Preface

The *9th Joint Conference on Declarative Programming APPIA-GULP-PRODE 2002* brings together the Italian GULP Conference on Logic Programming, the Spanish Congress on Declarative Programming, PRODE, as well as the meeting of the Portuguese Association for Artificial Intelligence, APPIA. The Spanish and the Italian Conferences took place together in 1994 in Peñíscola (Spain) and two years later in San Sebastián (Spain) the Portuguese association APPIA joined the Conference. Since then, the three associations have co-operated in subsequent editions: Grado — Italy (1997), La Coruña — Spain (1998), L'Aquila — Italy (1999), La Habana — Cuba (2000), and Évora — Portugal (2001).

This book contains the Proceedings of the *9th Joint Conference on Declarative Programming* held in Madrid, Spain in September 16–18, 2002. The technical programme for the Conference includes 20 communications. The papers in this volume are printed in their order of presentation at the Conference, with communications grouped into thematic sessions. The papers were selected from received submissions and all of them were evaluated by 3 or 4 reviewers.

In addition to the contributed papers, APPIA-GULP-PRODE 2002 featured three outstanding invited lecturers (jointly with SAS02 and LOPSTR02): John Gallagher (Bristol, UK), Thomas Ball (Microsoft Research, Reading, USA), and Thomas Reps (U. Wisconsin, USA).

We wish to express our gratitude to all the members of the Program Committee and to all the external referees who offered their expertise in the reviewing process. They are listed in the following pages. We extend our sincere gratitude to all the authors who submitted papers and to all conference participants. We gratefully acknowledge all the sponsors and institutions supporting the Conference. Moreover, we would like to specially thank the work of the Organizing Committee who made the conference possible.

September 2002

Juan José Moreno Navarro
Julio Mariño Carballo
Editors
AGP2002

Organization

APPIA-GULP-PRODE 2002 is co-located with *International Static Analysis Symposium (SAS'02)* and *Logic-based Program Synthesis and Transformation (LOPSTR'02)*, and has been organised by the LML and the CLIP groups of the Computer Science Department, Technical University of Madrid (UPM).

Program Commitee

Juan José Moreno Navarro	U.P. Madrid (chair)
José Júlio Alferes	U. Nova de Lisboa
Pedro Barahona	U. Nova de Lisboa
Stefano Bistarelli	CNR, Italy
Nicoletta Cocco	U. di Venezia
Carlos Damásio	U. Nova de Lisboa
Giorgio Delzanno	U. di Genova
Fco. José Galán Morillo	U. Sevilla
Roberta Gori	U. di Pisa
João Alexandre Leite	U. Nova de Lisboa
Francisco López Fraguas	U.C. Madrid
Julio Mariño	U.P. Madrid
Pedro Meseguer González	IIIA Barcelona
Luis Moniz Pereira	U. Nova de Lisboa
Angelo Montanari	U. di Udine
Danilo Montesi	U. di Bologna
Marisa Navarro	U. País Vasco
Fernando Orejas	U.P. Cataluña
Salvador Pinto Abreu	U. de Évora
Paulo Quaresma	U. Évora
Maria José Ramírez	U.P. Valencia
Blas Ruiz	U. Málaga
Pasquale Rullo	U. della Calabria
Fernando Silva	U. Porto
Enea Zaffanella	U. di Parma

Referees

Francisco Bueno	Stefania Costantini	María del Mar Gallardo
Manuel Clavel	Agostino Dovier	Alfredo Garro
Marco Comini	Santiago Escobar	Paola Giannini
Jesús Correas	Lígia Ferreira	Gianluigi Greco
Agostino Cortesi	Michel Ferreira	Ángel Herranz

Organizing Commitee

Sponsoring Institutions

MCYT (Spanish Ministry of Science and Technology)
The European Comission (Information Society Technologies Programme)
CoLogNET
Facultad de Informática (School of Computer Science)
Universidad Politécnica de Madrid
EAPLS

Table of Contents

Whatever Happened to Meta-Programming?

John Gallagher

University of Bristol
http://www.cs.bris.ac.uk/john/

Abstract. Meta-programming – writing programs that have other programs as data – is central to computer science. It has a fundamental place in the history of the subject, in the form of the universal Turing machine. In the future, if programming ever ceases to be the error-prone handcraft that it is today, meta-programming has to play a vital role. Producing programs that write, analyse, verify and optimize other programs, in various languages, requires a systematic understanding of meta-programming.
Effort was made to develop meta-programming approaches, including languages specifically designed for meta-programming, mainly focussing on issues of representation of syntax rather than semantics. However, it is undeniable that meta-programming today is not a very hot topic, and those meta-programming languages and techniques have fallen into disuse.
This talk will contain first a brief survey of meta-programming in declarative languages, especially logic languages. Leading from this, we ask what is the direction for meta-programming research, given its likely future importance. It will be argued (not for the first time) that the declarative languages have a special role to play in computing, as universal executable meta-languages. How is this role to be realised? What is the way to write meta-programs in logic languages with C or Java as object language?
Is the ground representation inherently inefficient? Can the semantics of all languages be effectively realised in logic and functional languages in use today? How should we represent states in imperative program execution? We discuss these and other unresolved questions about meta-programming techniques, and suggest lines for future research.

Secrets of Software Model Checking

Thomas Ball

Microsoft Research
`http://research.microsoft.com/tball/`

Abstract. The goal of the SLAM toolkit is to validate sequential C programs against temporal safety properties. The toolkit does not require program annotation (invariants are inferred) and minimizes noise (false error messages) through a process known as "counterexample-driven refinement".
We have successfully applied the SLAM toolkit to over 100 Windows device drivers, to both validate their behavior and find defects in their usage of kernel-level APIs. The SLAM toolkit currently is in transition from a research prototype to a product-grade tool that will be supported and used by the Windows development organization.
As is often the case, there is a gap between the theory and algorithms one publishes and the implementation of a tool that has been engineered to perform acceptably in practice. In this talk, I will detail some of the engineering "secrets" we applied to the basic SLAM process to improve its performance on Windows device drivers.
(Joint work with Sriram K. Rajamani, Westley Weimer and Satyaki Das)

Static Program Analysis via 3-Valued Logic

Thomas Reps

University of Wisconsin
reps@cs.wisc.edu

Abstract. Abstract interpretation serves as a powerful theoretical tool for developing and justifying program-analysis algorithms. It provides a way to establish that information extracted from a program by a program-analysis algorithm is a meaningful characterization of what can occur when the program is executed. Typically, however, it is not an easy task to obtain the appropriate abstract state-transformation functions and to show that they are correct. On the contrary, papers on program analysis often contain exhaustive (and exhausting) proofs to demonstrate that a given abstract semantics provides answers that are safe with respect to a given concrete semantics.

In contrast, the parametric framework for program analysis proposed by Sagiv, Reps, and Wilhelm [POPL '99, TOPLAS (in press)] holds out the promise of allowing different program-analysis tools to be generated automatically from high-level descriptions of what an analysis designer desires. Analyses generated in this fashion would be correct by construction. This framework has two parts:

- A language (3-valued logic) for specifying various properties that an entity may or may not possess, and how these properties are affected by the execution of the different kinds of statements in the programming language.
- A method for generating a static-analysis algorithm from such a description.

The talk will review the principles behind the paradigm of "abstract interpretation via 3-valued logic," discuss recent work to extend the approach, and summarize on-going research aimed at overcoming remaining limitations on the ability to create program-analysis algorithms fully automatically.

(Joint work with M. Sagiv, R. Wilhelm, F. DiMaio, N. Dor, T. Lev-Ami, A. Loginov, A. Mulhern, N. Rinetzky, and E. Yahav.)