Constraint Logic Programming Applied to Model Checking

Laurent Fribourg

LSV, Ecole Normale Supérieure de Cachan & CNRS
61 av. Pdt. Wilson, 94235 Cachan, France
Laurent.Fribourg@lsv.ens-cachan.fr

Abstract. We review and discuss here some of the existing approaches based on CLP (*Constraint Logic Programming*) for verifying properties of various kinds of state-transition systems.

1 Introduction: Model Checking and Fixed-Point Engines

Model-checking is an automatic technique for verifying finite-state concurrent systems. Specifications are expressed in a certain temporal logic, and the concurrent system is modeled as a state-transition system [9]. There are several kinds of temporal logic, the most used in model-checking being PLTL (*Propositional Linear Temporal Logic*) [37], CTL (*Computation Tree Logic*) [8] and the mu-calculus [32]. Reachability analysis is a variant of model-checking consisting in characterizing all the reachable states of the system. This is useful for proving safety properties, i.e. verifying that something "bad" never happens to the system. More generally, model checking is the problem of finding the set of states in a state transition graph where a given formula (of the underlying temporal logic) is true. This mainly reduces to computing least fixed-point(s) or greatest fixed-point(s) for the temporal logic operators involved in the formula. Model-checking has been extended for treating infinite-state systems such as timed automata [1] or linear hybrid systems [27, 22]. Logic Programming, and more generally CLP (*Constraint Logic Programming* [31]), has been recently used by several teams of researchers for constructing "engines" that compute fixed-points related to model-checking, and benefit from general techniques of program transformation. We review and discuss here some of the proposed approaches.

The plan of the paper is as follows. Section 2 explains how fixed-point engines have been developed for mu-calculus using tabled logic programming. Section 3 reports on an extension of mu-calculus for handling constraints over finite domains. CLP-based methods for analysing automata enriched with timing constraints, are reviewed in section 4, as well as methods for automata with integer constraints in section 5. Section 6 gives some experimental results obtained with CLP-based methods. The paper ends with a discussion in section 7.

2 Encoding Mu-Calculus in Tabled Logic Programming

This section is essentially borrowed from [40]. The modal mu-calculus [32] is an expressive temporal logic whose semantics is usually described over sets of states of labeled transition systems. The logic is encoded is an equational form, the syntax of which is given by the following grammar:

```
F ::= Z | tt | ff | F\/ F | F /\ F | diam(A, F) | box(A, F)
```

In the above, `Z` is a set of formula variables (encoded as Prolog atoms) and `A` is set of actions; `tt` and `ff` are propositional constants; `\/` and `/\` are standard logical connectives; and `diam(A, F)` (possibly after action `A` formula `F` holds) and `box(A, F)` (necessarily after action `A` formula `F` holds) are dual modal operators. Additionally, logical variables can be given defining equations of the form `X :== mu(F)` (least fixed-point) or `X :== nu(F)` (greatest fixed-point). For example, a basic property, the absence of deadlock, is expressed in this logic by a formula variable `deadlock_free` defined as:

```
deadlock_free ::= nu(box(-, deadlock_free) /\ diam(-, tt))
```

where '`-`' stands for any action. The formula states, essentially, that from every reachable state (`(box(-,deadlock_free)`) a transition is possible (`diam(-,tt)`).

The labeled transition system is a *finite-state automaton*, specified in terms of a set of facts `trans(`*Src*`,` *Act*`,` *Dest*`)`, where *Src*, *Act*, and *Dest* are the source location, label and target location, respectively, of each transition. The semantics of the modal mu-calculus is specified declaratively in logic programming by providing a set of rules for each of the operators of the logic. For example, the semantics of `\/` , `diam`, and `mu` are encoded as follows:

```
models(Loc_S, (F1 \/ F2))  :- models(Loc_S, F1).
models(Loc_S, (F1 \/ F2))  :- models(Loc_S, F2).

models(Loc_S, diam(A, F))  :- trans(Loc_S, A, Loc_T),
                               models(Loc_T, F).

models(Loc_S, Z)           :-  Z :== mu(F),
                               models(Loc_S, F).
```

Consider the rule for `diam`. It states that a location `Loc_S` (of a process) satisfies a formula of the form `diam(A, F)` if `Loc_S` has an `A` transition to some location `Loc_T` and `Loc_T` satisfies `F`. Greatest fixed-point computations are encoded as negation of their dual least fixed-point. This encoding provides a sound method for model checking any modal mu-calculus formula that is alternation free [17]. An extension for implementing *full* modal mu-calculus was proposed in in [34].

The computation of fixed-points using the standard strategy of Prolog generally fails due to nontermination problems. The XSB tabled logic programming system developed at SUNY Stony Brook is an implementation of *tabled resolution* [49, 7], which is integrated into Prolog-style SLD resolution (extended with negation-as-failure rule). Many programs that would loop infinitely in Prolog will terminate in XSB because XSB calls a predicate with the same arguments only once, whereas Prolog may call such a predicate infinitely often. For these terminating programs XSB efficiently computes the least model, which is the least fixed-point of the program rules understood as "equations" over sets of atoms.

By using XSB as a programmable fixed-point engine, one can construct an efficient model checker, called XMC, under 200 lines of code. XMC benefits from optimization techniques developed for deductive-database-style applications, such as literal reordering (see, e.g., [50]) and clause resolution factoring [11], as well as from source-code representational changes of process terms. These logic-based optimization techniques lead to considerable space and time reduction. Performance can be improved even further with XSB by *compiling* high-level specifications into low-level automata of a certain kind, using continuation-passing style [14].

3 Mu-Calculus for Automata with Finite Domains

In [43], Rauzy presents *Toupie*, a system implementing an extension of the propositional mu-calculus to finite domain constraints. *Toupie* encodes arithmetic relations over finite domains by means of *decision diagrams*, an extension to finite domains of Bryant's binary decision diagrams [4]. Decision diagrams (DD), as used in *Toupie*, admit a compact canonical coding. In order to solve systems of linear equations, i.e. to compute a DD that encodes all the solution of the system, *Toupie* uses the classical implicit enumeration/propagation technique [26]. The principle is to enumerate variables in the order of their indices, and to build the DD in a bottom-up way. The adopted propagation – the Waltz's filtering – consists in maintaining, for each variable a minimum and maximum value. Each time a variable domain is modified, this modification is propagated until a fixed-point is reached.

Rauzy mentions very good performances using general purpose constraint language *Toupie* with respect to specialized model checkers. These results are attributed to the use of extended decision diagrams instead of binary ones.

4 Reachability Analysis for Timed Automata

Timed automata [1] are finite-state automata enriched with a set of clocks, which are real-valued variables that increase continuously at the same rate with time. There are two kinds of transitions: *discrete* and *delay* transitions. Discrete transition are instantaneous transitions (as in the case of finite-state automata), which make the automaton switch from one location to another. Such transitions can be performed only if a certain relationship holds among the current values of the clocks (*guard*). When they are performed, these transitions update some of the clocks to 0. Discrete transitions are interleaved with delay transitions, which correspond to the evolution of time at a certain location. Time can increase continuously through delay-transitions as long as the *location invariant* (a specific constraint on clock values associated with each location) is not violated. We now review various CLP-based methods for performing reachability analysis of timed automata.

4.1 Tabled resolution

This subsection is essentially borrowed from [16]. Timed automata are represented in [16] within a (tabled) logic programming framework. This requires the definition of two relations `inv/2` and `trans/5`:

1. `inv(L, Inv)` associates location names with location invariants.

2. `trans(L1, Act, Rho, RSL, L2)` represents a transition of the timed automaton such that `L1`, `L2`. `Act`, `Rho`, `RSL` are the source and target locations, the label of the transition, the transition conditions, and the set of clocks to be reset, respectively.

The implementation of reachability analysis is done using the integration of a library of polyhedra manipulations, as explained below. In the case of an ordinary finite-state labeled transition system, reachability can simply be computed using predicate `trans/3`, as follows:

```
reach(X,Y) :- trans(X,_,Y).
reach(X,Y) :- reach(X,Z), trans(Z,_,Y).
```

The relation `reach` can be naturally generalized using constraint representation to compute *forward reachability* in timed automata. The desired generalization of `reach` is attained in two steps. First, a state of a timed automaton is represented by a pair $\langle L, R\rangle$, where L is a location and R is a constraint representing a set of clock valuations. Secondly, the transition `trans/3` is redefined, with the help of `trans/5`, as follows:

```
trans((L1,R1), Act, (L2,R2)) :-             % discrete transition
       trans(L1,Act,Rho,RSL,L2),
       conjunction(R1,Rho,E1),
       reset(E1, RSL, E2),
       inv(L2,Inv),
       conjunction(E2, Inv, R2).
trans((L1,R1), epsilon, (L2,R2)) :-             % time successor
       time_succ(L1, R1, R2).
```

The predicate `reset(E1,RSL,E2)` computes a constraint `E2` by projecting the constraint `E1` on the surface generated by setting all the (clock) variables in `RSL` to zero. This can be implemented using a library of polyhedra manipulations. The predicate `time_succ(L,R1,R2)` computes the time successor of `R1`. This can be still computed using primitive of polyhedra manipulations [23]. `R2` is computed by intersecting the resultant region with the location invariant `L`. Likewise, backward reachability can be defined through a `back_reach` predicate by essentially reversing the application of the `trans/3` relation.

4.2 Meta-Programming in CLP($\mathcal{R}$)

A similar solution has been developed by Urbina [52, 51]. However, in contrast with [14], Urbina performs forward and backward fixed-point computation, without appealing to tabled resolution or to any special polyhedra manipulation library. Urbina uses just CLP($\mathcal{R}$) [24], its built-in real arithmetic solver and some of its meta-programming facilities (`assert`, `not`, `consistent`, `infer`, ...).

In a related work by Pontelli and Gupta [38], the simulation of timed automata is performed via the top-down operational semantics of CLP($\mathcal{R}$).

4.3 Revesz's procedure

Another solution consists in using Revesz's bottom-up evaluation procedure instead of tabled resolution or meta-programming in CLP($\mathcal{R}$). Revesz has shown that bottom-up evaluation (with subsumption) terminates for a special class of logic programs with *gap-order constraints*. Gap-order constraints are inequalities of the form $X_1 \geq X_2 + c$ where X_1 and X_2 are variables and c a non-negative integer [44]. Originally in [44], the domain of variables was assumed to be $\mathcal{Z}$, but the procedure can be slightly modify in order to involve real-valued variables instead. The reachability problem can be easily encoded under the form of a logic program with constraints. For example, consider the timed automaton with three clocks X_1, X_2, X_3, represented in Figure 1. We assume that, initially: $X_1 = X_2 = X_3 = 0$. Beside transition `in` (resp. `down`) resets clocks X_1, X_2, X_3 (resp. X_3) to 0, and has *true* (resp. $1 \leq X_2 < 2$) as a guard. Location ℓ_1 (resp. ℓ_2) has $X_1 \leq 5 \wedge X_2 < 2$ (resp. $X_1 \leq 5$) as an invariant. The reachability relation for this automaton can be encoded under the form of the following CLP program:

```
p(l0,Y1,Y2,Y3,T)      :- Y1=Y2=Y3=0.

p(l1,Y1',Y2',Y3',T') :- Y1'=T, Y2'=T, Y3'=T, T=<T', T'=<Y1'+5,
                         T'<Y'+2, p(l0,Y1,Y2,Y3,T).

p(l2,Y1',Y2',Y3',T') :- Y1'=Y1, Y2'=Y2, Y3'=T, T=<T',
                         Y2+1=<T, T<Y2+2, T'=<Y1'+5,
                         p(l1,Y1,Y2,Y3,T).
```

where T denotes a new clock variable which is never reset, and Y_i (resp. Y_i') is a variable standing for $T - X_i$ (resp. $T' - X_i'$). A further simple variable manipulation allows to transform this program into a program with only gap-order constraints, and the computation of the reachability space can be then computed using Revesz's procedure. (See [18] for details.)

5 Reachability Analysis for Automata with Integers

Automata with integers is another extended form of finite-state automata. They correspond to finite-state automata enriched with a set of integer-valued variables. A (discrete) transition can only be performed if a certain constraint over

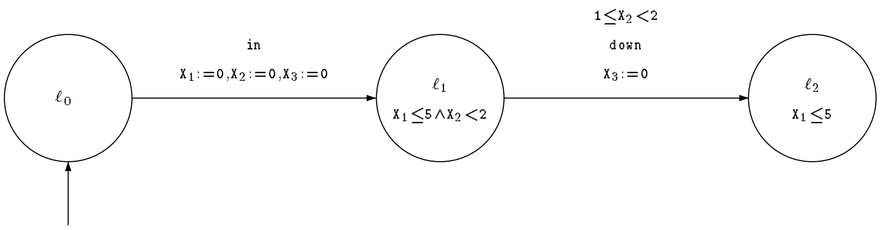

Fig. 1. A simple timed automaton

the values of these variables holds. The transition modifies the values of the variables according to a specified update relation.

5.1 Gap-order constraints.

As in the case of timed automata, it is easy to encode the reachability relation via a logic program with arithmetical constraints. When the guard and update relations all belong to the class of gap-order constraints, it is still possible to use Revesz's procedure in order to compute the set of reachable states. Otherwise, one may still use Revesz's procedure, but one has to approximate non-gap order constraints: typically, constraints of the form $X = Y + c$ are approximated under the form $X \geq Y + c$. What is then computed is still a fixed-point, but not necessarily the *least one*. However this may be sufficient to prove that some undesirable symbolic state is never reached [21].

5.2 Presburger arithmetic

In [5] the fixed-point calculations associated with transitions of automata with integers are executed symbolically using a Presburger solver called the Omega library [39]. Convergence is guaranteed by using *abstract interpretation* techniques such as "widening" [10].

In [19] a Presburger solver is similarly used in order to compute symbolically the reachable space of Petri nets, viewed as special forms of automata with integers. The fixed-point computation is optimized by integrating a method of "path decomposition" [20]. This method can be regarded, from a logic programming point of view, as a method of redundant derivation elimination [47, 33, 25, 41], or, from a model-checking point of view, as a method of *partial order* [30] (a technique for combating the combinatorial explosion that results from interleaving concurrent independent transitions in all possible orders).

5.3 Real arithmetic

An alternative proposed in [12] is to compute the least fixed-point of the reachability program via meta-programming facilities provided in SICStus Prolog via the CLP(Q,R) library [28]. Logic-based optimization techniques are also used, such as *magic set transformations* [42] or *local subsumption* [46]. Since they manipulate formulas over reals instead of integers, they add an extra abstraction for which in general a loss of precision is possible. Furthermore, in order to improve the termination of the procedure, it may be useful to appeal to classical (dynamic) techniques of approximations such as *widening* [10, 22]. Note that Delzanno and Podelski do not only treat reachability problems, but general problems of CTL logic by exhibiting a connection between CTL properties (safety, liveness) and model-theoretic or denotational semantics of logic programs with constraints.

The reachability analysis of Petri nets along the lines of [19] is further studied in [2] using the constraint solver over reals built-in HyTech instead of a Presburger solver. As in [12] the manipulation of reals instead of integers entails *a priori* a loss of precision, but it is shown in [2] that the computation of least fixed-points is actually exact when it terminates.

6 Experimental Results

We now review some of the most significant results obtained when running the CLP-based systems mentioned above.

6.1 Finite-state transition systems

XMC. In [40], XMC system is said to have good memory usage on a well-known set of benchmarks, as compared to SPIN [29], but its speed appears uneven. The speed weakness of XMC is attributed to the absence of partial order optimization [30]. In [15] results from XMC as well as 4 other model checkers are given for detecting a livelock error of the i-protocol, a sophisticated sliding-window protocol consisting of about 300 lines of C code. XMC performs much better than all the other model checkers on this example (except Murφ which has comparable results, when the error is fixed). The success of XMC is attributed to the local, top-down nature of its model-checking algorithm. In [14] further comparisons are done with an optimizing compiler for XMC, and confirms the good performance of XMC over SPIN on the i-protocol example. On another example (leader election protocol) however, SPIN significantly outperforms XMC, again because of use of partial order reduction.

Toupie. In [43], Toupie is favourably compared with two specialized model checkers on Milner's scheduler example [36]. The difference of performance in favor of Toupie is attributed to the use of extended decision diagrams instead of binary ones.

6.2 Timed automata

XMC/RT. The tabled-resolution-based XMC system, enriched with a constraint solver over reals is called XMC/RT, consists in 350 lines of Prolog code. Preliminary experiments on two standard examples (Fisher's mutual exclusion protocol, bridge crossing system) indicate that performance of XMC/RT is comparable to HyTech.

GAP. A fully automatic proof of a fundamental property of a parametric real-time protocol (ABR conformance) is obtained for the first time in [18], using an implementation, called GAP, of Revesz's bottom-up evaluation procedure.

6.3 Automata with integers

Presburger arithmetic. Omega library-based system of [5] is experimented on two example programs (bakery and ticket algorithms), for which safety and liveness properties are proved. In [19] the Presburger-based system is used for performing a parametric analysis of safety properties: the deadlock-freeness analysis of a flexible manufacturing system is, e.g., achieved for the first time in a fully mechanical way.

Real arithmetic. The SICStus-based system of [12] was experimented on several examples (among which the bakery and ticket algorithms). The execution times are about an order of magnitude shorter than those obtained in [5] using the Omega library for Presburger arithmetic.

In [2] experiments conducted with the real constraint solver of HyTech show a considerable performance improvement with respect to those obtained with Presburger solver of [19]. Besides, a deadlock is detected for the first time in a real-life protocol (PNCSA).

7 Discussion

Model-checking was originally designed for verifying finite-state transition systems. Only recently some extensions of the method were introduced in order to treat infinite-state systems.

When applied to the verification of finite-state systems, the stake of using constraint logic programming languages is (*a priori*) clear. The goal is to have a system written into a high-level language with declarative and flexible facilities while keeping good performance compared to specialized model checkers written in low-level code [40]. This goal seems to have been (partly) achieved by a system like XMC. However some qualifications should be made. First, the treated examples are often drawn out of simple sets of benchmarks. More experiments on real-life systems remain to be done. Second, in order to achieve comparable efficiency, some specialized optimizing or compiling techniques had sometimes to be introduced, which may compromise the initial objectives of flexibility and declarativeness. Third, CLP-based systems still cannot compete with specialized model-checkers on several examples, due to the lack of integration of partial order techniques.

As regards to infinite-state systems, the situation of CLP-based systems compared with specialized model-checkers is much less clear, mainly because the frontier between the two classes of systems is blurred. For example, should HyTech system be considered as a specialized model-checker or as a CLP-based one? On the one hand, HyTech is a specialized model-checker as it is written in C, and is designed to treat the class of so-called linear hybrid systems. On the other hand, it can be considered as a CLP-based system as it makes use of a library of convex real-valued constraints manipulation. The main difference between HyTech and a system like XMC/RT may just lie in the language in which they are written. In this context, all the methods are more or less at the same level, and may be called "constraint-based", as they necessarily appeal to symbolic manipulation of data. Now, to some extent, this observation also applies to *finite-state* systems model-checkers. For example, why should a system like Toupie be called "constraint-based" system, and not a specialized model-checker like SMV [35] which, still, relies basically also on (binary) decision diagrams for manipulating Boolean? Here again, the main distinctive feature may be the level of programming language in which the two systems are implemented.

Let us also mention an issue, which has received considerable attention from the model-checking community: the use of specialized data-structures for representing constraints. Binary Decision Diagrams (BDD) are thus considered in the model-checking community as a decisive breakthrough explaining the success of finite-state verification tools. One explanation for this success is that BDD provide a *canonical form* for boolean formulas that is often substantially more compact than conjunctive or disjunctive normal form, and very efficient algorithms have been developed for manipulating them [9]. Research in model-checking for infinite-state systems also exploits the use of such compact canonical forms for representing real-time constraints, such as "Difference Bounded Matrices" [13] or "Clock Difference Diagrams" [3] for timed automata, and "closed convex polyhedra" for linear hybrid systems [27, 22]. How can such data structures be smoothly and efficiently implemented in the framework of high-level languages such as CLP? This question has not yet been treated, except in the work of [43] and the recent work of [16]. We believe that performances of CLP-based systems can benefit considerably from this integration. However it would contribute to blur further the border between classical model-checkers and CLP-based systems.

Finally, let us point out that several researchers have also addressed a slightly different but very interesting and related issue: the relationship between static analysis (instead of constraint solving) and model-checking [6, 45, 48].

References

1. R. Alur and D. Dill. "Automata for Modeling Real-Time Systems". *Proc. 17th ICALP*, LNCS 443, 1990, pp. 322–335.
2. B. Bérard and L. Fribourg. "Reachability Analysis of (Timed) Petri Nets Using Real Arithmetic". *CONCUR'99*, LNCS 1664, Springer-Verlag, 1999, pp. 178–193.
3. G. Behrmann, K.G. Larsen, J. Pearson, C. Weise and W. Yi. "Efficient Timed Reachability Analysis Using Clock Difference Diagrams". *CAV'99*, LNCS 1633, Springer-Verlag, 1999, pp. 341–353.
4. R. Bryant. "Graph Based Algorithms for Boolean Function Manipulation". *IEEE transactions on Computers* 35:8, 1986, pp. 677–691.
5. T. Bultan, R. Gerber and W. Pugh. "Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics". *CAV'97*, LNCS 1254, Springer-Verlag, 1997, pp. 400–411.
6. W. Charatonik and A. Podelski. "Set-Based Analysis of Reactive Infinite-State Systems". *TACAS'98*, LNCS 1384, Springer-Verlag, 1998, pp. 358-375.
7. W. Chen and D.S. Warren. "Tabled evaluation with delaying for general logic programs". *J. ACM* 43:1, 1996, pp. 20–74.
8. E.M. Clarke and E.A. Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic." *Logic of Programs Workshop*, LNCS 131, Springer-Verlag, 1981, pp. 52–71.
9. E. Clarke, O. Grumberg and D. Long. "Verification Tools for Finite-State Concurrent Systems". *REX school/symposium on A decade of concurrency: reflections and perspectives*, LNCS 803, Springer-Verlag, 1993.
10. P. Cousot and N. Halbwachs. "Automatic Discovery of Linear Restraints Among Variables of A Program". *POPL'78*, ACM Press, 1978, pp. 84–97.
11. S. Dawson, C.R. Ramakrishnan, I.V. Ramakrishnan and T. Swift. "Optimizing clause resolution: Beyond unification factoring". *Intl. Logic Programming Symp.*, MIT Press, 1995, pp. 194–208.
12. G. Delzanno and A. Podelski. "Model Checking in CLP". *TACAS'99*, LNCS 1579, Springer-Verlag, 1999, pp. 74-88.
13. D. Dill. "Timing Assumptions and Verification of Finite-State Concurrent Systems." *Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer-Verlag, 1989, pp. 197–212.
14. Y. Dong and C.R. Ramakrishnan. "An Optimizing Compiler for Efficient model Checking". *FORTE/PSTV'99*, Beijing, 1999.
15. Y. Dong, X. Du, Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, O. Sokolsky, E.W. Stark and D.S. Warren. "Fighting Livelock in the i-Protocol: A Comparative Study of Verification Tools". *TACAS'99*, LNCS 1579, Springer-Verlag, 1999, pp. 74–88.
16. X. Du, C.R. Ramakrishnan and S.A. Smolka. "Tabled Resolution + Constraints: A Recipe for Model Checking Real-Time Systems", 1999. (`http://www.cs.sunysb.edu/ vicdu/`)
17. E.A. Emerson and C.-L. Lei. "Efficient model checking in fragments of the propositional mu-calculus". *LICS'86*, 1986, pp. 267–278.

18. L. Fribourg. "A Closed-Form Evaluation for Extended Timed Automata". *Technical Report LSV-98-2*, CNRS & Ecole Normale Supérieure de Cachan, March 1998. (`http://www.lsv.ens-cachan.fr/Publis/`)
19. L. Fribourg and H. Olsén. "Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmetic". *CONCUR'97*, LNCS 1243, Springer-Verlag, 1997, pp. 96–107.
20. L. Fribourg and H. Olsén. "A Decompositional Approach for Computing Least Fixed-Points of Datalog Programs with $\mathcal{Z}$-Counters". *Constraints: An International Journal*: 2, 1997, pp. 305–335.
21. L. Fribourg and J. Richardson. "Symbolic Verification with Gap-Order Constraints". *Proc. 6th Intl. Workshop on Logic Program Synthesis and Transformation (LOPSTR)*, LNCS 1207, 1996, pp. 20–37.
22. N. Halbwachs, Y.-E. Proy and P. Raymond. "Verification of Linear Hybrid Systems by means of Convex Approximations". *SAS'94*, LNCS, Springer-Verlag, 1994, pp. 223–237.
23. N. Halbwachs, Y.-E. Proy and P. Roumanoff. "Verification of Real-time Systems using Linear Relation Analysis". *Formal Methods in System Design* 11:2, 1997, pp. 157–185.
24. N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey and R. Yap. *The CLP($\mathcal{R}$) Programmer's Manual Version 1.2.* 1992.
25. A.R Helm. "On the detection and elimination of redundant derivations during bottom-up execution". *North American Conf. on Logic Programming*, Cleveland, Ohio, 1989, pp.945–961.
26. P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, 1989.
27. T. Henzinger, P.-H. Ho and H. Wong-Toi. "A User Guide to HYTECH". *Proc. TACAS'95*, LNCS 1019, 1995, pp. 41–71.
28. C. Holzbaur. *OFAI CLP(Q,R), Manual Edition 1.3.3.* Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.
29. G.J. Holzmann and D. Peled. "The state of SPIN". *CAV'96*, LNCS 1102, Springer-Verlag, 1996, pp. 385–389.
30. G.J. Holzmann, D. Peled and V. Pratt, editors. *Partial-Order Methods in Verification (POMIV'96).* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.
31. J. Jaffar and J.-L. Lassez. "Constraint Logic Programming". *POPL'87*, ACM Press, 1987, pp. 111–119.
32. D. Kozen. "Results on the propositional μ-calculus". *Theoretical Computer Science* 27, 1983, pp. 333–354.
33. J.-L. Lassez and M.J. Maher. "The denotational semantics of Horn clauses as a production system." *AAAI 83*, washington D.C., 1983, pp. 229–231.
34. X. Liu, C.R. Ramakrishnan and S.A. Smolka. "Fully local and efficient evaluation of alternating fixed points". *TACAS'98*, LNCS 1384, Springer-Verlag, 1998, pp. 5–19.
35. K.L. McMillan. *Symbolic Model-Checking: An Approach to the State Explosion Problem.* Kluwer Academic, 1993.
36. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
37. A. Pnueli. "The temporal semantics of concurrent programs". *TCS 13*, 1981, pp. 45–60.
38. E. Pontelli and G. Gupta. "A Constraint-based Approach for Specification and Verification of Real-time Systems". *RTSS'97.* IEEE Computer Society, 1997.

39. W. Pugh. "A Practical Integer Algorithm for Exact Array Dependence Analysis". *C.ACM* 35:8, 1992, pp. 102–114.
40. Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, T. Swift and D.S. Warren. "Efficient Model Checking Using Tabled Resolution". *CAV'97*, LNCS 1254, Springer-Verlag, 1997, pp. 143–154.
41. R. Ramakrishnan, Y. Sagiv, J.D. Ulmann and M.Y. Vardi. "Proof-Tree Transformation Theorems and their Application". *8th ACM Symp. on Principles of Database Systems*, Philadelphia, 1989, pp. 172–181.
42. R. Ramakrishnan, D. Srivastava and S. Sudarshan. "Efficient Bottom-up Evaluation of Logic Programs". *Computer Systems and Software Engineering: State-of-the-Art*, chapter 11. Kluwer Academic, 1992.
43. A. Rauzy. "Toupie: A Constraint Language for Model Checking". *Constraint Programming: Basics and Trends*, LNCS 910, Springer-Verlag, 1994, pp. 193–208.
44. P.Z. Revesz. "A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints", *Theoretical Computer Science*, 1993, vol. 116, pp. 117-149.
45. D.A. Schmidt. "Data Flow Analysis is Model Checking of Abstract Interpretations". *POPL'98*, ACM Press, 1998.
46. D. Srivastava. "Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints". *Annals of Mathematics and Artificial Intelligence* 8:3-4, 1993, pp. 315–343.
47. D. Srivastava and R. Ramakrishnan. "Pushing Constraints Selections". *11th ACM Symp. on Principles of Database Systems*, San Diego, 1992, pp. 301–315.
48. B. Steffen. "Data flow analysis as model checking". *TACS'91*, LNCS 526, Springer-Verlag, 1991.
49. H. Tamaki and T. Sato. "OLDT resolution with tabulation". *Intl. Conf. on Logic Programming*. MIT Press, 1986, pp. 84–98.
50. J.D. Ullman. *Principles of Data and Knowledge-base Systems*,vol.1 . Computer Science Press, Rockville, MD, 1988.
51. L. Urbina. "Analysis of Hybrid Systems in CLP($\mathcal{R}$)". *CP'96*, LNCS 1102, Springer-Verlag, 1996.
52. L. Urbina and G. Riedewald. "Simulation and Verification in Constraint Logic Programming". *2nd European Workshop on Real-Time and Hybrid Systems*, 1995, pp. 85–104.