Editorial

Moreno Falaschi¹ and Augusto Sampaio²

¹ Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Via Roma, 56, 50056 Siena, Italy

² Centro de Informática, Universidade Federal de Pernambuco, Recife, PE, Brazil

Formal methods based on logical theories are used widely and successfully in many areas of computer science, including software engineering. These methods are useful for a semi-automated development of programs. Program synthesis, program transformation, abstract interpretation, and other formal techniques improve both the process of software development and the quality and correctness of the programs. This special issue includes a selection of papers that are extended and revised versions of the ones presented at the 25th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2015). The LOPSTR symposia cover various topics related to logic-based program manipulation techniques and, more generally, logical theories and formal techniques for software development. LOPSTR 2015 was held in July 2015 in Siena, Italy, co-located with the 17th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP 2015). The LOPSTR 2015 post-proceedings have been published as Volume 9527 in Springer's Lecture Notes in Computer Science (LNCS). A total of eight of the best papers that had been presented at LOPSTR 2015 were selected and invited to submit significantly extended and revised versions for inclusion in this special issue. These papers focus on various aspects of logic-based program development.

Proof Checking and Logic Programming by Dale Miller shows that formal methods and formal proof can provide some basis for trusting software. Proof checking can help to reduce the size of the trusted base by trusting a proof checker rather than an entire theorem prover. The author shows that while there is considerable overlap between logic programming and proof checking (e.g., both benefit from unification, backtracking search, efficient term structures, etc), the discipline of logic programming has, in fact, played a minor role in the history of proof checking. The author shows that logic programming can have a major role in the future of this important topic.

On Dual Programs in Co-logic Programming and the Horn μ -Calculus by Hirohisa Seki studies the proof principle of co-induction, the dual of induction, in logic programming. This principle is useful for defining infinite data such as infinite streams and infinite trees. The author considers some extensions of co-logic programming, and studies its relationship with the Horn μ -calculus. Then the author shows that the Horn μ -calculus can be used as an extension of co-logic programming for handling “non-stratified” co-logic programs. The author shows a new iterated fixpoint characterization of the well-founded semantics and Answer Set Programming by using dual programs.

Equational Formulas and Pattern Operations in Initial Order-Sorted Algebras by José Meseguer and Stephen Skeirik presents a signature transformation for patterns which allows for a systematic reduction of the first-order logic properties of an initial order-sorted algebra supporting subtype-polymorphic functions to equivalent properties of an initial many-sorted (i.e., simply typed) algebra. This yields a new simple proof of the known decidability of the first-order theory of an initial order-sorted algebra. Pattern operations have applications to, e.g. machine learning, negation in logic programming, inductive theorem proving, and automated model building.

Operational Semantics of Resolution and Productivity in Horn Clause Logic by Peng Fu and Ekaterina Komentanskaya presents a study of operational and type-theoretic properties of different resolution strategies in Horn clause logic. The authors distinguish four different kinds of resolution: resolution by unification (SLD-resolution), resolution by term-matching, the recently introduced structural resolution, and partial (or lazy) resolution. These different kinds of resolution are expressed uniformly as abstract reduction systems, which allows the authors to undertake a thorough comparative analysis of their properties.

Transforming Boolean Equalities into Constraints by Sergio Antoy and Michael Hanus considers the problem that functional and logic languages use equality with a different operational meaning. The authors show that this distinction can be avoided in Curry (a functional logic language) by providing an analysis and transformation method that automatically selects the appropriate operation. This way the language design can be simplified and the execution of programs can be optimized.

Assumption Propagation through Annotated Programs by Dipak L. Chaudhari and Om Damani focuses on the correct-by-construction programming methodology. Programs are incrementally derived from their formal specifications, by repeatedly applying transformations to partially derived programs. Users may make assumptions whenever needed to proceed further. The authors present rules for propagating assumptions through annotated programs. They show how these rules can be integrated in a top-down derivation methodology to provide a systematic approach, materializing the assumptions with executable statements at a place different from the place of introduction, and strengthening loop invariants with minimal additional proof efforts.

A program analysis framework for tcp based on abstract interpretation by Marco Comini, María del Mar Gallardo, Laura Titolo and Alicia Villanueva considers a previously defined semantic framework and abstract diagnosis method for the Timed Concurrent Constraint Language, a timed extension of the concurrent constraint paradigm. The authors propose an abstract semantics that, together with a widening operator, is suitable for the definition of different analyses for tcp programs. The widening operator allows to guarantee the convergence of the abstract fixpoint computation.

Optimizing Sorting Algorithms by Using Sorting Networks by Michael Codish, Luís Cruz-Filipe, Markus Nebel and Peter Schneider-Kamp shows how the theory of sorting networks can be applied to synthesize optimized general-purpose sorting libraries. Significant speed-ups are obtained when taking advantage of instruction level parallelism and non-branching conditional assignment instructions. Experiments show that using code synthesized from efficient sorting networks as the base case for Quicksort libraries results in notable real-world speed-ups.

We are very grateful to Formal Aspects of Computing for publishing this special issue, and in particular to the managing editor John Cooke for his continuous support and advice. We would like to thank all the authors of the selected papers. Our deep gratitude also goes to all the reviewers who did an excellent work. Their constructive suggestions have contributed significantly to improve the quality of the articles.

Moreno Falaschi
Augusto Sampaio