B-Prolog: An Overview

www.bprolog.com

B-Prolog is a high-performance, high-quality, and award-wining implementation of the standard Prolog language with several useful extended features including action rules for event handling, finite-domain constraint solving, and tabling. First released in 1994, B-Prolog has become one of the most innovative and widely used CLP systems. Whether it comes to standard Prolog programs, or finite-domain constraint programs, or tabled programs, B-Prolog is proud to be a front runner. The constraint solver of B-Prolog was ranked top in two important categories in the Second International Solvers Competition[1]. B-Prolog is being used by numerous application projects ranging from (natural) language processing, business logic, CAD, data analysis, machine learning, to optimizations systems. B-Prolog underpins the PRISM system[2], a popular logic-based probabilistic reasoning and learning system.

A Fast Prolog Engine

The latest version of B-Prolog, version 7.0, adopts a new virtual machine, named TOAM Jr. [3], which is a successor of TOAM [1]. TOAM Jr., unlike the Warren Abstract Machine, is a pure stack machine that is suited for software emulation. The new design boosts the speed by over 50%, thanks mainly to reduction of the cost of parameter passing and bytecode interpretation, making B-Prolog one of the fastest Prolog engines.

As the name TOAM (short for Tree-Oriented Abstract Machine) entails, indexing has been a priority in B-Prolog since the very first version. For programs that consist of matching clauses in which input/output unifications are separated and programs with mode information, the compiler generates matching trees that index the clauses on all input arguments and inline tests. Because a linear-time algorithm is used that generates linear-size code, the compiler can compile even large programs in a short time. In fact, the entire B-Prolog library, which has over 23,000 lines of Prolog code, can be compiled in just one second on a PC.

Action Rules for Programming Agents

The lack of a facility for programming "active" sub-goals that can be reactive to the environment has been considered one of the weaknesses of logic programming. To overcome this, B-Prolog provides a simple and yet powerful language, called Action Rules (AR), for programming agents. An agent is a subgoal that can be delayed and can later be activated by events. Each time an agent is activated, some actions may be executed. Agents are a more general notion than delay constructs in early Prolog systems and processes in concurrent logic programming languages in the sense that agents can be responsive to various kinds of events including instantiation, domain, time, and user-defined events.

A Fast CLP(FD) System

B-Prolog initially had a finite-domain constraint solver implemented in version 2.1, which was released in March 1997. That solver was implemented in an early version of AR. During the past decade, AR has been extended to support a rich class of domain events for programming constraint propagators [5] and the system has been artfully engineered to make constraint solving fast. As testified by the results in the solvers competition, B-Prolog is arguably one of the fastest CLP(FD) systems.

[1] http://www.cril.univ-artois.fr/CPAI06
[2] http://sato-www.cs.titech.ac.jp/prism/

An Efficient Tabling System

Tabling has been found increasingly important for not only helping beginners write workable declarative programs but also developing real-world applications such as natural language processing, databases, model checking, and machine learning applications. B-Prolog implements a tabling mechanism, called linear tabling [4], which is based on iterative computation of looping subgoals rather than suspension of them to compute the fixed points. Optimization techniques have been developed to significantly reduce the cost of re-computation and the tabling system in B-Prolog is now competitive in both space and time efficiencies. As demonstrated by PRISM applications, the tabling system in B-Prolog is sustainable to large data sets.

User-friendly Memory Management

The B-Prolog system has an efficient garbage collector (GC) that reclaims space used by garbage on the control stack and the heap. It also compacts the trail stack to get rid of redundantly trailed items. GC is invoked automatically when needed and the user can control the frequency of GC by setting the `gc_threshold` flag or by invoking GC explicitly. The memory manager employs techniques to reuse space in the program and table areas. All the stacks and data areas expand automatically before they overflow, so applications can run with any initial setting for the spaces as long as the overall demand for memory can be met.

Wide Accessibility

B-Prolog runs on Windows, Linux, Mac, and Solaris. It also runs on 64-bit platforms, so large applications can have access to an almost unlimited large memory space. The system provides a bi-directional interface with C and Java (JIPL by Nobukuni Kino). It provides a classical debugging environment and several useful tools including a profiler and an XML parser (by Binding Time Limited). Last but not least, B-Prolog supports a declarative and easy-to-use interface to GLPK (by Andrew Makhorin) through which LP/MIP problems can be described declaratively in Prolog syntax.

References

[1] Neng-Fa Zhou. Parameter passing and control stack management in Prolog implementation revisited. *ACM Transactions on Programming Languages and Systems*, 18(6):752–779, 1996.

[2] Neng-Fa Zhou. Programming finite-domain constraint propagators in action rules. *Theory and Practice of Logic Programming (TPLP)*, 6(5):483–508, 2006.

[3] Neng-Fa Zhou. A register-free abstract Prolog machine with jumbo instructions. In *International Conference on Logic Programming*, 2007.

[4] Neng-Fa Zhou, Taisuke Sato, and Yi-Dong Shen. Linear tabling strategies and optimizations. *Theory and Practice of Logic Programming (TPLP)*, to appear, preliminary results appear in ACM PPDP'03 and ACM PPDP'04., 2007.

[5] Neng-Fa Zhou, Mark Wallace, and Peter J. Stuckey. The `dom` event and its use in implementing constraint propagators. Technical report TR-2006013, CUNY Compute Science, 2006.