The Knowledge Engineering Review, Vol. 10:1, 1995, 97–100

Report on the Workshop: Applications of Logic Programming in Software Engineering

P. CIANCARINI[1] and LEON STERLING[2]

[1]*University of Bologna, Italy*
[2]*Case Western Reserve University, USA*

1 Introduction

The Workshop on Applications of Logic Programming in Software Engineering was held at S. Margherita Ligure, Italy, on June 18 1994. This workshop was organized in conjunction with the International Conference of Logic Programming. We recognize that over the past decade, several sporadic research efforts have addressed the use of logic programming to improve the process of software development and the quality of its products. The workshop, we believed, would give an assessment on how effective logic programming has been and could be.

2 Background

One of the hopes of the pioneers of logic programming was to improve the quality of software artifacts by using logic to build programs directly, instead of using some arbitrary language, i.e. non-logic. An impediment has been the perceived inefficiency of logic programs relative to imperative programs caused by the use of inference as the main computation mechanism used by logic languages instead of sequential flow. Several effects have been devoted in the past and are currently being devoted to the design of new mechanisms to control inference improving the execution speed. One of the most notable projects was the Japanese Fifth Generation project, centred upon designing and building new hardware able to execute logic as a machine language. The end of the Fifth Generation project marks a crucial point in the history of logic programming. The evaluation of the results of the project has generated a debate centred around the effective applicability of such a technology in industrial contexts.

A relevant feature of the debate is the influence of logic programming on software engineering research and practice. In an era of increasing software costs, software engineers need to investigate new methods and techniques, to reduce the costs and the complexities of the software lifecycle, and to face and solve the problems presented by new software systems. Logic languages have proven adequate for "declarative programming", deductive database implementation, expert systems development, AI applications, and more generally as general-purpose tools for exploratory and nondeterministic programming. Being a logic-based language, Prolog can provide a mathematically based framework for symbolic evaluation and automatic deduction.

From a software engineer's perspective, logic languages, especially Prolog, have been advertised as tools useful for executable specification and/or rapid prototyping, often confusing the two activities. Also, in a number of projects logic programming environments have been successfully used to build rule-based tools for software engineers, e.g. expert systems for helping with maintenance of software systems, or generation of test suites.

3 Main themes

The papers accepted for presentation at the workshop were collected into two sessions: a session on the use of the logic programming languages as formal methods for specifying software systems, and

a session on the use of logic programming technology in the field of software process modelling. Other worthy papers were not accepted due to not fitting these themes, and are briefly discussed below.

3.1 Logic programming as a formal method

Formal methods have been suggested to improve the quality of software products and their development processes. A formal method is typically based on a formal specification language that is used to state in a formal document both the user requirements, and a number of important properties that should be valid for the required system. The document is used as reference for the subsequent software development process. Ideally, all other software documents should be formally derivable from the initial formal specification, or at least it should be possible to reason about the documents to prove their adherence to the requirements specification. Hence, a formal specification language should exhibit a clear formal semantics and a programming logic able to support reasoning about specifications. First order logic is widely accepted as a general-purpose reasoning tool for specification formalisms. Prolog is a special form of first-order logic that can be used as an executable specification language. In short, Prolog can be easily used for rapid prototyping of specifications, that run as reasonably efficient programs. In general, a logic language is a good specification language because it is a very high-level "declarative" language and, moreover, it is executable, easy to use and expressive. Being based on logic, it allows both writing high-level specifications and reasoning about them. However, modern Prolog programming systems have lost part of the declarative power of logic, sacrificing it to efficiency. Large effects are devoted today to define new logic languages that improve on the clarity and expressiveness of Prolog's semantics, while retaining an acceptable performance when executed.

This session included the following papers:

- "Design of redundant formal specifications by logic programming: Merging formal text and good comments", by Sophie Renault and Pierre Deransart (INRIA Rocquencourt, France). This paper suggests the use of "formal comments" inside normal logic programs: a specification is defined as the union of both the logic program and the comments that state properties of the program itself. Such properties concern completeness and correctness of the specification itself, and can be automatically checked.
- "An intelligent LOTOS interpreter in AMLOG", by A. Togashi, G. Mansfield and N. Shiratori (Tohoku Univ., Japan). AMLOG is a logic and functional language: here it is used to build a LOTOS interpreter. LOTOS is a standard specification language well known in the field of protocol design and validation. The authors show how the equational logic that is at the basis of AMLOG is useful to refine, execute and reason with LOTOS specifications.
- "Transformational development of logic programs from executable specifications", by M. Fromherz and N. Fuchs (XEROX PARC, USA and Univ. of Zurich, Switzerland). The authors advocate the use of Explore/L, an object-oriented extension of Prolog, for writing executable specifications that are modular and reusable.
- "Specifying industrial real-time systems in a logical framework," by E. Ciapessoni, E. Corsetti, M. Migliorati and E. Ratto (CISE, Italy). This work describes a real application of TRIO, a logic language based on temporal logic to be able to reason on real-time systems. The application concerns automation of control systems for electricity production, transport and distribution.
- "Executable requirements specifications in a logic specification language SPILL-2", by F. Kluzniak and M. Milkowska (Univ. of Warsaw, Poland). SPILL-2 is the second version of an executable typed logic language that is an extension of the language Gödel. A SPILL-2 program consists of type declarations, predicates defined by rules, and a number of queries that are used to test the specification.
- "Z Specifications: Syntactic sugar for Prolog?", by L. Sterling and T. Turnidge (Case Western Univ., USA). Z is a well known non-executable specification language based on the set theory

and first order logic. Sterling and Turnidge illustrate a novel approach to the animation of Z specifications, a topic that has been studied by several researchers.

3.2 Use of logic programming for software process modelling

The second major theme discussed at the workshop was the use of logic programming for software process modelling. Software process modelling is the activity of formalizing the production lifecycle of software systems. The aim of a software process model is to formally describe a software development process, that then is effectively used and possibly enacted by a software engineering environment. Logic programming is an interesting technology for software process modelling, for several reasons. First, several routine development activities in the software process can be defined by rules which are immediate to express within logic programming. In fact, a few initial proposals in the field aimed at animating a software process using a Prolog program with some embedded development rules. Second, a modern programming environment for a logic language can be easily used as a project database, usually implementing a number of special primitives to explicitly support process programs. Last, and by no means least, there is a trend toward more complex programming environments, called process-centred development environments, that should be able to "enact" software processes, i.e. to constrain and drive the activity of a group of programmers. Some parallel logic languages are being used to build this kind of environment.

The papers presented in this session were:

- "Applications of logic programming in software process modelling", by P. Ciancarini (Univ. of Bologna, Italy). Ciancarini presented a survey of software process modelling languages based on rules. It was intended as an introduction to the systems discussed in the succeeding papers, and to the context and problems they attack.
- "SPELL: A logic programming language for process modelling", by M. Nguyen and R. Conradi (Norwegian Institute of Technology, Norway). SPELL is the process modelling language of the EPOS system. EPOS is a software engineering environment offering integrated software and process configuration management. EPOS is centred upon a project database whose user interface has been programmed in SPELL.
- "A Prolog-based semantics of a dedicated process modelling language", by G. Junkermann (Univ. of Dortmund, Germany). Prolog is used to animate a graphical process modelling language, based on entity-relationship diagrams and on Statecharts, used in the Merlin process centred software development environment.
- "Pandora: a temporal logic based process engine", by P. Lago and G. Malnati (Politecnico di Torino, Italy). A combination of logic programming and temporal logic of traces is used to reason about software processes. Pandora is a logic language to write rule-based event-driven process programs, which drive the evolution of the software process.

3.3 Other applications

The papers submitted to the workshop which were accepted for the proceedings but not for presentation demonstrate a diversity of areas within software engineering to which logic programming has been applied. Of particular note are "Reverse engineering of COBOL programs into Prolog programs", by U. Geske and M. Nitsche (GMD, Germany), "SOFTM: A software maintenance expert system in Prolog", by L. Pau and J. Kristinsson (DEC, France), and "Behaviour Prolog: A Prolog based extensible programming language", by G. Sergievski (Moscow Institute of Physics Engineering, Russia).

4 Conclusions

Logic programming is a very useful technology for rapid prototyping and exploratory programming. This has been confirmed by researchers in software process modelling who have successfully

used logic environments to implement novel ideas and conduct experiments. However, its use in industrial contexts as the kernel technology of distributed software engineering environments remains untested and the technology is still somewhat immature.

The field of formal methods is currently the subject of much study. Here logic programming seems to be a strong candidate to offer a complete formal method. The amalgamation of computation and theorem proving seems a plus for formal methods advocates.

The workshop proceedings have been published as Technical Report CES-94-18 at Case Western Reserve University. Most papers are also available via ftp from the site ftp.cs.unibo.it, directory pub/WLPSE. Moreover, a selection of papers presented at the workshop will be published in a special issue of the *International Journal of Software Engineering and Knowledge Engineering*.