Preface

Historical Background

Logic programming was founded some 25 years ago. Its birth is usually associated with the seminal paper of Robert Kowalski [5] in which a computation mechanism was proposed that made it possible to use logical formulas as programs. About the same time this idea was realized by Alain Colmerauer and his team in a programming language, Prolog.

The creation of logic programming is the outcome of a long history that for most of its course ran within logic and only later inside computer science. For a proper historical perspective to this volume let us briefly retrace this history.

Logic programming is based on the syntax of first-order logic, which was originally proposed in the second half of the 19th century by Gottlob Frege and later modified to its current form by Giuseppe Peano and Bertrand Russell.

In the 1930s Kurt Gödel and Jacques Herbrand studied the notion of computatibility based on derivations. These works can be viewed as the origin of the "computation as deduction" paradigm. Additionally, Herbrand disscussed in his PhD thesis a set of rules for manipulating algebraic equations on terms that can be viewed now as a sketch of a unification algorithm.

Another thirty years passed before Alan Robinson published, in 1965, his seminal paper that lies at the foundations of the field of automated deduction. In this paper he introduced the resolution principle, the notion of unification and a unification algorithm.

Using the resolution method, one can prove theorems of first-order logic, but another step was needed to see how one could compute within this framework. In 1971 Kowalski and Kuehner introduced a limited form of resolution, called linear resolution. This finally formed the basis for Kowalski's subsequent proposal of what we now call SLD-resolution. This form of resolution is more restricted than the one proposed by Robinson in the sense that only clauses with a limited syntax are allowed. However, this restriction now has a side effect in that it produces a satisfying substitution, which can be viewed as the result of a computation.

A number of other ideas to realize the computation-as-deduction paradigm were proposed around the same time, notably by Cordell Green and Carl He-

witt, but Kowalski's proposal, probably because of its simplicity, elegance and versatility, became most succesful. In particular, the crucial insights of Alain Colmerauer on the programming side and David H. D. Warren on the implementation side made it possible to turn this approach to computing into a realistic approach to programming.

By now Prolog is standardized — see [3] and new books on Prolog and logic programming keep appearing, for instance [1] and [2].

Why This Volume

Logic programming is an unusual area of Computer Science in that it cuts across many fields that are themselves autonomous Computer Science areas. More specifically, chapters on it or its uses can be found in standard textbooks on programming languages such as [7] (see Chapter 8 "Logic Programming"), database systems such as [9] (see Chapter 3 "Logic as Data Model" that describes the datalog), compiler writing (see Chapter 4 "Compilation of Logic Programming Languages" in [10]), artificial intelligence (see, e.g., [8] in which all algorithms are implemented in Prolog), natural language processing (by employing Prolog, as in [4]), and more recently machine learning (see Chapter 10 "Learning Sets of Rules" in [6] that deals with inductive logic programming).

This richness of the logic programming paradigm can be attributed to the remarkable simplicity and conciseness of its syntax that, in its basic form, consists of Horn clause logic. The major discovery of Kowalski and Colmerauer 25 years ago was that this syntax is sufficient for computing. One of the first application areas of logic programming was natural language processing. Time has shown that this formalism can also be profitably used for a number of other purposes, for instance, knowledge representation, parallel computing, and machine learning. Further, various simple extensions made this formalism applicable for such diverse uses as database systems, formalization of commonsense reasoning, and constraint programming.

At the same time this omnipresence of logic programming is a sign of its weakness. In most of these areas logic programming has found a niche but did not become the main technology. In particular, in the case of software engineering the world seems to be ruled by the imperative programming paradigm, and the declarative programming paradigm embodied by logic programming and functional programming has not gained enough ground to be widely recognized by the industry.

On the other hand the number of industrial applications developed using the logic programming technology is steadily growing and is much larger than most of us realize. Originally, these applications have been developed using mainly Prolog. In more recent applications also constraint logic programming and inductive logic programming systems have been used.

These and related considerations about the current role of logic programming were behind the organization of a meeting in Shakertown, Kentucky, USA in April 1998.

Our idea was to review the state of the art in logic programming, to assess the situation in this field, and to clarify what progress has been made in it in recent years. We invited the leading researchers in all subareas of logic programming and asked them to review the field and to present promising future directions of research. In addition, we asked for input from colleagues from neighboring fields in order to put the situation within the logic programming field in an appropriate context. This led to the present volume that provides a unique perspective of this field, 25 years after its creation, a perspective that is broad in scope and rich in suggestions.

In fact, the articles here presented cover most of the areas of logic programming. Their emphasis is on the assessment of achievements in this field and on promising future directions.

Contributions

Contributions included in this book are organized according to their subject areas. The reader will find that, independently of their subjects, the papers can be classified into three main types. The first kind provides an assessment of a specific subfield of logic programming. The articles in this group deal with

- natural language processing, by Veronica Dahl,
- planning, by Vladimir Lifschitz,
- inductive logic programming, by Luc de Raedt,
- programming methodology, by Danny de Schreye and Marc Denecker, and
- concurrent logic programming, by Kazunori Ueda.

The contributions of the second kind offer some new lines of research by reassessing known ideas and concepts of logic programming and by shedding new light on their use. The articles in this group are by

- Krzysztof Apt and Marc Bezem, on an alternative approach to declarative programming,
- Howard Blair, Fred Dushin, David Jakel, Angel Rivera and Metin Sezgin, on relating logic programming to continuous mathematics,
- Marco Bozzano, Giorgio Delzanno, Maurizio Martelli, Viviana Mascardi and Floriano Zini, on the use of logic programming for rapid prototyping of multi-agent systems,
- Koichi Furukawa, on the use of inverse entailment in inductive logic programming,

- Manuel Hermenegildo, Germán Puebla and Francisco Bueno, on the use of abstract interpretations for program development,
- Gopal Gupta, on use of logic programming for program semantics and compilation,
- Michael Maher, on the addition of constraints to logical formalisms,
- Victor Marek and Mirek Truszczyński, on an alternative approach to logic programming via the use of stable models, and
- Carlo Zaniolo and Haixun Wang, on logic-based foundations for advanced database applications, such as data mining.

Finally, in the third type of contributions, the ideas originally conceived within logic programming are applied to areas that at first glance have nothing to do with it. These are contributions by

- Paul Tarau, on mobile agent programming,
- Jacques Cohen, on computational molecular biology, and
- Maarten van Emden, on numerical computing.

Additionally,

- Saumya Debray studies program optimization within a multi-language environment,

and, in a contribution from a neighbouring field,

- Philip Wadler provides some insights into the use of functional programming in industry.

We would like to take this opportunity to thank all the authors of the submitted papers for having agreed to contribute to this special issue and for their help in putting this volume together, and the referees for giving of their time and providing helpful reviews of the papers.

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, London, 1997.
2. W. F. Clocksin. *Clause and Effect.* Springer-Verlag, Berlin, 1997.
3. P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog: The Standard.* Springer-Verlag, Berlin, 1996.
4. G. Gazdar and C. Mellish. *Natural Language Processing in PROLOG.* Addison-Wesley, 1989.
5. R.A. Kowalski. Predicate logic as a programming language. In *Proceedings IFIP'74*, pages 569–574. North-Holland, 1974.
6. T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
7. R. Sethi. *Programming Languages: Concepts and Constructs.* Addison-Wesley, 1989.

8. Y. Shoham. *Artificial Intelligence Techniques in Prolog.* Morgan Kaufmann, San Francisco, CA, 1994.
9. J.D. Ullman. *Principles of Database and Knowledge-base Systems, Volume I.* Principles of Computer Science Series. Computer Science Press, 1988.
10. R. Wilhelm and D. Maurer. *Compiler Design.* Addison-Wesley, 1995.

Amsterdam — *Krzysztof R. Apt*
Lexington, KY — *Victor W. Marek*
Lexington, KY — *Mirosław Truszczyński*
Stony Brook, NY — *David S. Warren*