Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

260

David C. Luckham
Friedrich W. von Henke
Bernd Krieg-Brückner
Olaf Owe

ANNA
A Language for Annotating Ada Programs

Reference Manual

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

Authors

David C. Luckham
Computer Systems Laboratory, Stanford University
Stanford, CA 94305-2192, USA

Friedrich W. von Henke
SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025, USA

Bernd Krieg-Brückner
FB 3 Mathematik-Informatik, Universität Bremen
Postfach 330 440, 2800 Bremen 33
Federal Republic of Germany

Olaf Owe
Institute of Informatics, University of Oslo
P.O. Box 1080, Blindern, 0316 Oslo 3, Norway

CR Subject Classification (1987): D.2.1, D.2.2, D.2.4, D.2.5, D.2.10, D.3.2, I.2.2, I.2.4

ISBN 3-540-17980-1 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-17980-1 Springer-Verlag New York Berlin Heidelberg

Library of Congress Cataloging-in-Publication Data. ANNA, a language for annotating Ada programs. (Lecture notes in computer science; 260) Bibliography: p. Includes index. 1. ANNA (Computer program language) 2. Ada (Computer program language) I. Luckham, David C. II. Title: ANNA. III. Series.
QA76.73.A54A56 1987 005.13 87-13030
ISBN 0-387-17980-1 (U.S.)

Printed in Germany

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210

Table of Contents

PREFACE

Anna is a language extension of Ada to include facilities for formally specifying the intended behavior of Ada programs. It is designed to meet a perceived need to augment Ada with precise machine-processable annotations so that well established formal methods of specification and documentation can be applied to Ada programs.

The current Anna design includes annotations of all Ada constructs except tasking. Similar extensions for formal specification can be made to other Algol-like languages such as Pascal, PL-1, Concurrent Pascal, and Modula; essentially, these extensions would be subsets of Anna.

The design of Anna was initiated in 1980 by Bernd Krieg-Brueckner and David Luckham. An outline of some of the main features of Anna was presented at the Ada Symposium held in Boston, December 1980 [12]. This was based on the preliminary Ada design [11]. The current Anna design is a significant development from the 1980 version and was undertaken by the four present authors over the period 1981 — 1984. This effort proceeded in parallel with the Ada language revisions. It was sponsored by the Defense Advanced Research Projects Agency as part of the research project of the Program Analysis and Verification Group at Stanford University. The current Anna design corresponds to the latest Ada design, ANSI/MIL-STD 1815A.

Design Goals

The design of Anna was undertaken from the beginning with four principal considerations:

1. Constructing annotations should be easy for the Ada programmer and should depend as much as possible on notation and concepts of Ada.

2. Anna should possess language features that are widely used in the specification and documentation of programs.

3. Anna should provide a framework within which the various established theories of formally specifying programs may be applied to Ada.

4. Annotations should be equally well suited for different possible applications during the life cycle of a program. Such applications include not only testing, debugging and formal verification of a finished program, but also specification of program parts during the earlier stages of requirements analysis and program design.

Adaption to Ada

Goal (1) has had a major influence on both the syntax and semantics of Anna. The Anna syntax is a straightforward extension of the Ada syntax. Formal comments occur within the Ada comment framework. Anna programs are therefore accepted by Ada compilers. Ada concepts such as scope and visibility, elaboration, and generic instantiation apply to annotations. Most new concepts in Anna are extensions of concepts in Ada. For example Boolean expressions are extended to allow quantification. Collections of designated values associated with access types are available in annotations using the attribute notation of Ada. The central specification concept in Anna, the declarative annotation, is a generalization of the constraint concept in Ada and obeys the same Ada scope rules.

Basic Specification Techniques

Goal (2) requires that the underlying annotation language should be at least as powerful as any first order logic; i.e., it should contain the syntactic category of Ada expressions extended by logical quantifiers and implication operators. This choice is clearly dictated by the fact that most comments (informal or formal) are Boolean relationships between program variables. Also, in practice, annotations will often contain partially defined expressions, so the semantics of Anna must define clearly the meaning of such annotations.

Previous studies in program specification also indicate that inclusion of Ada text as formal comments — called virtual Ada text — should be permitted in Anna. This provides a powerful method of constructing executable specifications and syntactically separating them from the subject text.

Based on these decisions, the simplest kinds of annotations in Anna permit previous techniques for specifying Pascal programs to be applied to Ada [7, 10, 15, 17].

Established Specification Methods

Goal (3) is concerned with providing facilities for applying various established specification techniques that go beyond the simple comment or assertional techniques. For example, the most successful method of specifying access variable manipulations requires that annotations may refer to objects that are not available in the programming language, i.e., Collections [15]. (See also [Ada83 3.8 (3)].) Therefore access type collections and standard operations on them are provided in Anna as predefined attributes. Similarly, techniques for specifying module constructs by means of sequences of operations have been developed in previous literature, e.g., [16, 4] and [13]. Facilities for specifying packages in Anna include package states and sequences of state transitions. These facilities, together with the package axiom annotation, also enable the Anna user to apply other specification methods, e.g., the various algebraic methods of specifying abstract data types [8, 5], to packages.

To allow the specification of programs that may raise or propagate exceptions, propagation annotations are included in a notation adapted to Ada from [14].

Applications of Formal Annotations

Goal (4) is concerned with developing new applications of formal annotations. Use of Anna is not intended to be restricted to only verification of existing programs in the conventional proof-theoretic sense although this is clearly a major possible future application. A significant part of this document is devoted to describing a method of transforming annotations into runtime checks. It is applicable to most kinds of annotations. Consequently, an Anna program (or, in the most general case, large parts of an Anna program) can be transformed into an Ada program with runtime checks for consistency with the original annotations. This method can be implemented more easily than a formal verification system, and will provide a useful tool for testing and debugging Ada programs.

Anna is also intended for use during the design of a program, e.g., for formal specification of subprograms and packages prior to implementation of their bodies. Anna specifications may be generated as part of the development process from earlier requirements [3], and may then accompany a program through all stages of its development. (See also [2].)

Future Versions of Anna

Our philosophy in the current Anna design has been to provide a minimal set of basic kinds of annotations and predefined annotation concepts. It is the user's responsibility to define the special concepts needed for an application. As experience with Ada specifications and annotations accumulates we expect to revise and expand future versions.

It is certainly true that some features of Ada make it difficult to design an adequate annotation language. Any attempt to provide for every possibility, or to overly generalize facilities, will lead to a very complicated design.

The problems are caused to some extent by the basic set of Ada constructs, for example, access types and tasking. Specification of access type manipulations has always been a problem in previous languages such as Pascal, and one wishes that high level languages would progress to a stage where access types were omitted. Previous facilities appear to be the bare minimum for specification but do not seem to make the job easy in any real sense. The current facilities in Anna for specifying access type manipulations in particular, and composite types in general, may be extended in future versions.

Anna does not include any specific annotations for tasking. Some annotation can be provided for tasks by means of the facilities for subprograms. In general we feel that research in the overall area of specification of concurrent computation needs to progress before we make any extensions to Anna for tasking.

Most complications in the Anna design, however, are caused by the generality and permissiveness of Ada visibility rules, naming, overloading, and elaboration. In some cases the complicated Ada rules spill over into Anna; this we cannot avoid. In others, e.g., packages, the specialized annotations will not be useful if the programmer uses Ada in the wildest possible ways. Certainly, part of useful annotation is disciplined use of Ada; this matter will be addressed in one of the forthcoming Anna documents.

Anna can be modified for use as a program design language (PDL) at early stages of the software development process. This possibility is being studied and seems to improve on existing PDLs. It would provide automated analysis for some kinds of design errors during early systems design phases.

Documents and Support Tools

The preliminary reference manual is the first of a number of documents and planned support tools for Anna. Documents in preparation include: (1) An introduction to the use of Anna. This will include examples of various applications of formal annotations and a rationale for certain parts of the design. (2) Transformations from annotations to Ada runtime checks. (3) An axiomatic semantics of Anna.

Anna support tools currently being developed include : (1) syntax analyzers, structured editors, and tools for detecting simple kinds of errors; (2) a runtime checking system that will translate most annotations into Ada runtime checks. Development of this system as part of an environment of tools for debugging and testing by directly executing an Ada program against its Anna specifications is planned. A formal verification system is regarded as a longer-term undertaking.

Acknowledgements

Numerous comments on the November 1982 version of the preliminary Anna manual have been helpful in revising the design and producing this present manual. We owe special thanks to Norman Cohen, Anthony Gargaro, and Alec D. Hill for detailed reviews of that document which were most helpful. We are also greatly indebted to Rosemary Brock for her consistent efforts in preparing the manuscripts of the various versions of this manual over the past two years.

Acknowledgement to the first revision

This revision incorporates many of the corrections and suggestions over the past two years arising from the use of Anna and from the implementation of tools to support its applications. Our thanks are due to Doug Bryan, Chris Byrnes, Rob Chang, Geoff Mendal, Randall Neff, David Rosenblum, Sriram Sankar, and Will Tracz. As before, we are again indebted to Rosemary for overseeing the manuscript.

This work was supported by the Advanced Research Projects Agency, Department of Defense, under contracts N00039-82-C-0250 and N00039-84-C-0211.