Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

135

R.L. Constable
S.D. Johnson
C.D. Eichenlaub

An Introduction to the PL/CV2 Programming Logic

Springer-Verlag
Berlin Heidelberg New York 1982

Authors

R.L. Constable
S.D. Johnson
C.D. Eichenlaub
Cornell University, Dept. of Computer Science
405 Upson Hall, Ithaca, NY 14853, USA

ISBN 3-540-11492-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-11492-0 Springer-Verlag New York Heidelberg Berlin

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

PREFACE

This book is based on the reference manual for the PL/CV Programming Logic and on lecture notes used to teach the logic to first year college students. The Programming Logic consists of a formal system for reasoning about integers, arrays, and programming language commands (in the PL/I dialect called PL/CS). The arguments can be checked by the PL/CV Proof Checker (available in PL/I and in C, see [-14-]). The programs can be executed by PL/CS compilers (see [-10-]), including the Cornell Program Synthesizer ([-19-]) and the Cornell Program Environment.

The notes are written from the point of view that computer programming is formal algorithmic problem solving. The subject is formal because problem solutions must be written so that a computer can execute them. In some cases this formality can be extended to the entire argument which led to the solution, and the computer can be used to verify the argument.

In cases when the entire argument can be formalized, there are obvious advantages to doing so. For one thing, one's confidence in the solution is appreciably increased. This observation has been the basis for research in the subject called program verification (see the discussion in [5,6]). Another advantage of formalization is pedagogical - one is able to see the complete structure of the argument and explain it to someone who is learning to reason algorithmically. This is the same advantage that rigorous argument offers to any subject, and is a justification for teaching formal logic in the college curriculum.

Various computer systems to check proofs have been employed in the teaching of formal logic ([-13,17-]). We feel that such systems can play an especially interesting role in computer science courses. In the first place, programming courses by necessity teach a great deal of formalism and logic. For example, the treatment of boolean expressions in modern programming languages is an introduction to the propositional calculus, and the definition of a program state and its effect on assertions in programs is the same as the concept of an interpretation in the predicate calculus.

In the second place, the concepts of program verification, especially the notions of asserted program, weakest pre-condition, loop invariant, procedure call rules, etc. have an increasing place in the computer science curriculum. A rigorous treatment of these concepts is close to a formal treatment in a very high level logic such as PL/CV. A formal treatment allows computer assistance in teaching the subject. In particular, the student can experiment with forms of argument in private and at his own pace.

In the third place, the Proof Checker, like the language translator, is an interesting piece of computer software. Exposing students to it will enhance their appreciation of the potential of computer automation.

For these reasons we feel it is appropriate to teach a programming logic in the computer science curriculum. These notes can be used for that purpose. They introduce a completely formal programming logic, PL/CV2. The logic and its Proof Checker were designed by R. L. Constable, S. D. Johnson, and M. J. O'Donnell. The logic has been reported in the book A Programming Logic [-6-] and in various articles [-5,7,14-], and the Proof Checker is described in [-7-] and in the book A Computer System for Checking Proofs [-14-]. The underlying programming language PL/CS is described in the textbook [-11-]. The system is a merging of the predicate calculus and the Floyd-Hoare style of reasoning about programs [-16-]. It was designed to be simple, conventional, high level and efficient so that it could be used in college courses, and so that it could be used to explore elementary program verification.

The odd-numbered chapters introduce topics informally at a very elementary level, and the even-numbered chapters provide a succinct and precise summary of the logic (which may be skipped on a first reading). Numerous examples are provided, and all of the complete proofs have been checked by the Proof Checker (PL/I version). The exposition in the odd-numbered sections is oriented toward the reader with almost no programming experience. A more advanced account of the logic appears in A Programming Logic.

Experience with the system

We have used PL/CV2 at Cornell to teach logic and basic program verification in a sophomore discrete mathematics course. Our experiences here have been very positive. We also used PL/CV2 to teach introductory programming. We found that students were overwhelmed by the amount of formalism to be grasped at first encounter. We surmise that the system could be successfully used in a second course on programming to help teach the basics of programming methodology.

The interactive synthesizer version improves useability of PL/CV by a factor of 2 or 3 over the batch oriented system. We have not yet used that system in a course however.

The PL/CV programming logic has been considerably extended to include a rich constructive theory of types. In this language, called PL/CV3 it appears possible to formalize the kind of non-elementary algorithmic problem solving exhibited in such textbooks as The Design and Analysis of Computer Algorithms by Aho, Hopcroft and Ullman. The language was designed to allow a feasible formalization of any argument solving a sequential algorithmic problem

The reader interested in the concept of a constructive formal logic as a programming tool should follow the work of the Cornell Automated Logic group. The project on Program Refinement Logics, PRL, is building a programming system which extracts executable code fom formal constructive proofs (see [-1-]).

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the National Science Foundation; the project started under MCS-76-14293 and continued under MCS-78-00953. Finally the grant SED-79-18966 allowed us to experiment with PL/CV in the classroom and provided the impetus for the expository chapters of the monongraph.

The PL/CV1 logic was designed with Michael J. O'Donnell whose active interest and keen insights have shaped the project at every stage. Our initial effort relied on the stability of the PL/C system and the support of its director, Richard Conway.

Our faculty colleagues at Cornell provided both constructive criticism and encouragement. In particular we thank Corky Cartwright, Alan Demers, Jim Donahue, and David Gries. Work on the interactive system, AVID, owes a great deal to Tim Teitelbaum and the Cornell Program Synthesizer Project.

Many former students were active participants in discussions of the logic and implementation. Carl Hauser was co-author of the first manual. Gary Levin and Barry Bakalor were especially helpful.

Special thanks are due the contributing authors, all of whom have also worked on the implementation, including Tat-hung Chan, Dean B. Krafft, Ryan Stansifer and Daniel Zlatin.

Michelle Fish prepared large parts of the manuscript using the UNIX text editing facilities. We are grateful for her very careful work.

Finally we thank our department and its chairman, Juris Hartmanis, who have provided such a stimulating and tolerant atmosphere for our work.

R. L. Constable
S. D. Johnson
C. D. Eichenlaub

Ithaca, N.Y.
August 1981

TABLE OF CONTENTS