ICLP 2005 Doctoral Consortium
SiLCC Is Linear Concurrent Constraint Programming

Rémy Haemmerlé

Project Contraintes, INRIA Rocquencourt, France
Remy.Haemmerle@inria.fr

Introduction. The Concurrent Constraint (CC) [8] languages allow to model a large number of constraint programming systems, form Prolog coroutinings to constraints propagation mechanisms such these implemented in the Finite Domain (FD) solver of GNU-Prolog [2]. These languages are an extension of CLP obtained by an addition of a synchronization primitive based on constraint implication. The Linear Concurrent Constraint (LCC) [9] languages are generalizations of CC languages that consists of considerating constraint systems on Girard's linear logic [5] instead of the classical logic. This new kind of languages offer an unified framework combining constraints with state change and allow us to express the semantics of constraint programming together with concurrency and imperative features such as multiple assignment variables. It is, then, possible to defined solvers as pure libraries written in the same languages as the system.

Objectives. The prime objective of this thesis is to design and implement a LCC language called **SiLCC**, for "SiLCC is Linear Concurrent Constraint programming". The conception of this language should respected imperatives of efficiency and of programming ease. The main idea is to define a rich language combining constraints, concurrence and state change. The second objective is the realization of a completely bootstrapped implementation of a constraint system.

Hence any parts of the system could be modify or extended by any usual programmers. For this purpose we need to develop a kernel as small as possible provided with a robust modules system.

Related Works. This thesis falls under the continuity of previous works of our team. On the one hand there are the theoretical works of Fages, Ruet and Soliman [4] on semantics of LCC languages. Using refined versions of observable, they have extended Saraswat's results to obtain more precise and more general semantics. On the other hand there are the works of Diaz [2] on compilation to native code of LP programs. We plan to use his compilation technology to give efficient behavior to our system. The closest works to ours seem to be the Cabeza's ones [1]. In fact he has tried to develop "an extensible [and] global analysis friendly LP System" (Ciao Prolog). For this purpose he proposed a syntactic module system and designed a functional system in which the most of the code is written in Prolog. Nonetheless he did not provide to his modules system any formal definition and did not try to limit the size of his kernel.

Accomplish Results and Current Status. After one and half a year, the main result of this thesis concerns the module system. We have proposed in [6] a formal definition of closed syntactic module system. By closed we mean the property that the module system is able to prevent any call to the private predicates of a module from the other modules, in particular through meta-programming predicates. We show that this property necessitates to distinguish the execution of a term (meta-programming predicate `call`) from the execution of a closure (higher order). The system we propose is close to the one of CIAO Prolog in its implementation, but has the advantage of revealing the need for closures, and of positioning them w.r.t. meta-programming predicates in the framework of formal operational semantics. We have also developed a first prototype consisting of a layer, written in Prolog, on the top of GNU Prolog RH [3]. This layer is composed of a preprocessor that converts modular code into non-modular code and a new top-level, that transparently interprets modular programs and is able to dynamically compile and load modules and their dependencies.

Current Status. We are actively working on the bootstrapping of our first prototype. We think, hence, distinguish what must be native in our system from what could be rewritten as pure SiLCC libraries.

Expected Achievements and Open Issues. At the end of this thesis, we plan to have a complete implementation of SiLCC. The language will be based on a small kernel, we call LCC($\mathcal{K}$) combining a basic constraints system, concurrence and imperative features. We expect the language to be efficient and extensible, thanks to its compilation to native code, to its modular architecture and to the expressive power of its kernel. Moreover our team also plan to distribute this system with a large number of libraries such as CHR, constraints solvers, lexing/parsing, prescriptive typer. As future works, we can mention the possible implementation of a more advanced module system such as the Sanella and Wallen's one [7] extended with meta-calls and closures.

References

1. Daniel Cabeza. *An Extensible, Global Analysis Friendly Logic Programming System.* PhD thesis, Universidad Politécnica de Madrid, August 2004.
2. Daniel Diaz and Philipe Codognet. Design and implementation of the GNU Prolog system. *Journal of Functional and Logic Programming*, 6, October 2001.
3. Daniel Diaz and Rémy Haemmerlé. *GNU Prolog RH user's manual*, 1999–2004.
4. François Fages, Paul Ruet, and Sylvain Soliman. Linear concurrent constraint programming: operational and phase semantics. *Information and Computation*, 165(1):14–41, February 2001.
5. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.
6. Rémy Haemmerlé and François Fages. Closures are needed for closed module systems. Technical Report RR-5575, INRIA, 2005.
7. D. T. Sannella and L. A. Wallen. a calculus for the construction of modular Prolog programs. *Journal of Logic Programming*, pages 147–177, 1992.
8. Vijay A. Saraswat. *Concurrent constraint programming.* ACM Doctoral Dissertation Awards. MIT Press, 1993.
9. Vijay A. Saraswat and Patrick Lincoln. Higher-order linear concurrent constraint programming. Technical report, Xerox Parc, 1992.