ABSTRACT: The verification of Java/C++ codes is critical, especially for special projects where human life will be at stake. A system is required that uses integrated reasoning to split each verification condition into a conjunction of simpler sub formulas, then apply a diverse collection of specialized decision procedures, first-order theorem proves, and, in the worst case, interactive theorem provers to prove each sub formula. There exist some commercial tools for the verification of Java/C++ code such as Jahob. However, none of the commercially available tools does a good job at finding bugs dealing with concurrency. Most of them are focusing on testing (test drivers, test cases), and, some are using more advanced techniques such as static analysis (Coverity, Klocwork, PolySpace, Code Sonar to name only a few). Techniques such as replacing complex sub formulas with stronger but simpler alternatives, exploiting structure inherently present in the verification condition, and, when necessary, inserting verified lemmas and proof hints into the imperative source code make it possible to seamlessly integrate all of the specialized decision procedures and theorem provers into a single powerful integrated reasoning system. By appropriately applying multiple proof techniques to discharge different sub formulas, this reasoning system can effectively prove the complex and challenging verification conditions that arise in this context.

Keywords -- JAHOB; VERIFICATION; IMPLEMENTATION; SPECIFICATION

I. INTRODUCTION

In computer science, a linked data structure is a data structure which consists of a set of data record (node) linked together and organized by reference (links or pointer).

In linked data structures, the links are usually treated as special data type that can only be dereference or compared for equality. Linked data structures are thus contrasted with array and other data structures that require performing arithmetic operations on pointers. This distinction holds even when the nodes are actually implemented as elements of a single array, and the references are actually Linked data structures include linked list, search tree, expression tree, and many other widely used data structures.

The goal of the project is to implement some simple linked data structures, provide their specifications, and use Jahob to verify the implementation. A subset of Java is chosen as the implementation language because it is a familiar, imperative, memory-safe language and existing Java tools can be used to write and compile the programs.

Linked data structures such as lists, trees, graphs and hash tables are pervasive in modern software systems. But because of phenomena such as aliasing and indirection, it has been a challenge to develop automated reasoning systems that are capable of proving important correctness properties of such data structures.

Our specifications use abstract sets and relations to characterize the abstract state of the data structure. A verified abstraction function defines abstract sets and relations in terms of concrete objects and references that the implementation manipulates at run time. Specifications use abstract sets and relations to state externally visible properties of the data structure state and to specify method preconditions and post conditions. These specifications capture all of the semantic information that the developer needs to use the data structure.

Reasoning about a small fragment of a program at a time makes the verification process easier and more scalable, and allows the use of very specific, expressive properties. Jahob also allows the use of specification variables to support data abstraction. These variables are like concrete Java variables but they exist only to help the programmer reason about the abstract properties of the data structure (as opposed to its concrete implementation) in order to simplify the verification process. Specification variables can be used to denote sets, functions and relations.

II. REQUIREMENT FOR CODE VERIFYING

The Constellation project at NASA calls for designing a Crew Exploration Vehicle (Orion, also called CEV) and Cargo Launch Vehicle (Ares, also called CLV).
Both projects will rely on newly designed flight control Software. Yet both software for Orion and Ares are expected to be multi-threaded. With this work we are proposing to address the issue by developing a suite of tools that can be used to verify C++ code.

C++ is a very complex language in terms of verification. C++ relies on powerful (from a coding point of view) but complex (from a verification point of view) constructs such as templates, which are addressed through their instances (rather than the generic form) in verification, dynamic object allocation, which abundant use in C++ code renders static analysis difficult, implicit (and explicit) constructors and destructors can have confusing behaviours, multiple inheritance also has perverse effects, and overloaded operators can also be tricky.

This multitude of features is even more difficult to verify when they are combined (as is routinely done in C++ programs).

2.1. WHAT IS JAHOB?

**Jahob:** Jahob was developed by Viktor Kuncak and his collaborators as part of his PhD work at MIT. Jahob verification system uses these techniques for verification.

**First-order theorem provers:** By appropriately applying multiple proof techniques to discharge different Sub formulas, this system can effectively prove the complex and challenging verification conditions that arise in this field.

Using integrated reasoning in to verify a diverse set of challenges in linked data structure implementations.

Figure 1 summarizes key aspects of the Jahob architecture, illustrating the provers that Jahob uses to establish data structure correctness. Jahob is a command-line tool, with command-line options that support verification at the granularity of methods or even individual assertions. Jahob produces verification conditions by simplifying Java code and transforming it into extended guarded commands, desugaring extended guarded commands into simple guarded commands, and finally generating verification conditions from simple guarded commands in a standard way.

Jahob verification system: Jahob can be viewed as a verification system for modular analysis of imperative computer programs that manipulate relations as state components.

A subset of Java was chosen as the implementation language because it is a familiar, imperative, memory-safe language and existing Java tools can be used to write and compile the programs. The input language for Jahob is a subset of Java extended with annotations written as special comments. Therefore, Jahob programs can be compiled and executed using existing Java compilers and virtual Machines, and Statically verified to satisfy important data structure consistency properties.

2.2. BASIC VERIFICATION APPROACH

Jahob generates verification condition formulas, then proving these formulas for the preconditions, written as a requires clause, specify the properties that should be true at the entry point of the procedure the frame condition, written as a modifies clause, specifies which parts of the program state the procedure is allowed to modify during its execution.

The post conditions, written as an ensures clause, specify the properties that should be true after the procedure is finished executing. Operations such as array accesses and pointer dereferences.

III. PROBLEM STATEMENT

Upon examination, however, it becomes clear that while the verification conditions as a whole can be quite complex, they can also be represented as a conjunction of a large number of smaller subformulas, many of which are straightforward to prove. Moreover, the remaining sub formulas, while containing a diverse group of powerful logical constructs, often have enough structure to enable the successful application of specialized decision procedures or theorem provers. Specifically, some sub formulas can be proved with sufficient quantifier instantiations, congruence closure algorithms, and linear arithmetic solvers; precise reasoning about reach ability is sufficient to discharge others; still others require complex quantifier reasoning but do not require arithmetic reasoning. In the worst case, it is always possible to
use interactive theorem provers to discharge the remaining few complex sub formulas.

How to split verification conditions? How to form an approximate formula? How to decompose the proof?

Our main criticism regarding the use of Jahob for data structure verification is the fact that the automation starts to break down even when trying to prove some properties that at first glance seem simple.

Some programs require more specifications that actual code, and although we mentioned that we saw the utility in doing this, there is a limit to how much time and effort a programmer is willing to spend to ensure the complete correctness of his/her implementation.

IV. PROPOSED WORK

**Circular List:** A circular doubly-linked list implementing a set interface.

**Singly-Linked List:** A null-terminated singly-linked list implementing a set interface.

**Software verification tools.** Software verification tools based on theorem proving include, ESC/Java, Krakatoa, and KeY[1]. We are not aware of any effort to use these systems to verify that a significant collection of data structure implementations conforms to their specifications, nor of any other system that integrates the diversity of theorem provers and decision procedures that Jahob does.

3.1. BENEFITS FOR PROGRAMMERS

The programmer can reason about the modular components (classes and procedures) independently, verifying each procedure separately and then assume the correctness of the procedures when verifying the whole program.

Reasoning about a small fragment of a program at a time makes the verification process easier and more scalable, and allows the use of very specific, expressive properties.

In practice, Each of these reasoning systems proves the subset of sub formulas for which it is applicable; When the automation does not succeed (typically due to conjunction that contain large numbers of universally quantified assumptions), we manually guide the proof process by inserting verified lemmas and proof hints into the source code.

<table>
<thead>
<tr>
<th>Data Structures</th>
<th>SPASS</th>
<th>CVC3</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREE MAP</td>
<td>80(142.6s)</td>
<td>22(2.0s)</td>
<td>172.17</td>
</tr>
<tr>
<td>ARRAYLIST</td>
<td>16(66.7s)</td>
<td>2(9.9s)</td>
<td>161.1</td>
</tr>
<tr>
<td>HASH TABLE</td>
<td>222(58.3)</td>
<td></td>
<td>73.65</td>
</tr>
<tr>
<td>LINK LIST</td>
<td>94(5.9s)</td>
<td></td>
<td>6.94</td>
</tr>
<tr>
<td>VECTOR</td>
<td>99(2.5s)</td>
<td></td>
<td>70.91</td>
</tr>
</tbody>
</table>

**Figure 2** Sample Timing for different modules

Figure 2 illustrates how Jahob effectively combines the capabilities of multiple theorem provers and decision procedures to verify sophisticated data structure correctness properties. It also illustrates how the different capabilities of these theorem provers and/or decision procedures are necessary to obtain the full functional correctness proofs.

**Figure 3** Chart for Execution Timing for Jahob & Code

Finally, we use interactive theorem provers to prove verification conditions that require complex inductive reasoning or nonlinear arithmetic, relying on a body of previously proved lemmas and the ability to fully control the proof process.

V. CONCLUSION

Full functional verification has long been viewed as an impractical or even unrealizable goal. The results here demonstrate, for the first time, that this goal is within practical reach for linked data structure implementations. These results are especially compelling given the widespread reuse of data structure libraries and the central role that linked data structures play in computer science. In the near future it is not unreasonable to expect to see core data structure libraries shipped only after
full functional specification and verification. Jahob verification system

a. Automation by translation HOL®FOL
b. omitting sorts theorem gives speedup
c. filtering automates selection of assumptions
d. Promising experimental results
e. strong properties: correct implementation
f. Do not crash operations correctly update the content, clarifies behavior in case of duplicate keys, representation invariants preserved (ordering, treeness, each element is in appropriate bucket) relatively fast
g. Verification efforts much smaller than interactive proverb.

In the near future it is not unreasonable to expect to see core data structure libraries shipped only after full functional specification and verification. Full functional verification has long been viewed as an impractical or even unrealizable goal. The results in this paper demonstrate that this goal is within practical reach for linked data structures.

Using the Jahob system, we have verified many of the data structures that programmers use in practice. Our results are especially compelling given the widespread reuse of data structure libraries and the central role that linked data structures play in computer science. Fully verified the implementations of a singly-linked list with a header node and a queue, and all methods of a singly-linked cyclic list, doubly-linked cyclic list and leaf-linked tree also verify, but these classes require further work to ensure the specifications are correct in the initial state of the class.

When we began working on this project, we implemented very simple data structures which were easy to implement correctly. Therefore, as we were writing the specifications for those implementations, we were mostly sure they were correct, and a problem with the verification usually resulted from a problem with the specification rather than with the implementation.

We enjoyed using Jahob because we were already familiar with HOL, so we did not have too many problems with the syntax of the specifications. Another positive aspect of Jahob is its modular approach to data structure verification; verifying fragments of code independently made the task of verifying an entire data structure much easier.

Our main criticism regarding the use of Jahob for data structure verification is the fact that the automation starts to break down even when trying to prove some properties that at first glance seem simple.

Some programs require more specifications that actual code, and although we mentioned that we saw the utility in doing this, there is a limit to how much time and effort a programmer is willing to spend to ensure the complete correctness of his/her implementation.

VI. FUTURE WORK

Verify more examples such as balanced trees fancy priority queues (binomial, Fibonacci, …) hash table with dynamic resizing hash function verify clients of data structures Improve assumption filtering take rarity of symbols into account check for occurring polarity...

We can also check for the Time complexity for each and every linear data structures.

VII. ACKNOWLEDGMENT

We are thankful to Prof. Mrs. Roshni Dubey for his contributions to my verification process, which include the implementation of field constraint analysis.

VIII. REFERENCES:


