M.-C. Gaudel J.-P. Jouannaud (Eds.)

TAPSOFT '93: Theory and Practice of Software Development

4th International Joint Conference CAAP/FASE,
Orsay, France, April 13-17, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Marie-Claude Gaudel
Jean Pierre Jouannaud
LRI, Université de Paris-Sud et CNRS (UA 410)
Bâtiment 490, 91405 Orsay Cedex, France

CR Subject Classification (1991): D.1-3, F.1-3

6276

ISBN 3-540-56610-4 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56610-4 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author/editor
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

This volume contains the proceedings of the fourth International Joint Conference on the Theory and Practice of Software Development, TAPSOFT'93. Its predecessors were held in Berlin (1985), Pisa (1987), Barcelona (1989) and Brighton (1991). TAPSOFT'93 is being held from April 13 to April 17, 1993, in Orsay.
Since its creation in 1985, the aim of this conference has been to bring together theoretical computer scientists and researchers in software engineering with a view to discussing how formal methods can usefully be applied in software development.
Continuing with this tradition, TAPSOFT'93 consists of three parts: an advanced seminar, and two colloquiums, CAAP and FASE.

• The 1993 issue of the *Advanced Seminar* includes invited surveys by :
H-D. Ehrich, J. Guttag, C. Jones, B. Mahr, W. Thomas,
and invited conferences by:
A. Arnold, P-P. Degano, N. Dershowitz, G. Longo.

• The *Colloquium on Trees in Algebra and Programming* (CAAP) is held annually in conjunction either with TAPSOFT or ESOP and focuses on the theories underlying the overall theme of TAPSOFT. This year, the selected papers are organised in seven sessions: Specifications and Proofs, Concurrency, Automata and Counting, Constraints Solving, Rewriting, Logic and Trees, Analysis of Algorithms, plus a common session with FASE on Type Inference.

The program committee of CAAP'93 is the following: A. Arnold*, N. Dershowitz, H. Ganzinger*, J. Goguen, J-P. Jouannaud* (Chair), J-W. Klop*, D. Kosen, U. Montanari*, M. Nivat*, L. Pacholski*, B. Rovan*, W. Thomas*.

• The *Colloquium on Formal Approaches of Software Engineering* (FASE) focuses on formal methods and techniques for innovative software development. The selected papers are presented in the following sessions: Case Studies in Formal Design and Development, Compositionality Modules and Development, Formal Development, Foundations and Analysis of Formal Specifications, Verification of Concurrent Systems, Model Checking, Parallel Calculus, plus a common session with CAAP on Type Inference.

The program committee of FASE is the following: E. Astesiano*, M. Dinchas*, H. Erhig*, M-C. Gaudel*(Chair), S. Gerhart, D. Jacobs*, C. Jones*, T. Maibaum*, F. Orejas*, J. Sifakis, A. Tarlecki*.

The TAPSOFT'93 proceedings are published in a single volume, which departs from the tradition. It reflects the convergence of topics in the two colloquiums and in the advanced seminar: most lectures in the seminar are of keen interest for both the audiences of CAAP and FASE; moreover it turned out that papers on type inference were submitted and accepted in both conferences and this has resulted in a common session. We consider this as a success of TAPSOFT in being a link between theoretically inclined and methodologically inclined research.

Nearly 150 papers were submitted to TAPSOFT'93. Like a number of major conferences this year, we have noticed a very significant increase in the number of submitted papers and in their overall quality as well. This has resulted, of course, in a strong selection of 42 papers. We thank sincerely all the program committee members, especially those who managed to attend the final meeting (marked with a * in the two lists above), and the referees listed on the next page for their care and advice (we apologize for possible omissions). Special thanks are due to Claude Marché for his help in computerizing the collection of the review forms. Besides being quite useful, it pointed out that interoperability of computer systems is still an open issue...

Thanks are also due to the organising committee of TAPSOFT: A. Finkel, M-C. Gaudel (general chair), J-P. Jouannaud, B. Rozoy, M.-F. Kalogera (AFCET). Special thanks are due to Corinne Sweeney of AFCET and to Evelyne Jorion of LRI for helping us efficiently.

TAPSOFT'93 has been supported by CNRS, the ESPRIT Basic Research Working Group COMPASS, DRET-DGA, EATCS, GI, LRI, the Ministère des Affaires Etrangères, the Ministère de la Recherche et de l'Espace, the PRC Programmation et Outils pour l'Intelligence Artificielle, Université de Paris-Sud (Division Recherche).

Orsay, February 1993

Marie-Claude Gaudel
Jean-Pierre Jouannaud

List of Reviewers

Amadio R.
Anderson Stuart
Asperti A.
Astesiano E.
Badouel E.
Baldamus Michael
Basin David
Baumeister H.
Beaudouin-Lafon Michel
Bellia Marco
Bergstra J.A.
Berlioux Pierre
Bernot Gilles
Bertossi Alan
Beyer
Billaud M.
Bockmayr A.
Borzyszkowski Andrzej
Bouajjani Ahmed
Boudet A.
Brandenburg F.J.
Cardelli Luca
Caspi Paul
Clement T.
Comon H
Compton K.
Corradini Andrea
Costa G.
Cottam Ian
Courcelle B.
Curien P L.
Dahn B.I.
De Francesco Nicoletta
De Nicola Rocco
de Frutos David
Degano P.
Dershowitz N.
de Simone Robert
de Vink Erik
de Vries Fer-Jan
Diekert V.
Drossopoulou Sophia
Durand I.
Farinas del Cerro Luis
Faulhaber Joachim
Fedou J.M.
Félix P.
Fernandez de la Vega W.
Ferrari G.
Fey Werner
Finkel Alain
Fisher M.D.
Gabarro Joaquim
Ganascia A.
Garavel Hubert
Gnesi Stefania
Getzinger Thomas
Gogolla Martin
Götzke K.
Grädel Erich
Graf Susanne
Grieskamp Wolfgang
Gruska D.
Hermann Miki
Hui Shi
Hussman Heinrich
Inverardi Paola
Jaehnichen
Kahrs Stefan
Kent S.
Kindler Andrea
Klein H.J.
Klop Jan-Willem
Kok Joost
Korff M.
Kreowski Hans-Joerg
Krieg-Brückner Bernd
Krob D.
Kucherov Gregory
Langen Anno
Lescanne Pierre
Lindenstrauss Naomi
Liskiewicz Maciej
Litovsky I.
Lugiez Denis
Longo G.
Luo Zahohui
Magee J.
Maggiolo-Schettini Andrea
Maranguet L.
Marché C.
Marre Bruno
Mirkowska Grazyna
Moggi Eugenio

Möller Bernhard
Moller Faron
Monahan B.
Moore R
Mosses Peter.
Murphy C.
Nicollin Xavier
Nieuwenhuis Robert
Nipkow Tobias
Niwinski Damian
Noll Thomas
Ohlbach H.J.
Padawitz Peter
Palamidessi C.
Parisi-Presicce Francesco
Pawlowski Wieslaw
Pena Ricardo
Pepper Peter
Phillips I.C.C
Plaisted David A.
Plesnik J.
Pothoff A.
Privara Igor
Puel L.
Ramakrishnan Iv.
Reggio Gianna
Remy J.L.
Reingold Edward
Reinhardt Klaus
Rodriguez-Artalego Mario
Rossi Francesca
Rusinowitch Michael
Rutter J.J.M.M.
Ruzicka P.
Ryan Mark
Sagiv Y.
Sannella Don
Santen
Sargeant J.
Sassone Vladimiro
Savadovsky P.
Schieferdecker I.
Schott R.
Scollo Giuseppe
Seibert S.
Seese D.
Sergot M.J.
Serpette B.
Siegel M.
Siroky A.
Snyder W
Socher-Ambrosius Rolf
Sopena E.
Soria M.
Sotteau Dominique
Stalzer Mark
Steffen Bernhard
Stickel Mark E.
Stuber Jürgen
Sturc J.
Tönne A.
Tulipani Sandro
Valkema E.
van Oostrom V.
van Raamsdonk F.
Waldmann U.
Wiedermann J.
Wilke Th.
Wills A.
Wolczko M.
Wolff Burkhart
Wolz Dietmar
Yelick K.
Zhang Hantao
Zimmermann Paul
Zlatos P.
Zucca E.

Contents

Goldilocks and the Three Specifications

John V. Guttag*

Massachusetts Institute of Technology
Cambridge, MA 02139 USA
Email: guttag@lcs.mit.edu

Abstract. A young girl enters Threads Forest. She becomes lost. She searches for enlightenment.

1 Prolog

Goldilocks' mother had often warned her not to enter Threads Forest. It was a strange and sometimes dangerous place. On the floor of the forest, where the sun never penetrated, grew toxic fungi. Dangerous beasts lurked everywhere.

There were many paths through the forest, and since they often crossed and frequently dead ended it was easy to get lost, even to starve. Furthermore from time to time old paths disappeared and new ones appeared. Wise men and women asserted that some paths would always be there, but were cryptic about which ones.

Despite her mother's repeated warnings, Goldi ventured into the forest. She soon became hopelessly lost. After giving her plight a moment's thought, she took the only rational course of action. "Help, get me out of here," she screamed.

Somewhat to her surprise, her plea was answered almost immediately. Three strangers slipped from behind three graceful deciduous conifers. "What seems to be the problem?" they asked in unison. "I can't find my way out of this stupid forest," Goldi replied. "Not to worry," said the shortest of the strangers. "You're in luck. It just so happens that we're cartographers, and we've each just completed a map of the forest."

With that, the cartographers each handed Goldi a map. Upon examining them, Goldi discovered (to her disgust) that they were all different. "Of course they are," exclaimed the cartographers. "One is too weak, one is too strong, and one is juuuuust right."

For some reason, that explanation didn't satisfy Goldi. Seeing her puzzlement, the smallest cartographer tried to explain. "All of the paths on the too

*Support for this research has been provided in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research Research under contract N00014-89-J-1988, and in part by the National Science Foundation under grant 9115797-CCR.

weak map exist at present and will continue to exist forever. However, there are many paths through the forest that don't appear on the map. The too strong map contains all of the useful paths through the forest at the present time. However, some of these paths may disappear in the future. The just right map contains all of the paths that will always be available."

Again, the explanation did not satisfy Goldi. "Why," she demanded, "do you insist on confusing me by giving me three maps?" The tallest cartographer, who was more patient than the others, made one more attempt at educating Goldi. "The too weak map is easier to follow than either of the others, and does contain some paths (albeit often longer than necessary) through the forest. The too strong map contains the shortest paths through the forest today. The problem is that you may not be able to use them tomorrow. The just right map, well that's juuuuust right."

Goldi was still puzzled. "Perhaps," she pleaded, a tear glistening in the corner of one eye, "you could give me formal specifications of all this?" "Ok," chortled the middle-sized cartographer, with a mocking glint in his eye, "you asked for it."

2 Introduction

In designing an interface, there is sometimes a tradeoff between making it easy to implement and making it easy to use. This tradeoff often centers, particularly in concurrent programs, around the amount of nondeterminism allowed. More nondeterminism leaves freedom for the implementor to choose a simpler or more efficient implementation. Less nondeterminism may support the development of simpler or more efficient client programs.

This paper presents three alternative formal specifications of part of a threads interface. The specifications presented here differ in the amount of nondeterminism allowed, and describe a hierarchy of implementations: the implementations satisfying the "too strong" specification are a strict subset of those satisfying the "just right" specification which, in turn, are a strict subset of those satisfying the "too weak" specification.

The specifications presented here are based upon work the author did in conjunction with Andrew Birrell, Jim Horning, Roy Levin and Garret Swart—all of the Digital Equipment Corporation Systems Research Center (SRC). In [1] we published what purported to be a formal specification of the threads synchronization primitives implemented as part of the Topaz operating system on the Firefly multi-processor. To the best of our knowledge, the implementation of Topaz indeed satisfied these specifications. However, the specifications were simultaneously too weak and too strong. The specifications were too weak in that they could not be used to justify some reasonable uses of the specified primitives in client programs, because the specifications permitted paths that could not actually arise in the implementation. The specifications were too strong in that they would not be satisfied by contemplated remote procedure call (RPC) implementations, because the specifications guaranteed the existence

of paths that would disappear in those implementations.

The next section presents a short overview of Larch interface specifications. The section after that presents a short overview of what threads are about, lays out the issues involved in choosing the degree and types of non-determinism to be allowed, and presents formal specifications of three interesting alternatives in that space. The note concludes with a brief discussion of how the specifications were derived and the utility of the process and the specifications.

3 Larch Interface Specifications

Larch is a family of languages for writing formal specifications of interfaces in digital systems. The basic approach is described in [4].

The Larch family of languages supports a *two-tiered*, definitional style of specification. Each specification has components written in two languages: one language that is designed for a specific programming language and another language that is independent of any programming language. We call the former kind *Larch interface languages*, and the latter the *Larch Shared Language* (LSL).

Interface languages are used to specify the interfaces between program components. Each specification provides the information needed to use an interface. Each interface language deals with what can be observed by client programs written in a particular programming language. It provides a way to write assertions about program states, and it incorporates programming-language-specific notations for features such as side effects, exception handling, iterators, and concurrency.

Larch interface languages encourage a style of programming that emphasizes the use of abstractions, and each provides a mechanism for specifying abstract types. If its programming language provides direct support for abstract types, the interface language facility is modeled on that of the programming language; if its programming language does not, the facility is designed to be compatible with other aspects of the programming language.

An interface specification can describe exported types, constants, variables, and procedures. The specification of each procedure in an interface can be studied, understood, and used without reference to the specifications of other procedures. A specification consists of a procedure header (declaring the types of its arguments, results, and any global variables it may access) followed by a body of the form:

```
requires Predicate
modifies Target list
ensures  Predicate
```

A specification places constraints on both clients and implementations of the procedure. The *requires clause* is used to state restrictions on the state, including the values of any parameters, at the time of any call. The *modifies* and *ensures clauses* place constraints on the procedure's behavior when it is called properly. When specifying sequential programs, they relate two states, the state when