TLP **12** (1–2): 1–4, 2012. © Cambridge University Press 2011
doi:10.1017/S1471068411000524 First published online 12 September 2011

Introduction to the special issue on Prolog systems

BART DEMOEN
Department of Computer Science, K.U.Leuven, B-3001 Leuven, Belgium
(*e-mail:* Bart.Demoen@cs.kuleuven.be)

MARIA GARCIA DE LA BANDA
Faculty of Information Technology, Monash University, Melbourne, VIC 3145, Australia
(*e-mail:* Maria.GarciadelaBanda@monash.edu)

It has now been 40 years since the birth of the Prolog language[1] and of its first implementation by A. Colmerauer and P. Roussel. Since then, a large number of Prolog systems have been implemented. While the *core* of the Prolog language has not changed much in these 40 years, Prolog systems have undergone an extraordinary evolution that stems from two main sources. One is the trend to extend Prolog to incorporate ideas from other language paradigms that have proved useful in real-world applications. This includes concurrency, parallelism, higher order predicates, object-oriented programming, Web interfaces, processing of large amounts of data, and flexible developer tools that enhance reliability and robustness through assertions. A second source of change is the exploration of ideas for which Prolog systems are uniquely suitable and that have led to the creation of new programming paradigms. This includes tabling, constraint logic programming, answer set programming, and probabilistic logic programming.

This evolution and the significant effort of separate groups around the world has led to a rich variety of Prolog systems, only partly held together by the ISO/IEC 13211-1 standard. While the roots of most current Prolog systems lie in the 1980s, a period that is well accounted for in the survey by P. Van Roy,[2] there has been a very significant amount of change since then. More than 15 years after that survey, we believed it was time for a new look at the current Prolog systems, one that lets the systems' authors have their say.

To achieve this, in April 2009, we invited a selected number of Prolog systems to contribute to this special issue based on the following criteria: the system must be actively maintained, freely available, clearly visible in the academic community and scientific literature, open source, have a real user base, a vision on the future, and be used in real applications. While not every invited system conforms to all criteria, all invited systems conform to most. The submitted papers were subject to a very

[1] Alain Colmerauer, "The Birth of Prolog" III, CACM, Vol. 33, No. 7, 1993.
[2] Van Roy Peter, "1983–1993 – The wonder years of sequential Prolog implementation" Journal of Logic Programming, vol. 19–20, pp. 385–441, 1994.

https://doi.org/10.1017/S1471068411000524 Published online by Cambridge University Press

thorough peer-review process that included both anonymous and non-anonymous phases.

The resulting issue can be seen as a catalogue of the Prolog systems currently available that emphasizes their commonalities as well as their differences, and offers insight not only into the current state of the art, but also into the future trends for each system.

Lacking any better and easily applicable criterion, the order in which the Prolog systems appear in this special issue is by the date of their first release under the current name. The resulting order is quite surprising since some systems have evolved from a predecessor with a different name, or were popular among academics long before their actual public release. Let us briefly introduce the selected systems in the chosen order.

YAP (Yet Another Prolog) was released in 1986 with Luís Damas as the main designer and implementor. *YAP*'s aim was to deliver fast compilation and execution, something that was achieved through the implementation of a compiler in C and the use of innovative emulation techniques. These two characteristics earned *YAP* a significant user base. Since then, *YAP* has become particularly well known for its dynamic indexing mechanism (which allows it to deal efficiently with large amount of data) and for its integration of or-parallelism and tabling in a single engine. Today, the developers look forward to achieving faster execution through parallelism and native execution, better memory scalability, and to continue working with other Prolog systems to increase the amount of code shared.

SICStus Prolog was released in 1987, with Mats Carlsson as its main developer, and it quickly gained appreciation in academic circles due to its speed and robustness. Its commercialization (by SICS) provided *SICStus Prolog* with the resources needed to add many components – such as robust constraint solvers and comprehensive library modules – that continued to attract academics and were crucial for its success in commercial applications. *SICStus Prolog* is well known also for two efforts it introduced at great cost and suspended later: native code generation and or-parallelism. Today, SICStus Prolog is being used in teaching, research, and production, running on everything from servers to handheld devices.

SWI-Prolog was also released in 1987 and is the achievement of mainly one person: Jan Wielemaker. It was first developed as an in-house Prolog system to support the implementation of software prototypes in other research areas. Thus, its focus was on integrating technology that could improve this supporting role such as, for example, achieve fast interaction with external resources and reductions in memory usage or start-up times. The resulting system has obtained a considerable user and developer base both in the academic and commercial environments. Today, the developers' aim is to convince a wider IT community that Logic Programming is a viable paradigm for developing and deploying applications. Critical to this is cooperation with developers and users of all Prolog systems to build a community with large-scale code sharing.

BinProlog was first released by Paul Tarau in 1991. *BinProlog* was designed for exploring different language design and implementation decisions. In this regard, *BinProlog* can pride itself on a minimalistic approach to design and implementation

https://doi.org/10.1017/S1471068411000524 Published online by Cambridge University Press

that led to a simplified virtual machine based on binarization, the use of a single language constructor (Interactors) to specify (and even implement) built-ins and language extensions, and a large number of versions written eventually in Java (and called Jinni Prolog). Today, a new implementation (Lean Prolog) is on the way, combining the simplifications derived from binarization and first-class engines. As an unusual feature, it switches as needed between a C-based emulator for fast execution and a Java-based emulator, running the same byte-code, for flexibility.

ECL^iPS^e was released in 1992 by ECRC as an integration of constraints, parallelism, and database functionality into the code base of its forerunner system SEPIA. While the team was large, the main architect of the early days was Micha Meier, and much of his code survives even today. Later, development of ECL^iPS^e refocused squarely on the integration of diverse constraint solving techniques (such as mathematical programming and local search) and on the provision of features required to program in the large. This has made ECL^iPS^e a popular choice for industrial-scale applications that have constraint problems at their core, such as those found in airline and telecom companies. Today ECL^iPS^e's open-source development is focused at furthering its strengths as a practical problem solving platform. This includes new libraries, interfaces, tools, and sometimes language features. Increased compatibility with other Prolog systems is another goal.

XSB was first released in 1993, with David S. Warren as its principal designer and developer. *XSB* extends the application areas for Prolog by adding tabling to the Prolog computation mechanism (SLD-resolution): a table of subgoals and their associated answers is maintained and, any time a subgoal already in the table is called, its computed answers are re-used rather than re-computed. This technique has major theoretical and practical advantages, and is now part of many Prolog systems. Today, XSB continues to break new ground by using tabling to implement extensions of Prolog into non-monotonic logics and logics of uncertainty, extensions that form the basis of exciting research and commercial applications. Current research work includes implementation of probabilistic logics, extension of termination properties of tabling, and support for deep reasoning employing ontologies and other external sources.

B-Prolog had its first release in 1994 with Neng-Fa Zhou as its sole implementor. While *B-Prolog* started out as just another WAM implementation, it has evolved in several steps to the current TOAM Jr. with Jumbo instructions, tabling, and fast finite-domain constraint solvers. Particular to *B-Prolog* is its implementation of delayed goals as frames on the execution stack, and its argument passing that differs from the WAM. Besides being used in a number of commercial applications, it also forms the basis for PRISM, a popular logic-based probabilistic reasoning and learning system. Today, B-Prolog is being extended to meet the demands of real-world applications. This includes the capability of exploiting parallelism for constraint solving and optimizing the tabling system for ever large-scale machine-learning applications.

Ciao was first released as a stand-alone system by Manuel V. Hermenegildo's team in 1997, even though some of its main components had already been implemented in &-Prolog, a parallel Prolog system that started as a branch of SICStus Prolog in 1987.

https://doi.org/10.1017/S1471068411000524 Published online by Cambridge University Press

Ciao stands out for two reasons: its assertion language, which allows its preprocessor to statically analyse, document, verify, and optimize programs producing correct and efficient code, and its language extension packages, which allow users and system implementors alike to easily extend the syntax and semantics of *Ciao*'s basic kernel language. Today, the Ciao team aims at extending the expressive power of the language with, amongst other things, a new object model and improved concurrency and parallelism support. Another direction of development consists in improving the power and scalability of the preprocessor to promote its routine use in everyday programming, even in the context of other programming paradigms.

GNU-Prolog was released by Daniel Diaz and Philippe Codognet in 1999 and had its roots in wamcc. Its main distinguishing feature is its sophisticated low-level compilation technology, which results in a native code compiler that achieves good performance without becoming too complex. This allows *GNU-Prolog* to be used as a research platform for many language extensions. Other significant features are its "RISC approach" integration of finite-domain constraint solving, its ISO Core 1 fully compliant implementation, and its team's enthusiasm in promoting Prolog to the GNU community. Today, GNU Prolog development is aimed primarily at a more efficient, portable, and manageable back-end, possibly based on a typed assembly language such as LLVM. The future lies in broadening the developer base, combined with more code and feature sharing with the other implementations of Prolog.

Five of the systems introduced above (*YAP*, *SWI-Prolog*, *XSB*, *Ciao*, and *GNU-Prolog*) have recently combined their efforts in an attempt to converge again. The main aims are to increase portability between these systems and to reduce the overall implementation effort needed to maintain and further develop them. Thus, the cooperation is focused at the level of libraries, interfaces with other languages, and packages like threads. We applaud this collaboration and wish that it keeps its momentum, since we believe that such a combined effort will make Prolog much stronger.

In closing this preface, the editors would like to acknowledge the help provided by the external referees (listed at the end) and Annalisa Bossi who, as editor of TPLP when this special issue was being first conceived, gave us the opportunity to realize it.

https://doi.org/10.1017/S1471068411000524 Published online by Cambridge University Press