GUEST EDITOR'S INTRODUCTION

PASCAL VAN HENTENRYCK

The design of programming languages is an art involving difficult trade-offs between expressiveness, efficiency, and semantic simplicity. The developments of logic programming and Prolog are typical examples of this tension. Logic programming, by sacrificing the full expressiveness of predicate calculus, opens the door to efficient implementations of programming languages based on logic. Prolog, by imposing further restrictions on logic programming such as a depth-first search strategy and a left-to-right computation rule, achieves a compromise between expressiveness and efficiency that is particularly appropriate for a variety of applications such as natural language and symbolic processing. The shortcomings of Prolog have often been stressed in the past, yet few other logic programming languages have achieved so much popularity and it is still hard to come up with better incarnations of the logic programming ideal. Constraint logic programming (CLP) is probably one of the most promising attempts in this direction.

CLP is based on the idea of replacing unification by constraint solving as the kernel operation of logic programming languages. CLP is a scheme that can be instantiated to various constraint solvers depending on the class of applications targeted. For instance, languages like Prolog III, CHIP, and CLP($\mathcal{R}$) embed constraint systems over real or rational numbers, Booleans, and finite sets of integers. The idea underlying CLP is amazingly simple retrospectively, yet it is far-reaching for a number of reasons that we can only outline in this introduction. From a semantic standpoint, CLP generalizes and simplifies the theory of logic programming because constraints are conceptually simpler than unifiers. From a programmer standpoint, CLP amplifies the traditional advantages of logic programming without sacrificing its strengths. In particular, CLP enhances our ability to work with partial information (generalizing the logic variable), increases the multidirectionality of programs, and improves the expressiveness and declarative nature of the language. Finally, CLP opens new horizons and application areas for logic programming. For instance, the foregoing CLP languages have been applied to numerous problems in operations research, design, biology, decision-support systems, and artificial intelligence, to name a few. They are now used in industry to solve practical problems that were previously considered outside the application domain of logic programming. Perhaps the nicest feature of CLP is its balance between semantic simplicity and practicality that is at the heart of logic programming since its inception.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$6.00

Research on CLP started quietly in a small number of research groups and has become an active area of research and development. The impact of CLP has also crossed the borders of our community and led to new directions in other areas of computer science such as programming languages, databases, and artificial intelligence. Despite these early developments, many issues are still left open both in the foundation, design and implementation of CLP languages, and their practical applications. The purpose of this special issue is to review some of the current developments in this area with the hope of stimulating further interest in this new direction. It contains six papers covering areas such as semantics, constraint systems, extensions of CLP, and constraint databases.

CLP, as an abstraction of logic programming, provides a rich setting to study the semantics of programming languages. This is illustrated by the first paper by Ait-Kaci and Podelski in their quest for a meaning to LIFE. They show that a subset of LIFE can be viewed as an instance of the CLP scheme. The result is particularly interesting because of the absence of element-denoting terms in the constraint system of LIFE. The paper gives a comprehensive account of this approach, including type-theoretic, logical, and algebraic characterizations.

One of the key issues in the design of CLP languages is the choice of a constraint system. An ideal CLP language should include an expressive constraint system endowed with a complete, efficient, and incremental decision procedure. In practice, CLP languages achieve a trade-off between efficiency, expressiveness, and completeness as illustrated by the second and third papers in this issue. The paper by Imbert, Cohen, and Weeger considers linear constraints over real numbers as a constraint system. They propose a complete decision procedure for this constraint system based on the simplex algorithm. The key features of their algorithm are the use of a lexicographic normal form and the classification of constraints into subclasses. The paper by Lee and van Emden also considers real numbers, but takes a very different approach: They use interval arithmetics to prune the domains of the variables. As a consequence, they do not restrict themselves to linear constraints, but give up completeness. The paper makes the interesting point that this approach provides a logical framework for computing with floating-point numbers, showing that interval computation can be seen as deduction.

The next two papers are concerned with extensions to the CLP scheme. Much research has been devoted in recent years to enhance the descriptive and operational expressiveness of CLP languages. The paper by Wilson and Borning is motivated by the many applications in which some of the constraints express preferences rather than requirements. Their new scheme HCLP (hierarchical constraint logic programming) accommodates both preferential and required constraints using the idea of constraint hierarchies. Their paper gives a comprehensive account of HCLP from the theoretical foundations to the applications. The paper by Le Provost and Wallace introduces the idea of generalized propagation to reconcile the constraint propagation approach of finite domains in CHIP and complete constraint solvers. The key idea behind generalized propagation amounts to approximating all answers to a goal by a constraint, which can then be added to the constraint store, and to coroutining its execution with the rest of the resolvent. The main contribution of generalized propagation is to increase the operational expressiveness of CLP languages.

In the last paper, Srivastava and Ramakrishnan consider the bottom-up evaluation of CLP programs over linear real constraints. Their key idea is to use

constraints to speed up bottom-up evaluation. They also integrate this approach with magic templates. This last paper lies on the frontier of logic programming, databases, and constraints, an area that has received much attention in recent years.

This special issue would never have been completed without the dedication of many individuals. First, I would like to thank M. Bruynooghe for initiating the issue, giving me instant feedback whenever necessary, and providing me with wise advice when the time came for decisions. I am also grateful to the many authors who responded to the call for papers and submitted high-quality papers, only a fraction of which could be accommodated. Every paper was sent to three reviewers, two specialists and an outsider to try to increase the accessibility of the special issue. I would like to express my deepest gratitude to the 58 reviewers who took part in this process and provided me with detailed comments on the papers. Finally, I would like to thank A. Herman for her help and patience during this long process.

Pascal Van Hentenryck