).dms := (Others => Nill);

-- op underflow : -> elt
C.Ops(3).Profile := 110;
C.Ops(3).Symbol := "underflow ";
C.Ops(3).ds := 12;
C.Ops(3).dms := (Others => Nill);

-- op isempty: Deque -> Bool
C.Ops(6).Profile := 220;
C.Ops(6).Symbol := "isempty ";
C.Ops(6).ds := 13;
C.Ops(6).dms := (1,Others => Nill);

-- op clear: deque -> deque
C.Ops(7).Profile := 2201;
C.Ops(7).Symbol := "clear ";
C.Ops(7).ds := 11;
C.Ops(7).dms := (11,others => Nill);

-- op lengthof: deque -> Nat
C.Ops(8).Profile := 220;
C.Ops(8).Symbol := "lengthof ";
C.Ops(8).rs := 14;
C.Ops(8).dms := (11, others => Nill);

-- op frontof: deque -> Elt
C.Ops(9).Profile := 220;
C.Ops(9).Symbol := "frontof ";
C.Ops(9).rs := 12;
C.Ops(9).dms := (11, others => Nill);

-- op backof: deque -> Elt
C.Ops(10).Profile := 220;
C.Ops(10).Symbol := "backof ";
C.Ops(10).rs := 12;
C.Ops(10).dms := (11, others => Nill);

-- op isequal: deque deque -> Bool
C.Ops(11).Profile := 3210;
C.Ops(11).Symbol := "isequal ";
C.Ops(11).rs := 13;
C.Ops(11).dms := (11,11,others=> Nill);

-- op copy: deque deque -> deque
C.Ops(12).Profile := 3301;
C.Ops(12).Symbol := "copy ";
C.Ops(12).rs := 11;
C.Ops(12).dms := (11,11,others=> Nill);

-- op pop: location deque -> deque
C.Ops(13).Profile := 3211;
C.Ops(13).Symbol := "pop ";
C.Ops(13).rs := 11;
C.Ops(13).dms := (15,11,others=> Nill);

-- op add: Elt location deque -> deque
C.Ops(14).Profile := 4221;
C.Ops(14).Symbol := "add ";
C.Ops(14).rs := 11;
C.Ops(14).dms := (12,15,11,others=> Nill);

C.Num_ops := 14;
C.Obj_filename := "deque.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Deque,Others => Nill);
C.ProfileList := (110,220,220,3210,3301,3210,4221,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "Deque ";
Stable.Sort_table(11).Ground_term := "empty ";
Stable.Sort_table(15).Sort_id := 15;
Stable.Sort_table(15).Sort_Symbol := "Location ";
Stable.Sort_table(15).Ground_term := "front ",
Stable.Sort_table(15).Main_sort := 15;
Stable.Sort_table(15).Sort_rank := 1;
Stable.Sort_table(15).Sort_type := Unconfined;
Stable.Sort_table(15).Csortid := Nil;
Stable.Sort_table(15).Children_ids(1) := Nil;

-- Bint1.obj, Binary Tree Obj version one.

When Bint1_obj =>
  -- put("Working with binary tree version one"); New_line;
  -- op empty : -> bintree
  C.Ops(1).Profile := 110;
  C.Ops(1).Symbol := "empty ",
  C.Ops(1).rs := 11;
  C.Ops(1).dms(1) := Nil;

  -- op isempty: bintree -> Bool
  C.Ops(2).Profile := 220;
  C.Ops(2).Symbol := "isempty ";
  C.Ops(2).rs := 13, -- Alias to Bool sortid 4
  C.Ops(2).dms(1) := 11;
  C.Ops(2).dms(2) := Nil;

  -- op node : bintree -> elt
  C.Ops(3).Profile := 220,
  C.Ops(3).Symbol := "node ",
  C.Ops(3).rs := 12;
  C.Ops(3).dms := (11,Others => Nil);

  -- op left: bintree -> bintree
  C.Ops(4).Profile := 2201;
  C.Ops(4).Symbol := "left ",
  C.Ops(4).rs := 11;
  C.Ops(4).dms(1) := 11;
  C.Ops(4).dms(2) := Nil;

  -- op right: bintree -> bintree
  C.Ops(5).Profile := 2201;
  C.Ops(5).Symbol := "right ",
  C.Ops(5).rs := 11;
  C.Ops(5).dms(1) := 11;
  C.Ops(5).dms(2) := Nil;

  -- op height : bintree -> Nat
  C.Ops(6).Profile := 220;
  C.Ops(6).Symbol := "height ",
  C.Ops(6).rs := 14;
  C.Ops(6).dms := (11,Others => Nil);
-- op isin: Elt bintree -> Bool
C.Ops(7).Profile := 330;
C.Ops(7).Symbol := "isin";
C.Ops(7).rs := 13;
C.Ops(7).dms := (12,11,Others => Nill);

-- op naterror : -> Elt
C.Ops(8).Profile := 110;
C.Ops(8).Symbol := "naterror";
C.Ops(8).rs := 12;
C.Ops(8).dms := (Others => Nill);

-- op make: Elt bintree bintree -> bintree
C.Ops(9).Profile := 4311;
C.Ops(9).Symbol := "make";
C.Ops(9).rs := 11;
C.Ops(9).dms := (12,11,11,Others => Nill);
C.Num_ops := 9;
C.Obj_filename := "bintl.obj";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Binary_tree,Others => Nill);
C.ProfileList := (110,2201,220,330,4311,Others => Nill);
Stable.Sort_table(11).Ground_term := "empty";

-- Set.obj

When set_obj =>
-- put("Working on set_obj"); New_line;
-- op create : -> set
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "create";
C.Ops(1).rs := 11;
C.Ops(1).dms := Nill;

-- op isempty: set -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isempty";
C.Ops(2).rs := 13; -- Alias to Bool sortid 4
C.Ops(2).dms := 11;
C.Ops(2).dms(2) := Nill;

-- op add: Elt set -> set
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "add";
C.Ops(3).rs := 11;
C.Ops(3).dms := 12;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nill;

-- op clear: set -> set
C.Ops(4).Profile := 2201;
C.Ops(4).Symbol := "clear ",
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nil;

-- op cardinality: set -> Nat
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "cardinality ",
C.Ops(5).rs := 14;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nil;

-- op seterror: -> set
C.Ops(6).Profile := 110;
C.Ops(6).Symbol := "seterror ",
C.Ops(6).rs := 11;
C.Ops(6).dms(1) := Nil;

-- op copy: set set -> set
C.Ops(7).Profile := 3301;
C.Ops(7).Symbol := "copy ",
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := 11;
C.Ops(7).dms(3) := Nil;

-- op union: set set -> set
C.Ops(8).Profile := 3301;
C.Ops(8).Symbol := "union ",
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := 11;
C.Ops(8).dms(3) := Nil;

-- op intersection: set set -> set
C.Ops(9).Profile := 3301;
C.Ops(9).Symbol := "intersection ",
C.Ops(9).rs := 11;
C.Ops(9).dms(1) := 11;
C.Ops(9).dms(2) := 11;
C.Ops(9).dms(3) := Nil;

-- op remove: Elt set -> set
C.Ops(10).Profile := 3211;
C.Ops(10).Symbol := "remove ",
C.Ops(10).rs := 11;
C.Ops(10).dms(1) := 12;
C.Ops(10).dms(2) := 11;
C.Ops(10).dms(3) := Nil;
-- op isequal: set set -> Bool
C.Ops(11).Profile := 3210;
C.Ops(11).Symbol := "isequal ";
C.Ops(11).rs := 13;
C.Ops(11).dms(1) := 11;
C.Ops(11).dms(2) := 11;
C.Ops(11).dms(3) := Nill;

-- op subsetof: set set -> Bool
C.Ops(12).Profile := 3210;
C.Ops(12).Symbol := "subsetof ";
C.Ops(12).rs := 13;
C.Ops(12).dms(1) := 11;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

-- op prosubsetof: set set -> Bool
C.Ops(13).Profile := 3210;
C.Ops(13).Symbol := "prosubsetof ";
C.Ops(13).rs := 13;
C.Ops(13).dms(1) := 11;
C.Ops(13).dms(2) := 11;
C.Ops(13).dms(3) := Nill;

-- op memberof: Elt Set -> Bool
C.Ops(14).Profile := 330;
C.Ops(14).Symbol := "memberof ";
C.Ops(14).rs := 13;
C.Ops(14).dms(1) := 12;
C.Ops(14).dms(2) := 11;
C.Ops(14).dms(3) := Nill;

C.Num_ops := 14;
C.Obj_filename := "set.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Set,Others => Nill);
C.ProfileList := (110,2201,220,321 l,3210,330,3301,Others=> Nill);
Stable.Sort_table(11).Sort_Symbol := " Set ";
Stable.Sort_table(11).Ground_term := "create ";

-- stack1.obj

When Stack1_obj =>
-- put("Working on Stack1_obj"); New_line;
-- op create: -> Stack
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "create ";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nill;

-- op isempty: Stack -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isempty"
C.Ops(2).rs := 13; -- Alias to Bool sortid 4
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Nil;

-- op push: Elt Stack -> Stack
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "push"
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 12;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nil;

-- op pop: Stack -> Stack
C.Ops(4).Profile := 2201;
C.Ops(4).Symbol := "pop"
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nil;

-- op top: Stack -> Elt
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "top"
C.Ops(5).rs := 12;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nil;

-- op stacksize: Stack -> Nat
C.Ops(6).Profile := 220;
C.Ops(6).Symbol := "stacksize"
C.Ops(6).rs := 14;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := Nil;

-- op emptyerror: -> Stack
C.Ops(7).Profile := 110;
C.Ops(7).Symbol := "emptyerror"
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := Nil;

-- op clear: stack -> Stack
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "clear"
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nil;

C.Num_ops := 8;
C.Obj_filename := "stack1.obj";
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Stack,Others => Nil);
C.ProfileList := (220,2201,110,3211,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "Stack ";
Stable.Sort_table(11).Ground_term := "create ";

-- queue_obj

When queue_obj =>
-- put("Working on queue"); New_line;
-- op empty: -> queue
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "empty ";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nill;

-- op pop.Q : queue -> queue
C.Ops(2).Profile := 2201;
C.Ops(2).Symbol := "pop.Q ";
C.Ops(2).rs := 11;
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Nill;

-- op add.Q: Elt queue -> queue
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "add.Q ";
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 12; -- Elt generic sort
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nill;

-- op frontof: queue -> elt
C.Ops(4).Profile := 220;
C.Ops(4).Symbol := "frontof ";
C.Ops(4).rs := 12;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nill;

-- op isempty: queue -> Bool
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "isempty ";
C.Ops(5).rs := 13; -- Alias to 4 for Bool
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nill;

-- op underflow: -> queue
C.Ops(6).Profile := 110;
C.Ops(6).Symbol := "underflow ";
C.Ops(6).rs := 11;
C.Ops(6).dms(1) := Nill;

-- op underflow: -> Elt
C.Ops(7).Profile := 110;
C.Ops(7).Symbol := "underflow ";
C.Ops(7).rs := 12;
C.Ops(7).dms(1) := Nill;

-- op lengthof: queue -> Nat
C.Ops(8).Profile := 220;
C.Ops(8).Symbol := "lengthof ";
C.Ops(8).rs := 14;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nill;

-- op isequal: queue queue -> Bool
C.Ops(9).Profile := 3210;
C.Ops(9).Symbol := "isequal ";
C.Ops(9).rs := 13;
C.Ops(9).dms(1) := 11;
C.Ops(9).dms(2) := 11;
C.Ops(9).dms(3) := Nill;

-- op copy.Q2: queue queue -> queue
C.Ops(10).Profile := 3301;
C.Ops(10).Symbol := "copy.Q2 ";
C.Ops(10).rs := 11;
C.Ops(10).dms(1) := 11;
C.Ops(10).dms(2) := 11;
C.Ops(10).dms(3) := Nill;

-- op clear.Q : queue -> queue
C.Ops(11).Profile := 2201;
C.Ops(11).Symbol := "clear.Q ";
C.Ops(11).rs := 11;
C.Ops(11).dms(1) := 11;
C.Ops(11).dms(2) := Nill;

C.Num_ops := 11;
C.Obj_filename := "queue.obj ";
C.ProfileList := (110,2201,220,3210,3301,3210,Others => Nill);
C.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Queue,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "Queue ";
Stable.Sort_table(11).Ground_term := "empty ";

-----------------------------------------------
-- stack2.obj
-----------------------------------------------

When others =>
-- put("Working on stack2_obj"); New_line;
-- op create: -> Stack
C.Ops(1).Profile := 110;
C.Ops(1).Symbol := "create ";
C.Ops(1).rs := 11;
C.Ops(1).dms(1) := Nil;

-- op isempty: Stack -> Bool
C.Ops(2).Profile := 220;
C.Ops(2).Symbol := "isempty ";
C.Ops(2).rs := 13; -- Alias to Bool sortid 4
C.Ops(2).dms(1) := 11;
C.Ops(2).dms(2) := Nil;

-- op push: Elt Stack -> Stack
C.Ops(3).Profile := 3211;
C.Ops(3).Symbol := "push ";
C.Ops(3).rs := 11;
C.Ops(3).dms(1) := 12;
C.Ops(3).dms(2) := 11;
C.Ops(3).dms(3) := Nil;

-- op pop: Stack -> Stack
C.Ops(4).Profile := 2201;
C.Ops(4).Symbol := "pop ";
C.Ops(4).rs := 11;
C.Ops(4).dms(1) := 11;
C.Ops(4).dms(2) := Nil;

-- op top: Stack -> Elt
C.Ops(5).Profile := 220;
C.Ops(5).Symbol := "top ";
C.Ops(5).rs := 12;
C.Ops(5).dms(1) := 11;
C.Ops(5).dms(2) := Nil;

-- op depthof: Stack -> Nat
C.Ops(6).Profile := 220;
C.Ops(6).Symbol := "depthof ";
C.Ops(6).rs := 14;
C.Ops(6).dms(1) := 11;
C.Ops(6).dms(2) := Nil;

-- op copy: Stack Stack -> Stack
C.Ops(7).Profile := 3301;
C.Ops(7).Symbol := "copy ";
C.Ops(7).rs := 11;
C.Ops(7).dms(1) := 11;
C.Ops(7).dms(2) := 11;
C.Ops(7).dms(3) := Nil;

-- op clear: stack -> Stack
C.Ops(8).Profile := 2201;
C.Ops(8).Symbol := "clear ";
C.Ops(8).rs := 11;
C.Ops(8).dms(1) := 11;
C.Ops(8).dms(2) := Nill;

-- op empty : -> Stack
C.Ops(9).Profile := 110;
C.Ops(9).Symbol := "empty ",
C.Ops(9).rs := 11;
C.Ops(9).dms(1) := Nill;

-- op StackError : -> Stack
C.Ops(10).Profile := 110;
C.Ops(10).Symbol := "StackError ",
C.Ops(10).rs := 11;
C.Ops(10).dms(1) := Nill;

-- op StackError : -> Elt
C.Ops(11).Profile := 110;
C.Ops(11).Symbol := "StackError ",
C.Ops(11).rs := 12;
C.Ops(11).dms(1) := Nill;

-- op isequal : Stack Stack -> Bool
C.Ops(12).Profile := 3210;
C.Ops(12).Symbol := "isequal ",
C.Ops(12).rs := 13;
C.Ops(12).dms(1) := 11;
C.Ops(12).dms(2) := 11;
C.Ops(12).dms(3) := Nill;

C.Num_ops := 12;
C.Obj_filename := "stack2.obj ";
C.KeywordList := (Kw_Booch,Kw_DataStracture,Kw_Stack,Others => Nill);
C.ProfileList := (220,2201,110,3211,3301,3210,Others => Nill);
Stable.Sort_table(11).Sort_Symbol := "Stack ",
Stable.Sort_table(11).Ground_term := "create ";
End Case;

-- Now, common sorts Initialization
Stable.Sort_table(11).Sort_id := 11;
Stable.Sort_table(11).Main_sort := 11;
Stable.Sort_table(11).Sort_rank := 1;
Stable.Sort_table(11).Sort_type := Unconfined;
Stable.Sort_table(11).Csortid := Nill;
Stable.Sort_table(11).Children_ids(1) := Nill;
Stable.Sort_table(11).Parent_ids(1) := Nill;
Stable.Sort_table(12).Sort_id := 12;
Stable.Sort_table(12).Sort_Symbol := "Elt ";
Stable.Sort_table(12).Main_sort := 12;
Stable.Sort_table(12).Sort_rank := 1;

185
Stable.Sort_table(12).Sort_type := Unconfined;
Stable.Sort_table(12).Sortid := Nill;
Stable.Sort_table(12).Children_ids(1) := Nill;
Stable.Sort_table(12).Parent_ids(1) := Nill;

Stable.Sort_table(13) := Stable.Sort_table(Bool_type); -- Alias to bool

Stable.Sort_table(14) := Stable.Sort_table(Nat_type); -- Alias to nat

KwRank := KeywordRank(Q,C);
PrfRank := ProfileRank(Q,C);
End InitComponentData;

Function LookupBlockIndex(Pindex: Natural) return Integer is
Profile_table : Array (0..18) of Integer :=
(0,110,2201,220,321 l,330,3301,3210,4311,4221,Nill,11,12,13,14,15,16,17,18);
Begin
For i in 0.. 18
Loop
If Profile_table(i) = Pindex then
Return i;
End If;
End Loop;
Return(Nill);
End LookupBlockIndex;

Procedure UpdateCandidatesTable(Candidates: Out CandidatesTable;
H : In Out Hasse_diagram_type;node: Integer;Eoa: In Out
Natural) is
Begin
-- Get component(s) in current node
For i in 1..Component_id_list'Length
Loop
If H(LookupBlockIndex(node)).Component_list(i) /= Nill then
Candidates(Eoa).ComponentId := H(LookupBlockIndex(node)).Component_list(i);
Eoa := Eoa + 1;
Else
Candidates(Eoa).ComponentId := Nill;
Exit;
End If;
End Loop;
End Loop;
H(LookupBlockIndex(node)).Visit := True; -- Mark node visit

-- now get component in children nodes
For i in 1..Children_id_list'Length
Loop
If H(LookupBlockIndex(node)).Children_list(i) /= Nill then
For j in 1..Component_id_list'Length
Loop
If H(LookupBlockIndex(H(LookupBlockIndex(node)).Children_list(i))). Visit = False then
    If H(LookupBlockIndex(H(LookupBlockIndex(node)).Children_list(i))). Component_List(j) /= Nill then
        Candidates(Eoa). ComponentId := H(LookupBlockIndex(H(LookupBlockIndex(node)).Children_list(i))). Component_List(j);
        Eoa := Eoa + 1;
    Else
        Candidates(Eoa). ComponentId := Nill;
        Exit;
    End If;
End If;
End Loop;
End UpdateCandidatesTable;

Procedure FindCandidateComponents(Q: In Out Qc; Candidates: In Out CandidatesTable; Profile_err: Out Profile_err_list) is
    H : Hasse_diagram_type;
    m : Natural := 1;
    Eoa : Natural := 1;
    Begin
    -- Check for valid frequency in each test cases
    For i in 1..Q.Num_cases Loop
        For j in 1..ProfileList_defLength Loop
            If Q.Geq(i).ProfileList(j) /= Nill Then
                If LookupBlockIndex(Q.Geq(i).ProfileList(j)) = Nill then
                    Profile_err(m) := Q.Geq(i).ProfileList(j);
                    m := m + 1;
                    Profile_err(m) := Nill;
                    Q.Geq(i). Status := Void;
                End If;
            End If;
        End Loop;
    End Loop;
    -- Initialize Hasse diagram with components from CS4520 course
    InitializeSwb(H);
    -- Now, start searching
    PerformDFsw(H, Q.RootNode, Candidates, Eoa);
    End FindCandidateComponents;

Function DetermineProfileIntersection(GeqProfile:Profilelist_def; Pprofile: Profile_id_list) Return Boolean is

187
Found : Boolean;
Begin
For i in 1..Profilelist_length
Loop
  Found = False;
  If GeqProfile(i) /= Nill then
    For j in 1..Profile_id_list'Length
      Loop
        If GeqProfile(i) = Pprofile then
          Found := True;
          Exit;
        End If;
      End Loop;
    If Found = False then
      Return False;
    End If;
  Else
    Exit;
  End If;
End Loop;
Return True;
End DetermineProfileIntersection;

Procedure PeformDfsfw(H: In Out Hasse_diagram_type;Q: Qc; node: Integer;
Candidates: Out CandidatesTable;Eoa : In Out Natural) is
Begin
H(LookupBlockIndex(node)). Visit := True; -- Mark node visit
For j in 1..Q.Num_cases
  Loop
    If Q.Geq(j).Status /= Void then
    If DetermineProfileIntersection(Q.Geq(j).ProfileList,
      H(LookupBlockIndex(node)).Profile_List) then
      UpdateCandidatesTable(Candidates,H,node,Eoa),
      Exit;
    End If;
  End Loop;
For j in 1..Children_id_list'Length
  Loop
    If H(LookupBlockIndex(node)).Children_list(j) /= Nill then
      If H(LookupBlockIndex(H(LookupBlockIndex(node)).Children_list(j))).Visit = False then
        PeformDfsfw(H,Q,H(LookupBlockIndex(node)).Children_list(j),Candidates,Eoa);
      End If;
    End If;
  End Loop;
End PeformDfsfw;

Procedure InitializeSwb(H : In Out Hasse_diagram_type) is
Begin
-- Root node
H(LookupBlockIndex(0)).Pipro := True;
H(LookupBlockIndex(0)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(0)).Children_list :=
(110,2201,220,3211,330,3301,3210,4311,4221,Others => Nil);
H(LookupBlockIndex(0)).Component_list := (Others => Nil);
H(LookupBlockIndex(0)).Profile_list := (0,Others => Nil);

-- -> A profile, P1
H(LookupBlockIndex(110)).Pipro := True;
H(LookupBlockIndex(110)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(110)).Children_list := (12,17,Others => Nil);
H(LookupBlockIndex(110)).Component_list := (Others => Nil);
H(LookupBlockIndex(110)).Profile_list := (110,Others => Nil);

-- A -> A profile, P2
H(LookupBlockIndex(2201)).Pipro := True;
H(LookupBlockIndex(2201)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(2201)).Children_list := (12,17,Others => Nil);
H(LookupBlockIndex(2201)).Component_list := (Others => Nil);
H(LookupBlockIndex(2201)).Profile_list := (2201,Others => Nil);

-- A -> B profile, P3
H(LookupBlockIndex(220)).Pipro := True;
H(LookupBlockIndex(220)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(220)).Children_list := (12,17,Others => Nil);
H(LookupBlockIndex(220)).Component_list := (Others => Nil);
H(LookupBlockIndex(220)).Profile_list := (220,Others => Nil);

-- A B -> A profile, P4
H(LookupBlockIndex(3211)).Pipro := True;
H(LookupBlockIndex(3211)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(3211)).Children_list := (12,Others => Nil);
H(LookupBlockIndex(3211)).Component_list := (Others => Nil);
H(LookupBlockIndex(3211)).Profile_list := (3211,Others => Nil);

-- A B -> C profile, P5
H(LookupBlockIndex(330)).Pipro := True;
H(LookupBlockIndex(330)).Visit := False;
H(LookupBlockIndex(330)).Children_list := (14,17,Others => Nil);
H(LookupBlockIndex(330)).Component_list := (Others => Nil);
H(LookupBlockIndex(330)).Profile_list := (330,Others => Nil);

-- A A -> A profile, P6
H(LookupBlockIndex(3301)).Pipro := True;
H(LookupBlockIndex(3301)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(3301)).Children_list := (14, Others => Nill);
H(LookupBlockIndex(3301)).Component_list := (Others => Nill);
H(LookupBlockIndex(3301)).Profile_list := (3301, Others => Nill);

-- A A -> B profile, P7
H(LookupBlockIndex(3210)).Pipro := True;
H(LookupBlockIndex(3210)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(3210)).Children_list := (13, Others => Nill);
H(LookupBlockIndex(3210)).Component_list := (Others => Nill);
H(LookupBlockIndex(3210)).Profile_list := (3210, Others => Nill);

-- A B B -> B profile, P8
H(LookupBlockIndex(4311)).Pipro := True;
H(LookupBlockIndex(4311)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(4311)).Children_list := (18, Others => Nill);
H(LookupBlockIndex(4311)).Component_list := (Others => Nill);
H(LookupBlockIndex(4311)).Profile_list := (4311, Others => Nill);

-- A B C -> B profile, P9
H(LookupBlockIndex(4221)).Pipro := True;
H(LookupBlockIndex(4221)).Visit := False;
-- No need to put all children for pi prime node
H(LookupBlockIndex(4221)).Children_list := (16, Others => Nill);
H(LookupBlockIndex(4221)).Component_list := (Others => Nill);
H(LookupBlockIndex(4221)).Profile_list := (4221, Others => Nill);

-- Profile index 1,2,3,4, P12 Stack2
H(LookupBlockIndex(12)).Pipro := False;
H(LookupBlockIndex(12)).Visit := False;
H(LookupBlockIndex(12)).Children_list := (13,14,15,16,18, Others => Nill);
H(LookupBlockIndex(12)).Component_list := (Stack2_obj, Others => Nill);
H(LookupBlockIndex(12)).Profile_list := (110,2201,220,3211,3301,3210, Others => Nill);

-- Profile index 1,2,3,4,6,7, P13 , Stack1, Queue, Ring
H(LookupBlockIndex(13)).Piro := False;
H(LookupBlockIndex(13)).Visit := False;
H(LookupBlockIndex(13)).Children_list := (15,16,18, Others => Nill);
H(LookupBlockIndex(13)).Component_list := (Stack1_obj, Ring_obj, Queue_obj, Others => Nill);
H(LookupBlockIndex(13)).Profile_list := (110,2201,220,3211,3301,3210, Others => Nill);

-- Profile index 1,2,3,4,5,6, P14, Set
H(LookupBlockIndex(14)).Piro := False;
H(LookupBlockIndex(14)).Visit := False;
H(LookupBlockIndex(14)).Children_list := (15, Others => Nill);
H(LookupBlockIndex(14)).Component_list := (Set_obj, Others => Nill);
H(LookupBlockIndex(14)).Profile_list := (110,2201,220,3211,3301,3300, Others => Nill);

-- Profile index 1,2,3,4,5,6,7, P15a, Array, Bag, List1, List2
H(LookupBlockIndex(15)).Pipro := False;
H(LookupBlockIndex(15)).Visit := False;
H(LookupBlockIndex(15)).Children_list := (Others => Nil);
H(LookupBlockIndex(15)).Component_list := (Array_obj,Bag_obj,List1_obj,List2_obj,Others => Nil);
H(LookupBlockIndex(15)).Profile_list := (110,220,220,321 l,330,3301,3210,Others => Nil);

-- Profile index 1,2,3,4,6,7,9, P16, Deque
H(LookupBlockIndex(16)).Pipro := False;
H(LookupBlockIndex(16)).Visit := False;
H(LookupBlockIndex(16)).Children_list := (18,Others => Nil);
H(LookupBlockIndex(16)).Component_list := (Deque_obj,Others => Nil);
H(LookupBlockIndex(16)).Profile_list := (110,220,220,3211,3301,3210,4221,Others => Nil);

-- Profile index 1,2,3,5,8, P17, Bint1
H(LookupBlockIndex(17)).Pipro := False;
H(LookupBlockIndex(17)).Visit := False;
H(LookupBlockIndex(17)).Children_list := (Others => Nil);
H(LookupBlockIndex(17)).Component_list := (Bint1_obj,Others => Nil);
H(LookupBlockIndex(17)).Profile_list := (110,220,220,330,4311,Others => Nil);

-- Profile index 1,2,3,4,6,7,8,9, P18, Bint2
H(LookupBlockIndex(18)).Pipro := False;
H(LookupBlockIndex(18)).Visit := False;
H(LookupBlockIndex(18)).Children_list := (Others => Nil);
H(LookupBlockIndex(18)).Component_list := (Bint2_obj,Others => Nil);
H(LookupBlockIndex(18)).Profile_list := (110,220,220,3211,3301,3210,4311,4221,Others => Nil);

End InitializeSwb;

End Swb_pkg;

--- Module Name: getquery.a
--- Description: This module initialize stack query data.
--- Author: Nguyen Doan
--- History: Nov, 6 1995
--- Verified as a first prototyped

With Global_def; Use Global_def;
With Swb_def_pkg; Use Swb_def_pkg;
With Float_io; Use Float_io;
With Integer_io; Use Integer_io;
With Text_io; Use Text_io;
Package Query_Processing_pkg is
Procedure Get_Query(Q: out QC; Stable: in out Query_table_def);
End Query_Processing_pkg;

Package body Query_processing_pkg is

Procedure Get_Query(Q: out QC; Stable: in out Query_table_def) is
-- Query spec
-- op empty -> Stack
Q.Ops(1).Profile := 110;
Q.Ops(1).Symbol := "Empty";
Q.Ops(1).rs := 21;
Q.Ops(1).dms(1) := Nill;

-- op top Stack -> Nat
Q.Ops(4).Profile := 220;
Q.Ops(4).Symbol := "Top";
Q.Ops(4).rs := 22; -- Nat alias to sortid 3
Q.Ops(4).dms(1) := 21;
Q.Ops(4).dms(2) := Nill;

-- op push Nat Stack -> Stack
Q.Ops(2).Profile := 3211;
Q.Ops(2).Symbol := "Push";
Q.Ops(2).rs := 21;
Q.Ops(2).dms(1) := 22; -- Nat alias to sortid 3
Q.Ops(2).dms(2) := 21;
Q.Ops(2).dms(3) := Nill;

-- op pop Stack -> Stack
Q.Ops(3).Profile := 2201;
Q.Ops(3).Symbol := "Pop";
Q.Ops(3).rs := 21;
Q.Ops(3).dms(1) := 21;
Q.Ops(3).dms(2) := Nill;

-- op depthof: Stack -> Nat
Q.Ops(5).Profile := 220;
Q.Ops(5).Symbol := "Depthof";
Q.Ops(5).rs := 22;
Q.Ops(5).dms(1) := 21;
Q.Ops(5).dms(2) := Nill;

Q.Num_ops := 4;
Q.Num_cases := 1;

Q.Geq(1).ProfileList := (220,3211,110,2201, Others => Nill);
Q.Geq(1).Eqtext := "Top(Push(1,Empty)) == Top(Pop(Push(6,Push(l,Empty))))";

Q.Geq(2).ProfileList := (220,3211,110,Others => Nill);
Q.Geq(2).Eqtext := "Depthof(Push(l,Push(2,Push(3,Empty)))) == 3";

Q.KeywordList := (Kw_Booch,Kw_DataStructure,Kw_Stack,Others => Nill);
Q.ProfileList := (2201,3211,110,220,Others => Nill);

Q.Geq(1).Status := Unvoid;
Q.Geq(2).Status := Unvoid;
-- Initialize predefine sort
Stable :=
(5,(1,
"FLOAT       ",1,Basic,1,Nill,(2,3,others => Nill),(others => Nill),
(others => ' '),
(2,
"INT         ",1,Basic,2,Nill,(3,others => Nill),(others => Nill),
(others => ' '),
(3,
"NAT         ",1,Basic,3,Nill,(others => Nill),(others => Nill),
(others => ' '),
(4,
"BOOL        ",4,Basic,1,Nill,(others => Nill),(others => Nill),
(others => ' '),
(5,
"CHAR        ",5,Basic,1,Nill,(others => Nill),(others => Nill),
(others => ' '),
others => -- To be initialized by program
(Nill,
"           ",Nill,Nill,Nill,Nill,(others=>Nill),(others=>Nill),
(others => ' ')));

-- Now initialize sort table for query sort
Stable.Sort_table(21).Sort_id := 21;
Stable.Sort_table(21).Sort_Symbol := "Stack       ";
Stable.Sort_table(21).Main_sort := 21;
Stable.Sort_table(21).Sort_rank := 1;
Stable.Sort_table(21).Sort_type := Unconfined;
Stable.Sort_table(21).Csortid := Nill;
Stable.Sort_table(21).Children_ids(1) := Nill;
Stable.Sort_table(21).Parent_ids(1) := Nill;

Stable.Sort_table(22) := Stable.Sort_table(Nat_type);

End Get_Query;
End Query_processing_pkg; -- End of query processing

-- Module Name: fresult.a
-- Description: This module calculated the total rank and formulate the
-- result and output to the user
-- Author: Nguyen Doan
-- History: Nov, 6 1995
-- Verified as a first prototyped

With Global_def; Use Global_def;
With text_io; Use text_io;
With integer_io; Use integer_io;
With Float_io; Use Float_io;

Package Formulate_Result_Output_pkg is
Procedure Display_invalid_operations(Q: Qc;Profile_err: Profile_err_list);
Procedure Sort_Display_Result(SortArray :In Out Asort_Array;Count : Natural);
Procedure Calculate_total_rank(V : Signature_Map; Q: Qc; C:SWC;
   Idx,Num_vmaps: Natural;SortArray : In Out Asort_Array;
   KeywordRank,ProfileRank: Float);
End Formulate_Result_Output_pkg;

Package body Formulate_Result_Output_pkg is

Procedure Display_invalid_operations(Q: Qc;Profile_err: Profile_err_list) is
Begin
If Profile_err(1) /= Nill then
   New_line;
   Put("The following operation(s) is unknown: "); New_line;
   For i in 1..Profile_err_list'Length
      Loop
         If Profile_err(i) = Nill then
            Exit;
         Else
            For k in 1..Q.Num_ops
               Loop
                  If Q.Ops(k).Profile = Profile_err(i) then
                     Put(String(Q.Ops(k).Symbol)); New_line;
                  End If;
               End Loop;
            End If;
         End Loop;
      End If;
   End Loop;
End If;
End Display_invalid_operations;

Procedure Sort_Display_Result(SortArray :In Out Asort_Array;Count : Natural) is
Smallest: Integer;
Temp : SortRank;
Begin
   -- Selection Sort
   For j in 1 ..Count-1
      Loop
         Smallest := j;
         For q in j+1 .. Count
            Loop
               If SortArray(q).MaxoverallRank < SortArray(Smallest).MaxoverallRank then
                  Smallest := q;
               End If;
            End Loop;
         End If;
      End Loop;
   If smallest > j then
      Temp := SortArray(Smallest);
      SortArray(Smallest) := SortArray(j);
      SortArray(j) := Temp;
   End If;
End Sort_Display_Result;
End If;
End Loop;
-- Now display result
For i in reverse 1..Count
Loop
If SortArray(i).Mapnum /= 0 then
    New_line;
    put("Find Component: ");
    put(String(SortArray(i).ModuleName)); New_line;
    put("Using Map Number: ");
    put(SortArray(i).Mapnum); New_line;
    put("KeywordRank: ");
    put(Float(SortArray(i).SelKeywordRank),4,1);
    put("ProfileRank: ");
    put(Float(SortArray(i).SelProfileRank),4,1);New_line;
    put("SignatureRank: ");
    put(Float(SortArray(i).SelSignatureRank),4,1);
    put("SemanticRank: ");
    put(Float(SortArray(i).SelSemanticRank),4,1);New_line;
    put("The ComponentRank: ");
    put(Float(SortArray(i).MaxoverallRank),4,1);New_line;
Else
    New_line;
    put("Find Component: ");
    put(String(SortArray(i).ModuleName)); New_line;
    put("KeywordRank: ");
    put(Float(SortArray(i).SelKeywordRank),4,1);
    put("ProfileRank: ");
    put(Float(SortArray(i).SelProfileRank),4,1);New_line;
    put("No map is found!");New_line;
End If;
End Loop;
End Sort_Dispaly_Result;

Procedure Calculate_total_rank(V : Signature_Map; Q: Qc; C:SWC; 
    Idx,Num_vmaps: Natural;SortArray : In Out AsortArray;
    KeywordRank,ProfileRank: Float) is
    IntermRank, MaxoverallRank : Float := 0.0;
    SelSemanticRank,SelSignatureRank : Float := 0.0;
    Mapnum : Natural := 0;
    Begin
    SortArray(idx).ModuleName := C.Obj_filename;
    SortArray(idx).SelKeywordRank := KeywordRank;
    SortArray(idx).SelProfileRank := ProfileRank;
    For i in 1..Num_vmaps
    Loop
        Intern_Rank :=
ProfileRank * KeywordRank * v(i).SemanticRank * v(i).SignatureRank;
if Interm_Rank > SortArray(idx).MaxoverallRank then
    SortArray(idx).SelSignatureRank := v(i).SignatureRank;
    SortArray(idx).SelSemanticRank := v(i).SemanticRank;
    SortArray(idx).MaxoverallRank := Interm_Rank;
    SortArray(idx).Mapnum := i;
End If;
end loop;

If SortArray(Idx).SelSemanticRank <= 0.0 then
    -- Recalculate SelSignatureRank
    SelSignatureRank := 0.0;
    For i in 1..Num_vmaps
        Loop
            If SortArray(idx).SelSignatureRank <= V(i).SignatureRank then
                SortArray(idx).SelSignatureRank := V(i).SignatureRank;
                SortArray(idx).Mapnum := i;
            End If;
        End Loop;
    End If;
End If;
End Calculate_total_rank;

End Formulate_Result_Output_pkg; -- End of Formulate_Result_Output_pkg
APPENDIX D - RUNOBJ AND SCS BUILD FILES

#########################################################################
# This is runobj script file. It is invoked from the SCS to do perform term rewriting on the
# translate ground equations.
# Author: Nguyen, Doan
# Date: 12/4/95
#########################################################################
obj < testfile.obj | grep result | sed -e 's/.*result // > testrun.dat

#########################################################################
# This is SCS build file. It is used to build the SCS prototype
# Author: Nguyen, Doan
# Date: 12/4/95
#########################################################################
a.cleanlib
ada gldef.a; ada asable.a; ada semops.a; ada utilop.a; ada utilso.a;
ada nsa.a; ada sigmat.a; ada semata; ada fresult.a; ada swb_def.a;
ada getquery.a; ada swb.a; ada init.a; ada scs.a
a.ld Main -o SCS
APPENDIX E - A PROOF FOR THEOREM 1

In this appendix contains two parts. First, we will show that $\equiv$ is an equivalence relation on the transitivity-closure(symmetry-closure( $\leq$ ))). Secondly, we will prove the Theorem 1.

A. ASSUMPTION: The subsort relation $\leq$ is a partial ordering relation.

B. DEFINITION: The relation $\equiv$ is defined to be the transitive closure of symmetry closure of the symmetry closure of $\leq$.

A. SHOW $\equiv$ IS AN EQUIVALENCE RELATION

- Reflexivity: Yes, because $x \leq x$. $x \equiv x$.
- Symmetry: Yes, because the symmetric closure of any relation is symmetric and $R \leq$ transitive closure $R$.
- Transitivity: Yes, because the transitive closure of any relation is transitive for any relation $R$.

The important consequence of this equivalence relation is that sort group are equivalence classes and any two equivalence classes must be equal or disjoint. This is important for our next proof.

B. PROOF OF THEOREM 1

Theorem 1: Given a query operation and a component operation with their corresponding profile values, if these profile values are not equal, then these operations can not possibly be related by a signature match.

Proof:

- Case 1: Consider query and component operators which do not have the same number of sort occurrences. The profile values for both operators are different
by rule 1 of definition 4.

By the requirement 7 of section IV.D, the number of argument sorts, n, must be the same for both operators. Therefore, these query and component operators can not be matched.

Case 2: Consider query and component operators which have the same number of sort occurrences. The number of sort group in query operator is less than component operator. The profile values for both operators are different by rules 2, 3, and 4 of definition 4.

Since the number of sort groups of query is less than the sort groups of component, the correspond mapping will be result in an non injective map. By the requirement 1 of section IV.D, the sort mapping must be injective. Therefore, these query and component operators can not be matched.

Case 3: Consider query and component operators which have the same number of sort occurrences. The number of sort groups in query operator is greater than component operator. The profile values for both operators are different by rules 2, 3, and 4 of definition 4.

Since the number of sort groups is of query operator is greater than the sort groups of component operator, the correspond mapping will be result in an non injective map. By the requirement 1 of section IV.D, the sort mapping must be injective. Therefore, these query and component operators can not be matched.

Case 4: Consider query and component operators which have the same number of sort occurrences and the same number of sort groups. The cardinality of one sort group in query operator is not equal to a cardinality of any operator’s sort groups. The profile values for both operators are different by rules 2, 3, and 4 of definition 4.

Since one query sort group has a cardinality different from any other sort group of the component, there are one of the two possible violations of the mapping can be happened. First, the injective sort mapping is violated. One of the sort in this sort group in the query would map to more than two components sorts or two or more sort in this sort group in the query would map to one component’s sort. Second, the requirement 7 of section IV.D is violated if any sort from query or component operator is left dangling. Therefore, these query and component operators can not be matched.
INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center ................................................................. 2
   8725 Joh J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Library, Code 013 ................................................................................................. 2
   Naval Postgraduate School
   Monterey, California 93943-5101

3. Chairman, Code CS ......................................................................................... 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Dr. Luqi, Code CS/Lq ..................................................................................... 5
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Dr. Valdis Berzins, Code CS/Vb ................................................................. 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Dr. Man-tak Shing, Code CS/Sh .................................................................... 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000

7. Mr. Nguyen, Doan ......................................................................................... 2
   8213 Tevin Way
   Sacramento, California 95828