Ajay Bansal et al.: Verification and Planning Based on Coinductive Logic Programming, Proceedings of The Sixth NASA Langley Formal Methods Workshop, p.9–11

Verification and Planning Based on Coinductive Logic Programming

Ajay Bansal, Richard Min, Luke Simon, Ajay Mallya, Gopal Gupta

Department of Computer Science, University of Texas at Dallas, USA
Contact author: Gopal Gupta, e-mail: gupta@utdallas.edu

Extended Abstract

Coinduction is a powerful technique for reasoning about unfounded sets, unbounded structures, infinite automata, and interactive computations [6]. Where induction corresponds to least fixed points semantics, coinduction corresponds to greatest fixed point semantics. Recently coinduction has been incorporated into logic programming and an elegant operational semantics developed for it [11, 12]. This operational semantics is the greatest fix point counterpart of SLD resolution (SLD resolution imparts operational semantics to least fix point based computations) and is termed co-SLD resolution. In co-SLD resolution, a predicate goal `p(`$\bar{t}$`)` succeeds if it unifies with one of its ancestor calls. In addition, rational infinite terms are allowed as arguments of predicates. Infinite terms are represented as solutions to unification equations and the occurs check is omitted during the unification process: for example, `X = [1 | X]` represents the binding of `X` to an infinite list of `1`'s. Thus, in co-SLD resolution, given a single clause

```
p([ 1 | X ]) :- p(X).
```

the query `?- p(A)` will succeed with the (infinite) answer:

```
A = [1 | A]
```

Coinductive Logic Programming (Co-LP) and Co-SLD resolution can be used to elegantly perform *model checking* and *planning*. A combined SLD and Co-SLD resolution based LP system forms the common basis for planning, scheduling, verification, model checking, and constraint solving [9, 4]. This is achieved by amalgamating SLD resolution, co-SLD resolution, and constraint logic programming [13] in a single logic programming system. Given that parallelism in logic programs can be implicitly exploited [8], complex, compute-intensive applications (planning, scheduling, model checking, etc.) can be executed in parallel on multi-core machines. Parallel execution can result in speed-ups as well as in larger instances of the problems being solved.

In the remainder we elaborate on (i) how planning can be elegantly and efficiently performed under real-time constraints, (ii) how real-time systems can be elegantly and efficiently model-checked, as well as (iii) how hybrid systems can be verified in a combined system with both co-SLD and SLD resolution. Implementations of co-SLD resolution as well as preliminary implementations of the planning and verification applications have been developed [4].

Co-LP and Model Checking: The vast majority of properties that are to be verified can be classified into *safety* properties and *liveness* properties. It is well known within model checking that safety properties can be verified by reachability analysis, i.e, if a counter-example to the property exists, it can be finitely determined by enumerating all the reachable states of the Kripke structure. Checking for reachability amounts to finding the least fixed-point, which is relatively straightforward to compute (for example, using *tabled logic programming* [2]). Verification of

liveness properties is however problematic because counterexamples to liveness properties take the form of infinite traces, which are semantically expressed as greatest fixed-points. Co-LP can be directly used to verify liveness properties by constructing counterexamples using greatest fixed-point temporal logic formulae. Intuitively, a state S is not live if there is an infinite loop (cycle) intervening between the current state and S. In a coinductive formulation, liveness also reduces to the reachability problem. Liveness counterexamples are elegantly found by (coinductively) enumerating all possible states that can be "reached" via infinite loops.

Co-LP and Planning: Coinduction can also be used to develop methods for goal-directed execution of non-monotonic logics, traditionally used for developing planners. In particular, top-down, goal-directed execution methods can be designed for *answer set programs*, a popular formalism for non-monotonic reasoning [1, 5]. Developing such a goal-directed reasoner has been an open problem for some time. It turns out that one can use co-SLD resolution to solve this problem [3]. In planning, a domain description D is given along with a set of observations about the initial state O and a collection of fluent literals $G = \{g_1, \ldots, g_l\}$, which is referred to as a goal. The problem is to find a sequence of actions $a_1, \ldots, a_n$ such that $\forall i,\ 1 \leq i \leq l$, D entails g_i from initial state O, after actions $a_1, \ldots, a_n$. The sequence of actions $a_1, \ldots, a_n$ is called a plan for goal G w.r.t. *(D,O)* [5]. Action Description Languages have been designed to encode the domain descriptions [1]. These Action Description Languages are implemented through rules of non-monotonic logic, in particular, answer set programming; these languages can be elegantly and efficiently implemented using co-LP and used for solving planning problems [4].

Timed Planning and Verification: Within logic programming, continuous time and time-deadlines can be modeled as constraints over reals [7]. Together with co-induction, this ability can be used to perform verification of timed-systems as well as perform planning under time constraints. Elsewhere we illustrate the verification of timed-systems by considering a formulation of the dining philosophers problem with stop-watches and proving that it is deadlock free (safety) and starvation free (liveness) [4]. We also illustrate the solution of time constrained planning problems using the soccer-playing planning domain extended with real-time constraints [4]. Note that a combination of generalized constraints and coinduction leads to a general framework for verifying hybrid systems as well as performing *hybrid planning*, i.e., planning under discrete and continous constraints.

References

[1] M. Balduccini, M. Gelfond, et al. An A-Prolog Decision Support System for the Space Shuttle-I. In Lecture Notes in Computer Science: Proceedings of Practical Aspects of Declarative Languages '01, Vol. 1990, pp 169-183, 2001.

[2] Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, T. Swift, and D.S. Warren. Efficient Model Checking Using Tabled Resolution. Proc. CAV'97, Haifa, Israel, Lecture Notes in Computer Science, Vol. 1243.

[3] A. Bansal, R. Min, G. Gupta. Goal-directed Execution of Answer Set Programs. UTD Technical Report. Feb. 2008.

[4] A. Bansal. Towards next generation logic programming systems. Ph.D. thesis. University of Texas at Dallas. Dec. 2007.

[5] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.

[6] J. Barwise, L. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*. CSLI Publications, 1996.

[7] G. Gupta, E. Pontelli. Constraint-based Specification and Verification of Real-time Systems. In *Proc. IEEE Real-time Symposium* (RTSS'97). pp. 230-239.

[8] G. Gupta et al. Parallel Execution of Prolog Programs: A Survey. In *ACM Transactions on Programming Languages and Systems*, Vol 23, No. 4, pp. 472-602.

[9] G. Gupta, A. Bansal, R. Min, L. Simon, A. Mallya. Coinductive Logic Programming and Its Applications. Invited Tutorial. Proc. Int'l Conf. on Logic Programming (ICLP'07). pp. 27-44.

[10] L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-Logic Programming. Proc. Int'l Conf. on Automata, Languages and Programming (ICALP'07), pp. 472-483.

[11] L. Simon, A. Mallya, A. Bansal, and G. Gupta. Coinductive Logic Programming. Int'l Conf. on Logic Prog. (ICLP'06). Springer Verlag LNCS 4079. pp. 330-344.

[12] Luke Simon. Extending Logic Programming with Coinduction. University of Texas at Dallas. Ph.D. Thesis. 2006.

[13] K. Marriott and P. Stuckey. Prog. with Constraints: An Introduction. *MIT Press, 1998*.