Adapting the Gypsy Verification System to Ada

Workshop on Formal Specification and Verification of Ada
Institute for Defense Analysis
18-20 March 1985

John McHugh - Research Triangle Institute
Karl Nyberg - Verdix Corporation

1. Introduction

DoD directive 5000.31 [DoD] requires that new mission critical computer programs written for the department of defense be written in Ada[1] [Ada]. The statutory definition of mission critical (10 USC 2315) includes security applications specifically. Computer security has been one of the principle driving forces for applied verification work in recent years. These factors lead us to one of two conclusions: 1) The time is rapidly approaching when it will be necessary to apply verification techniques to programs written in Ada; or 2) DoD 5000.31 will have to be modified to exclude secure systems. While there exists a well known antipathy towards Ada within parts of both the verification and the computer security communities, it is unlikely that the DoD policy towards Ada will undergo substantial change in the near future. If this is the case, it will be necessary to develop an Ada verification capability in the near future.

There are several ways in which such a capability could be developed. A first option would be to start from scratch, using any of the formal models of program specification and verification and build a system specifically designed to verify Ada programs. A second option is to ignore the Ada specific aspects of the problem entirely. Under the current certification criteria of the DoDCSC, it is not necessary to deal with the implementation language for a system in a formal manner, so it could be argued that current systems are just as suitable (or unsuitable) for Ada as for any other language. In this case, it is only necessary to provide a convincing argument for the conformance of the Ada implementation code to the verified formal top level specification of the system in question. Finally, it is possible to adapt an existing verification system to deal with Ada.

The first approach is possible, but would take an excessive amount of time and resources. Current verification systems represent investments of ten or more man years each, expended over periods of five to ten years. The second approach is representative of the practice followed for the Honeywell SCOMP, a product currently approaching A1 certification by the DoDCSC. It appears that the requirement for a convincing argument concerning the equivalence of the FTLS and the implementation resulted in an extremely complex and concrete FTLS and greatly increased the verification effort. Being able to verify an Ada based FTLS for an Ada based implementation should

[1] Ada is a registered trademark of the Ada Joint Project Office.

obviate these difficulties. Additionally, there is substantial interest in systems which go beyond the A1 criteria by requiring code verification for which second approach would not be viable. The third approach offers a chance to capture much of the investment in a current verification system while gaining experience with the verification of Ada. We argue for such an approach, based on the Gypsy [Good78] system, suggesting that it will lead to a prototype code verification system for Ada with minimum (although not insubstantial by any means) effort. Taking advantage of the Ada packaging mechanism, we feel that verified packages can function within a larger Ada environment, making possible the implementation of security kernels and the like.

The remainder of the paper discusses some of the problems associated with the verification of Ada, suggests ways in which these problems might be addressed, and indicates ways in which the Gypsy system could be combined with the front end of an Ada compiler and transformed into a prototype system for the verification of Ada.

2. Trouble spots in Ada

Although one of the early design objectives for Ada (in the days when it was still known as DoD-1) was to facilitate proofs of program properties, the committee nature of the requirements process resulted in a language which was required to carry a certain amount of the baggage of 1960s style programming languages. Among the potentially most troublesome of these are the presence of arbitrary control flow constructs i.e. the "go to" statement, and unrestricted access to global variables which, in addition to complicating proofs about sequential programs, render concurrent programs intractable under many circumstances. Other features of the language include the possibility of side effects from function invocations, exceptions during expression evaluation, and the lack of an explicit evaluation order for the operators of an expression. These factors, combined with the lack of a formal definition for the semantics of the language, have lead some workers to despair of verifying any aspect of the language. Indeed, it has been noted that given the proper Ada context, it may be impossible to prove anything about the value of X after the execution of so simple a statement as

```
X := 1;
```

We maintain that the situation is not quite as grim as indicated above. Just because a language contains a particular feature does not mean that all programs written in the language will contain that feature. The adverse interaction among features of the language, does not mean that all of them must be discarded, or that all occurrences of a feature in a given program are intractable. Although the word "subset" is an anathema to the Ada world, we feel that a useful set of Ada constructs and programming practices can be defined in such a way that realistic and functional programs can written and verified using them. Although the task is substantially more difficult, because of the extra complexity of the language, we feel that a theory of verifiable Ada can be developed in much the same way as Boyer and Moore developed their FORTRAN

[Boyer80] theory. Platek [Odyssey84] and his colleagues at Odyssey Associates have recently defined an initial subset of Ada which they feel is suitable for verification. One feature which they rule out is the exception mechanism. We feel that the Ada exception mechanism is sufficiently like the Gypsy mechanism so that its verification is tractable, and we propose to include exceptions in our system.

Ada as currently defined has no specification mechanism. While it is possible to use an external specification mechanism, i.e. one in which the program and specification are joined only during the verification process, we are more comfortable with an internal mechanism, similar to that used in Gypsy. At the same time, we would like our verifiable code to be acceptable to a variety of Ada translators. An extension of Luckham's Anna notation [Luckham84] to accommodate exception returns from routines appears to be the most promising mechanism available at the present time, although a specification language using the Ada PRAGMA construct cannot be ruled out.

3. A hybrid system

We propose to base our prototype Ada verification system on a combination composed of an existing Ada compiler and an existing verification system. The Ada compiler is the one developed and recently validated by the Verdix corporation of McLean, Virginia, while the verification system is the Gypsy Verification Environment, developed at the University of Texas. There are several reasons for the choice of such a hybrid system. Ada is a large language with a complex syntax and semantics. Using an existing front end from a validated compiler eliminates much of the effort required to implement a front end for the verification system. It also provides a direct method for providing executable versions of the verified programs, as well as facilitating systems which contain mixtures of verified and unverified programs. The use of a modified version of the GVE as a back end for the Ada verification system offers similar advantages. We feel that the initial set of Ada constructs which can be verified will be roughly equivalent in power and flavor to the Gypsy language. Previous efforts to model Ada constructs in Gypsy [Akers83], and vice versa provide evidence for this assumption. Although Ada type rules are "stronger" than those for Gypsy, it is possible to write Gypsy as though it were typed like Ada. The Gypsy exception mechanism, though somewhat more tractable than the Ada exception mechanism is suitable for modeling Ada. Most of the Ada operators are already present in Gypsy.

The proposed hybrid consists of three primary components, the Ada front end, the intermediate form translator, and the verification back end. Each of these are described briefly in the sections which follow.

4. The Ada front end

As noted above, the front end of the proposed system is based on the parser and semantic checker of an existing, validated, Ada compiler. The parser and semantic checker will require some modifications to accept Ada with embedded specifications. The output of the modified front end will consist of the compiler's internal representation of Ada programs, extended to include the specification constructs. Assuming that a specification language such as Anna is chosen, these modifications should be relatively straight forward. The internal representation will be captured at a stage in the compilation process where name resolution has been performed and operator overloading has been removed so as to simplify subsequent operations.

5. The intermediate form translator

The intermediate form translator serves a dual purpose. Its primary function is to convert the Ada compiler's representation of a program into a representation which can be entered into the verification back end as though it were the output of the Gypsy parser. Its secondary function is to ensure that the code to be verified conforms to the set of constructs acceptable to the verification system, i.e. that the program to be verified is in fact written in the verifiable Ada subset. Given that both the Ada front end and the Gypsy back end use internal representations which are abstractions of prefix trees, the translation operation is a straightforward, if complex, syntactic one. The enforcement function, on the other hand, may involve substantial semantic analysis. It is hoped to simplify both of these tasks by taking advantage of utilities, already present within the front end, for manipulating the internal form of Ada programs.

6. The modified GVE

The output of the translation process will be a Gypsy-like representation of the Ada code to be verified in a form suitable for loading into the modified GVE. Once such an Ada database has been restored into the GVE, verification conditions can be generated and proved in the same way these steps are performed for Gypsy programs in current versions of the system. To support Ada verification, substantial modifications will be required for a number of components of the GVE. The verification condition generator will require modification to reflect the semantic differences between Ada and Gypsy statements. In a similar fashion, the expression simplifier will also require modification and extension. The prefix to infix conversion routine, used to display internal forms to the user will be modified to use an Ada syntax. We hope to take advantage of the previous work on a Gypsy to Ada translator for much of this step. It is hoped that the prover will require little or no modification. Modifications to the top-level or user interface to the system should be restricted to the removal of unneeded functionality

and system components such as the optimizer and code generators.

7. Summary and conclusions

We have proposed a prototype Ada verification system based on a hybrid of an existing compiler and verification system. Although such a system is not capable of supporting verification of the entire Ada language, it is claimed that it will support a language comparable to those now being verified and suitable for similar programs. While the construction of such a system involves a substantial effort, we are confident that the effort is much less than that involved in building a verification system for Ada from scratch. A hybrid system, such as we propose, will allow the verification community and the growing applications community it supports to obtain experience with Ada verification in the near future. Such experience will provide a sound basis for future revisions of the language to support verification should this prove necessary or desirable.

8. References

[Ada] - *Ada Programming Language,* ANSI/MIL-STD-1815A, Department of Defense, 22 January 1983.

[Akers83] - Akers, Robert L., *A Gypsy-to-Ada Program Compiler,* Technical Report 39, Institute for Computing Science, The University of Texas at Austin, Austin, TX 78712, December 1983.

[Boyer80] - Boyer, Robert S., Moore, J Strother, *A Verification Condition Generator for Fortran,* Technical Report CSL-103, SRI International, June, 1980.

[DoD] - DeLauer, Richard D., "DoD Policy on Computer Programming Languages", Department of Defense Directive 5000.31, The Under Secretary of Defense, Washington, DC, 20301, June 1983.

[Good78] - Good, Donald I., Cohen, Richard M., Hoch, Charles G., Hunter, Lawrence W., Hare, Dwight F., *Report on the Language Gypsy, Version 2.0,* Technical Report ICSCA-CMP-10, Institute for Computing Science and Computer Applications, The University of Texas at Austin, Austin, TX 78712, September 1978.

[Luckham84] - Luckham, David C., von Henke, Friedrich W., Krieg-Brueckner, Bernd, Owe, Olaf, *Anna - A Language for Annotating Ada Programs, Preliminary Reference Manual,* Technical Report No. 84-261, Program Analysis and Verification Group, Computer Systems Laboratory, Stanford University, Stanford, CA 94305, July 1984.

[Odyssey84] - Odyssey Research Associates, Inc. *A Verifiable Subset of Ada,* (Revised

Preliminary Report), Odyssey Research Associates, Inc., 713 Clifton St., Ithaca, NY 14850, 14 September 1984.