Constraint-Based Infinite Model Checking and Tabulation for Stratified CLP

Witold Charatonik[1,2], Supratik Mukhopadhyay[1,3⋆], and Andreas Podelski[1]

[1] Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
{witold|supratik|podelski}@mpi-sb.mpg.de
[2] University of Wrocław, Poland
[3] University of Pennsylvania

Abstract. Forward analysis procedures for infinite-state systems such as timed systems were limited to safety properties. We give the first constraint-based forward analysis for infinite-state systems that goes beyond safety properties. Namely, we take the restriction of the μ-calculus to least-fixpoint formulas where negation is applied to closed subformulas only. We characterize these properties as perfect models of constraint logic programs, and we present a tabulation procedure for the top-down evaluation of stratified constraint logic programs.

1 Introduction

Symbolic model checking for systems with variables over an infinite numeric domain, e.g. timed or hybrid systems, integer-valued network protocols or models of parametrized systems, is an important topic of research (see e.g. [1–4, 14, 13, 9, 10, 16, 15, 18, 24, 26, 23, 28, 29, 39]). Here, 'symbolic' refers to the representation of a generally infinite set of states by a disjunctive constraint, i.e. a disjunction of conjunctions of atomic formulas with free variables (such as inequalities $x \leq y+c$ for timed systems).

One distinguishes between forward and backward methods. For example, for reachability properties ("no bad state can be reached"), the backward analysis starts with a constraint representing the 'bad' states and iteratively (1) goes through the disjunction, (2) for each disjunct computes new constraints (representing predecessor states) and (3) adds the new constraints to the disjunction. The forward analysis is defined accordingly; it starts with initial states and iteratively computes successor states. The backward method extends directly to general temporal logic properties defined e.g. in the μ-calculus (with the usual 'backward' modalities). When it exists, the forward method is preferred in practice because it does not explore unreachable states. For example, the tool UppAal uses forward analysis for specific safety properties of timed systems [7].

⋆ Support from the grants NSF award CCR99-70925, SRC award 99-TJ-688, and DARPA ITO Mobies award F33615-00-C-1707 is gratefully acknowledged.

So far, forward methods for infinite-state systems have been developed only for safety properties. It seems difficult to extend such methods to existing temporal logics such as Computation Tree Logic (CTL), the Alternation-Free μ-calculus AFμ, and more expressive fragments of μ-calculus. The difficulty arises with the operators AX and EG in CTL and the greatest-fixpoint operator in AFμ together with constraints.

In this paper, we give a forward method for a new temporal logic that we call Sμ. It does not contain any of the above-mentioned operators, but it subsumes safety properties and it can express interesting liveness properties such as *convergence* (a.k.a. zenoness or timelock) [24, 39] for timed automata.

Sμ is the subset of all μ-calculus formulas (built up with the Boolean operators, the existential predecessor operator EX and and the least fixpoint combinator μ) where negation is applied to closed subformulas only.

We present the forward method for Sμ as a tabulation procedure for the top-down evaluation of stratified constraint logic programs. This is possible because we characterize Sμ properties as perfect models of such programs.

In the next section, we define Sμ as a subset of AFμ and investigate its expressiveness. In fact, we have a hierarchy where two levels correspond to already existing subsets of branching time logic, namely STL (see e.g. [5] and EF-Logic (see e.g. [18]), and the third, most general one corresponds to Sμ. We also show that, for finite systems, the lower bounds of the model checking problem for Sμ and AFμ are the same. In Section 3 we express Sμ properties in terms of the perfect model semantics for stratified constraint logic programs. In Section 4 we present a tabled-resolution procedure for computing this semantics and prove its correctness. Finally, in Section 5 we discuss the relation of our work with existing work and the *convergence* problem for timed automata.

2 The Temporal Logic Sμ

We first give the syntax of (closed) formulas φ of the modal μ-calculus and then define its subset Sμ.

$$\varphi \ ::= \ p \mid x \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \Diamond\varphi \mid \mu x.\, \varphi$$

Here p is an atomic proposition, x is a variable, $\Diamond$ is the next operator (written EX in CTL syntax), and in $\mu x.\varphi$, φ is monotone in x (i.e., all occurrences of the free variable x in φ lie in the scope of an even number of negations), and the least fixed point operator μx is considered to be a binder for the variable x. A formula φ is closed if it does not contain free variables. If we require that x does not occur free in any negative occurrence of a subformula $\neg\mu y.\psi$ of $\mu x.\, \varphi$, we obtain the *alternation-free* fragment of the μ-calculus AFμ.

Sometimes we use the formulas $\nu x.\varphi(x)$ and $\Box\varphi$ informally as abbreviations for $\neg\mu x.\neg\varphi(\neg x)$ and $\neg\Diamond\neg\varphi$, respectively.

Definition 1 (Sμ). *An Sμ formula is a closed formula φ of the modal μ-calculus such that in any subformula $\neg\psi$ of φ, the formula ψ is closed.*

The above definition immediately implies that the logic Sμ is a fragment of alternation-free μ-calculus. The example of the formula $\mu x.p \vee \neg\Diamond\neg x$ (abbreviated $\mu x.p \vee \Box x$, and written $\mathsf{AF}(p)$ in CTL syntax) of the alternation-free μ-calculus shows that Sμ is strictly less expressive than AFμ.

The Fisher-Ladner closure $cl(\varphi)$ of a formula φ is the smallest set of formulas containing φ such that

- if $\psi \in cl(\varphi)$ and ψ' is a subformula of ψ then $\psi' \in cl(\varphi)$, and
- if $\mu x.\psi(x) \in cl(\varphi)$ then $\psi(\mu x.\psi(x)) \in cl(\varphi)$.

Here, for a formula $\mu x.\psi(x)$, the formula $\psi(\mu x.\psi(x))$ is obtained from $\psi(x)$ by replacing each free occurrence of x with $\mu x.\psi(x)$.

Proposition 1. *Let φ be a modal μ-calculus formula. The following are equivalent.*

- *φ is an Sμ formula,*
- *the is-subformula relation over $cl(\varphi)$ can be made a partial order $\preceq$ that is strict wrt. to negation, i.e., $\varphi' \prec \neg\varphi'$,*
- *there exists a* stratification *function $\mathcal{S}$ assigning a natural number to each formula in the closure $cl(\varphi)$ such that*
  - *if $\varphi'' \in cl(\varphi')$ then $\mathcal{S}(\varphi'') \leq \mathcal{S}(\varphi')$, and*
  - *$\mathcal{S}(\varphi') < \mathcal{S}(\neg\varphi')$ for all $\varphi' \in cl(\varphi)$.*

Proof. By inspection. $\Box$

Thus, Sμ consists of all *stratifiable* formulas of the modal μ-calculus (in the syntax given above, i.e., with least fixpoints and Boolean set operators including the complement, but *without* greatest fixpoints).

Example 1. Consider once more the formula $\mu x.p \vee \neg\Diamond\neg x$. To see that it is not stratified, suppose that there exists an appropriate stratification function $\mathcal{S}$. Let $\psi(x)$ be the formula $p \vee \neg\Diamond\neg x$. Then $\mathcal{S}(\mu x.\psi(x)) \geq \mathcal{S}(\psi(\mu x.\psi(x))) \geq \mathcal{S}(\neg\Diamond\neg\mu x.\psi(x)) > \mathcal{S}(\Diamond\neg\mu x.\psi(x)) \geq \mathcal{S}(\neg\mu x.\psi(x)) > \mathcal{S}(\mu x.\psi(x))$, which is a contradiction.

Figure 1 situates the expressiveness of Sμ relative to the alternation-free μ-calculus, CTL and STL (i.e., STL is a subset of both CTL and Sμ). We recall the definition of the syntax of STL and of its sublogic, the EF-Logic.

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \mathsf{EX}(\varphi) \mid \mathsf{EF}(\varphi) \qquad \text{(EF-Logic)}$$
$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \mathsf{EX}(\varphi) \mid \varphi\mathsf{EU}\varphi' \qquad \text{(STL)}$$

The inclusion EF $\subseteq$ STL $\subseteq$ Sμ is obvious if we use the usual encoding of CTL in AFμ. To see that Sμ is incomparable with CTL in terms of expressiveness, observe that (1) every satisfiable formula of Sμ has a model whose paths are all finite; the CTL formula $\mathsf{EG}(p)$, for example, does not have such a model; (2) the Sμ formula $\mu x.p \vee \Diamond\Diamond x$ expresses the property "p is reachable in an even number of steps", which cannot be expressed in CTL.

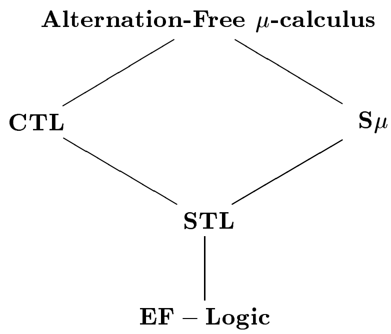

Fig. 1. A lattice for situating the expressiveness of Sμ and STL

Our motivation for Sμ stems from infinite-state systems with generally undecidable model checking problems. In fact, the following proposition shows that the restriction of the alternation-free μ-calculus to Sμ does not trade with a decrease of the theoretical complexity of model checking for finite-state systems. As in the alternation-free μ-calculus, the problem is hard for P even if the formula is fixed. The existing P-hardness proofs for the alternation-free μ-calculus, however, reduce the alternating graph-reachability problem, using the $\Box$ modality in an essential way, and cannot be carried over directly. Our proof is close in spirit to the proof of the P-completeness of Datalog LIT in [22]; we also reduce the monotone circuit value problem (MCVP), a well-known P-complete problem.

Proposition 2 (Finite-state systems). *The program complexity of Sμ model checking is P-complete.*

Proof. An instance of MCVP is a sequence of variables $X_1, \ldots, X_n$ and boolean equations of the form $X_i = \mathit{true}$, $X_i = \mathit{false}$, $X_i = X_j \wedge X_k$ or $X_i = X_j \vee X_k$ where for all equation of the last two forms we have $i > \max(j, k)$, such that the value of a variable does not depend on itself. The question we ask is whether the value of X_n is true.

Given such an instance I of MCVP, we construct a Kripke structure $K = \langle S, \rightarrow, L\rangle$ as follows. We define the set of atomic propositions as $\{p, p_1, p_2\}$. The set S of states consists of the variables $X_1, \ldots, X_n$ and their copies (two for each variable) $X_1^1, X_1^2, \ldots, X_n^1, X_n^2$. The labeling function $L : S \rightarrow 2^{\{p,p_1,p_2\}}$ is defined as follows. $L(X_i) = \{p\}$ iff $X_i = \mathit{true}$ is in I. For all $i = 1, \ldots, n$ we define $L(X_i^1) = \{p_1\}$ and $L(X_i^2) = \{p_2\}$. The transition relation $\rightarrow$ is defined as follows. For all equations $X_i = X_j \vee X_k$ we have $X_i \rightarrow X_j$ and $X_i \rightarrow X_k$, and for all equations $X_i = X_j \wedge X_k$ we have $X_i \rightarrow X_i^1$, $X_i \rightarrow X_i^2$ $X_i^1 \rightarrow X_j$ and $X_i^2 \rightarrow X_k$. Now it is easy to see that the state X_n in the structure K satisfies the formula

$$\mu x.p \vee \Diamond x \vee (\Diamond(p_1 \wedge \Diamond x) \wedge (\Diamond(p_2 \wedge \Diamond x)))$$

if and only if the value of the variable X_n in the circuit is true. $\Box$

3 Translation

If we restrict the translation of [22] to Sμ, we can formulate it in terms of Stratified CLP (stratified constraint logic programs, see [27]). The parametric constraint system of the CLP paradigm is used to express the transition relation of the system to be checked (e.g. arithmetic constraints over reals for timed or hybrid systems).

We assume the definition of predicates $\mathsf{trans}(\mathbf{s},\mathbf{s}')$, $\mathsf{init}(\mathbf{s})$ and $p_{ap}(\mathbf{s})$ expressing, respectively, that there is a transition from the state $\mathbf{s}$ to $\mathbf{s}'$, that $\mathbf{s}$ is an initial state, and that the atomic proposition ap holds in the state $\mathbf{s}$.

We translate an Sμ formula ψ to the constraint logic program $\mathcal{P}_\psi$ that contains all clauses defining the predicate trans and the clauses below defining new predicates p_φ, for each formula φ in the Fisher-Ladner closure $cl(\psi)$.

$$\begin{array}{lr@{\;}l}
(1) & p_{\varphi_1\wedge\varphi_2}(\mathbf{s}) \leftarrow & p_{\varphi_1}(\mathbf{s}), p_{\varphi_2}(\mathbf{s})\\
(2) & p_{\varphi_1\vee\varphi_2}(\mathbf{s}) \leftarrow & p_{\varphi_1}(\mathbf{s})\\
 & p_{\varphi_1\vee\varphi_2}(\mathbf{s}) \leftarrow & p_{\varphi_2}(\mathbf{s})\\
(3) & p_{\neg\varphi}(\mathbf{s}) \leftarrow & \mathsf{not}(p_\varphi(\mathbf{s}))\\
(4) & p_{\diamond\varphi}(\mathbf{s}) \leftarrow & \mathsf{trans}(\mathbf{s},\mathbf{s}'), p_\varphi(\mathbf{s}')\\
(5) & p_{\mu x.\varphi(x)}(\mathbf{s}) \leftarrow & p_{\varphi(\mu x.\varphi(x))}(\mathbf{s})
\end{array}$$

We can abbreviate the translation of formulas of EF-Logic or STL(i.e. without first expressing them as Sμ formulas).

$$\begin{array}{lr@{\;}l}
(6) & p_{\mathsf{EF}\varphi}(\mathbf{s}) \leftarrow & p_\varphi(\mathbf{s})\\
 & p_{\mathsf{EF}\varphi}(\mathbf{s}) \leftarrow & \mathsf{trans}(\mathbf{s},\mathbf{s}'), p_{\mathsf{EF}\varphi}(\mathbf{s}')\\
(7) & p_{\varphi_1\mathsf{EU}\varphi_2}(\mathbf{s}) \leftarrow & p_{\varphi_2}(\mathbf{s})\\
 & p_{\varphi_1\mathsf{EU}\varphi_2}(\mathbf{s}) \leftarrow & p_{\varphi_1}(\mathbf{s}), \mathsf{trans}(\mathbf{s},\mathbf{s}'), p_{\varphi_1\mathsf{EU}\varphi_2}(\mathbf{s}')
\end{array}$$

The resulting constraint logic program is *stratified*, which means that there exists a mapping assigning each predicate p its *level* such that in every clause of the form $p(\boldsymbol{x}) \longleftarrow B \wedge \varphi$, the level of the predicate of an atom occurring positively in B is less than or equal to that of p, and the level of the predicate of an atom occurring negatively in B is strictly less than the level of p.

The original definition of a *perfect model* of a stratified constraint logic program $\mathcal{P}$ is model-theoretic [32]. In the proof of our theorem below, we use the following equivalent characterization of [6].

The perfect model $pm(\mathcal{P})$ of the stratified constraint logic program $\mathcal{P}$ with maximum predicate level k, is the set M_k in a sequence $M_{-1}, M_0, \ldots, M_k$ where
— $M_{-1} = \emptyset$, and
— M_j is the least fixpoint of $T^j_{\mathcal{P}}$, the restriction of the logical consequence operator $T_{\mathcal{P}}$ to the lattice C_j that contains M_{j-1} and all subsets of atoms with predicates of level j,

$$C_j = \{M_{j-1}\} \cup 2^{\{p(\boldsymbol{v})\in B_{\mathcal{D}} \mid level(p)=j\}}.$$

Here, given the constraint domain $\mathcal{D}$ for the program $\mathcal{P}$, $B_{\mathcal{D}}$ denotes the $\mathcal{D}$-base $\mathcal{P}$ (in generalization of the Herbrand base). In the sequel, we assume that the constraint domain $\mathcal{D}$ admits elimination of existential and universal quantifiers.

We define the denotation of a predicate p in the perfect model of a program $\mathcal{P}$ by

$$[\![p]\!]_{pm(\mathcal{P})} = \{\boldsymbol{v} \mid p(\boldsymbol{v}) \in pm(\mathcal{P})\}.$$

Similarly, for other sets M, the denotation of p in M is $[\![p]\!]_M = \{\boldsymbol{v} \mid p(\boldsymbol{v}) \in M\}$. For negative literals $\neg p$ we define $[\![\neg p]\!]_{pm(\mathcal{P})} = \mathcal{D}^n - [\![p]\!]_{pm(\mathcal{P})}$ where n is the arity of the predicate p. We use a similar notation for the denotation of Sμ formulas: if a transition system is defined by a program $\mathcal{P}$ as above, the set of states satisfying the formula φ is denoted by $[\![\varphi]\!]_{\mathcal{P}}$.

Theorem 1 (Adequacy of Translation). *The translation maps a Sμ formula φ to a stratified constraint logic program $\mathcal{P}_\varphi$ such that the atom $p_\varphi(\mathbf{s})$ belongs to the perfect model of the program $\mathcal{P}_\varphi$ if and only if φ is true of the state* $\mathbf{s}$*; or: the denotation of the predicate p_φ in the perfect model semantics of the program $\mathcal{P}_\varphi$ is exactly the denotation of the Sμ formula φ wrt. the transition system defined by $\mathcal{P}$,*

$$[\![p_\varphi]\!]_{pm(\mathcal{P}_\varphi)} = [\![\varphi]\!]_{\mathcal{P}}.$$

4 Tabulation

We now present a tabulation procedure for the top-down query evaluation of a query for a stratified constraint logic program, which, by Theorem 1, is a local symbolic model checking procedure for Sμ formulas. The input is a query consisting of an atom $p(\boldsymbol{x})$ and a constraint Φ. Upon termination the answer is a *complete* list of answer constraints $[\Phi_1, \ldots, \Phi_n]$. This means: for every tuple $\boldsymbol{v}$ of values for the argument tuple $\boldsymbol{x}$, the atom $p(\boldsymbol{v})$ lies in the perfect model of $\mathcal{P}$ and $\boldsymbol{v}$ is a solution of the constraint Φ if and only if $\boldsymbol{v}$ is a solution of one of the answer constraints Φ_i. In short: the denotation of the predicate p in the perfect model semantics of the program $\mathcal{P}$ intersected with the set of solutions of Φ is the set of solutions of all answer constraints, i.e., of the disjunction $\Phi_1 \vee \ldots \vee \Phi_n$,

$$[\![p]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!] = [\![\Phi_1 \vee \ldots \vee \Phi_n]\!]$$

where for a constraint Φ its denotation is its set of solutions $[\![\Phi]\!] = \{\boldsymbol{v} \mid \boldsymbol{v} \models \Phi\}$.

If the program $\mathcal{P}$ arises from the translation given in the previous section (i.e., it is of the form $\mathcal{P}_\psi$ for an Sμ formula ψ) and the constraint Φ describes the set of initial states, then tabled resolution starting with the query $\langle p(\boldsymbol{x}), \Phi\rangle$ corresponds to forward analysis (depth-first or breadth-first, depending on the selection strategy of the resolution procedure; here, we consider a depth-first procedure). The answer constraints then specify which of the initial states satisfy the Sμ formula ψ (possibly all of them, namely if the disjunction is equivalent to Φ).

The procedure is based on *tabled resolution*. Originally tabled resolution was an execution strategy for definite logic programs [37]; it was extended to logic programs with negation in [34, 11], but for soundness reasons its use in the case of negative literals is limited to ground queries. In this paper we extend it to handle constraints and nonground constructive negation (i.e., constructive negation with nonground queries).

The central data structure is a table T of answer constraints. An index in the extendable table is a query of the form $\langle p'(\boldsymbol{x}), \Phi' \rangle$ (consisting of an atom $p'(\boldsymbol{x})$ and a constraint Φ'); the corresponding field contains a list of answers to that query that have been computed to far (we will often identify a list of constraints with the disjunction of these constraints). Initially, the table has only one index, namely the original query $\langle p(\boldsymbol{x}), \Phi \rangle$, whose field contains the empty list.

The procedure is defined as the iterative application of the rules in Figure 2 until a fixpoint is reached. We start with the initial query $\langle p(\boldsymbol{x}), \Phi \rangle$ by creating an index with this query in the table T. Iteratively, we apply resolution steps, hereby creating new indexes for the new queries and thus extending the table (for the two extension rules), and we store all answers to a query encountered so far in the table T (for the two completion rules). The stored answers are reused in further resolution steps. This goes on until a fixpoint is reached.

To force the termination of the computation on all programs where the usual resolution terminates as well as many other programs, we obey the following strategy.

- All constraints occurring in the table (both in the indexes and entries) must be satisfiable (unsatisfiable constraints are simply omitted),
- Redundant indexes are not created. If the table contains an index $(p(\boldsymbol{x}), \Psi)$ and $\Phi \models \Psi$ then the index $(p(\boldsymbol{x}), \Phi)$ is not created; that is, the respective extension rule is not applied.
- Redundant answers are not stored. If for some query Q the entry $T(Q)$ contains a constraint Ψ and $\Phi \models \Psi$ then the constraint Φ is not added to $T(Q)$; that is, the positive completion rule is not applied.
- For a given query $\langle \neg p(\boldsymbol{x}), \Phi \rangle$, the negative completion rule is applied only once.

We now discuss briefly the rules in Figure 2. The two positive rules extend (in higher-level presentation) the original tabling procedure [37] to handle constraints. Their application correspond directly to one step of resolution. In both cases the query $(p(\boldsymbol{x}), \Phi)$ is resolved with, respectively, the clauses $p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge l(\boldsymbol{y}) \wedge R \wedge \Phi'$ and $p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge \Phi'$. The answers to the calls of $l_i(\boldsymbol{y}_i)$ are not computed, but taken from the table. (Note that the conjunction $\bigwedge_i l_i(\boldsymbol{y}_i)$ may be empty; this accommodates the case where $l(\boldsymbol{y})$ is the first literal of the clause.) Here, R represents the rest (the conjunction of the remaining literals) of the body in the first clause. In the first case the resolution results in a new query, in the second case the result is an answer that is stored in the table (unless it is redundant).

The two negative rules are to handle constructive negation. In order to compute an answer to a negative query $\langle \neg p(\boldsymbol{x}), \Phi \rangle$ we first run the procedure for

(positive extension)

$$\frac{\begin{array}{l} \mathit{Index}(p(\boldsymbol{x}), \Phi) \\ p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge l(\boldsymbol{y}) \wedge R \wedge \Phi' \in \mathcal{P} \\ \Phi_i \in T(l_i(\boldsymbol{y}_i), \Psi_i) \text{ where } \Phi \wedge \Phi' \models \Psi_i \text{ for all } i \end{array}}{\mathit{Index}(l(\boldsymbol{y}), \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi')}$$

(positive completion)

$$\frac{\begin{array}{l} \mathit{Index}(p(\boldsymbol{x}), \Phi) \\ p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge \Phi' \in \mathcal{P} \\ \Phi_i \in T(l_i(\boldsymbol{y}_i), \Psi_i) \text{ where } \Phi \wedge \Phi' \models \Psi_i \text{ for all } i \end{array}}{\exists_{-\boldsymbol{x}} \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi' \in T(p(\boldsymbol{x}), \Phi)}$$

(negative extension)

$$\frac{\mathit{Index}(\neg p(\boldsymbol{x}), \Phi)}{\mathit{Index}(p(\boldsymbol{x}), \Phi)}$$

(negative completion)

$$\frac{\begin{array}{l} \mathit{Index}(\neg p(\boldsymbol{x}), \Phi) \\ \Phi \models \Psi \\ \mathit{Index}(p(\boldsymbol{x}), \Psi) \end{array}}{T(\neg p(\boldsymbol{x}), \Phi) = \mathit{DNF}(\Phi \wedge \neg T(p(\boldsymbol{x}), \Psi))} \quad \text{(provided } T(p(\boldsymbol{x}), \Psi) \text{ is fully computed)}$$

Fig. 2. Rules for the tabulation procedure

the positive query (it is important here that we collect *all* the answers for the positive query) and then negate the disjunction of the answers. Note that every entry in the table is a disjunction of constraints, so we have to bring the negation of the disjunction into disjunctive normal form. Note also that if the program is stratified, the computation for $p(\boldsymbol{x})$ never involves computing $\neg p(\boldsymbol{y})$ and thus the side condition "$T(p(\boldsymbol{x}), \Psi)$ is fully computed" is well-founded.

If the procedure terminates (which in general cannot be guaranteed for decidability reasons), then the table contains all answers to all sub-queries and to the original query $p(\boldsymbol{x})$.

The examples below give an intuition about the procedure. Example 2 shows how tabling helps avoiding infinite loops in resolution. Example 3 shows how answers to *all* subgoals are stored in the table (for all successful subderivations). Finally, Example 4 shows how the negation is handled.

Example 2. Consider the query $p(x)$ for the following program.

$$\begin{aligned} p(x) &\leftarrow x = 0. \\ p(x) &\leftarrow x = 1. \\ p(x) &\leftarrow p(x). \end{aligned}$$

We start with the table consisting of one entry $T[p(x), \mathit{true}] = [\,]$. After a double application of the positive completion rule we obtain two answers $x = 0$ and $x = 1$. Now, an application of the positive extension rule to the clause $p(x) \leftarrow p(x)$ leads to a redundant index in the table; an application of the positive completion rule to this clause gives only redundant answers. Hence, at this point no more resolution steps are possible and the procedure terminates.

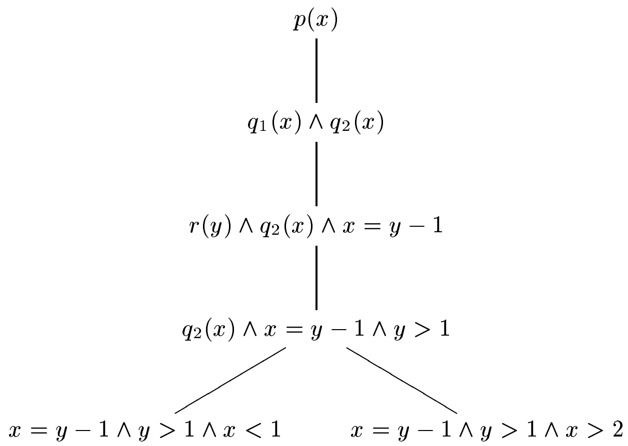

Fig. 3. Computation tree for Example 3

Example 3. Consider the query $p(x)$ for the following program.

$$\begin{aligned} p(x) &\leftarrow q_1(x), q_2(x). \\ q_1(x) &\leftarrow x = y - 1, r(y). \\ r(x) &\leftarrow x > 1. \\ q_2(x) &\leftarrow x < 1. \\ q_2(x) &\leftarrow x > 2. \end{aligned}$$

The derivation tree for this query is shown on Figure 3. In the fourth node of this tree we obtain an answer $x = y - 1 \wedge y > 1$ to the subquery $r(y)$. This answer is normalized (the formula $\exists x.\ x = y - 1 \wedge y > 1$ is simplified by eliminating the existentially quantified x to the formula $y > 1$) and stored in the table: $T[\langle r(y), x = y - 1\rangle] = [y > 1]$.

Note that the constraint $x = y - 1 \wedge y > 1$ is not only an answer to the subquery $r(y)$, but also to the subquery $q_1(x)$. This is covered by the application of the positive completion rule to the query $\langle q_1(x), true\rangle$ and the clause $q_1(x) \leftarrow x = y - 1, r(y)$, using the already stored answer for $\langle r(y), x = y - 1\rangle$. The answer is again normalized (this time the variable y, and not x, is existentially quantified) and stored: $T[\langle q_1(x), true\rangle] = [x > 0]$.

The same happens when we reach leaves of the tree. The two answers that we obtain have to be stored both in the entry for $q_2(x)$ and $p(x)$, thus we obtain $[x > 0 \wedge x < 1, \quad x > 2]$ as the value for $T[\langle p(x), true\rangle]$ and $T[\langle q_2(x), x = y - 1 \wedge y > 1\rangle]$.

Example 4. Consider the program from Example 3 together with a clause $p'(x) \leftarrow \neg p(x)$ and a query $p'(x)$. Since the tree from Figure 2 cannot be extended anymore (it is totally successful in the terminology of Stuckey [36]), we can negate the answers collected for the query $p(x)$ and obtain correct answers for the query $\neg p(x)$ that (after transforming to the disjunctive normal form) we store in the table: $T[\langle \neg p(x), true\rangle] = [x \leq 0, \quad x \geq 1 \wedge x \leq 2]$. (Note: since we always eliminate existential quantifiers, negating constraints does not introduce universal quantification; i.e., we do not have to treat queries with universally quantified variables.)

We next investigate the correctness of the tabulation procedure.

Theorem 2 (Soundness and completeness). *If the tabling procedure terminates then, for every query $\langle l(\boldsymbol{x}), \Phi\rangle$ that occurs as an index in the table T, the entry of T at that index is a list of constraints $[\Phi_1, \ldots, \Phi_n]$ such that*

$$[\![l]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!] = [\![\Phi_1 \vee \ldots \vee \Phi_n]\!].$$

That is, the denotation of the literal p in the perfect model semantics of the program $\mathcal{P}$ intersected with the set of solutions of Φ is the set of solutions of all constraints $\Phi_1, \ldots, \Phi_n$. Equivalently, for every solution $\boldsymbol{v}$ of the constraint Φ then the atom $p(\boldsymbol{v})$ lies (respectively, in the negated case, does not lie) in the perfect model of $\mathcal{P}$ and if and only if $\boldsymbol{v}$ is a solution of one of the constraints Φ_i.

Proof. The proof goes by the induction on the level of the literal $l(\boldsymbol{x})$. For this, we extend the definition of the level mapping from positive atoms to negative literals by putting $level(\neg p) = 1 + level(p)$.

First, recall that $pm(\mathcal{P}) = \bigcup_m M_m$ where $M_{m+1} = T_{\mathcal{P}}^{m+1} \uparrow \omega$, and that for predicates p of level m we have $[\![p]\!]_{pm(\mathcal{P})} \subseteq M_m$.

Suppose that the procedure terminates and that for all literals $l(\boldsymbol{x})$ of level less than or equal to m, where $m \geq -1$, and all constraints Φ such that $\langle l(x), \Phi\rangle$ is an index in the table we have

$$[\![l]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!] = [\![T(l(\boldsymbol{x}), \Phi)]\!].$$

Below we prove that this equality holds for all literals of level $m+1$. Note that the case $m=-1$ covers the induction basis.

Take a table entry $T(l(\boldsymbol{x}),\Phi) = \bigvee_i \Phi_i$ where $l(\boldsymbol{x})$ has level $m+1$. Now we prove the theorem separately for the two cases, where $l(x)$ is positive and negative, respectively.

Case 1. $l(\boldsymbol{x})$ is a negative literal $\neg p(\boldsymbol{x})$.

For the $\subseteq$ direction (completeness of constructive negation), suppose $\boldsymbol{v} \in [\![\neg p]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!]$. Since $\langle \neg p(\boldsymbol{x}),\Phi\rangle$ is an index in the table and the procedure has terminated, we know that by the negative extension rule the table contains an index $\langle p(\boldsymbol{x}),\Psi\rangle$ such that $\Phi \models \Psi$. By the induction hypothesis $[\![p]\!]_{pm(\mathcal{P})} \cap [\![\Psi]\!] = [\![T(p(\boldsymbol{x}),\Psi)]\!]$. Now, $\boldsymbol{v} \in [\![\neg p]\!]_{pm(\mathcal{P})} = \mathcal{D}^n - [\![p]\!]_{pm(\mathcal{P})}$ where n is the arity of p. Therefore, $\boldsymbol{v} \notin [\![p]\!]_{pm(\mathcal{P})}$ and so $\boldsymbol{v} \not\models T(p(\boldsymbol{x}),\Psi)$. Thus, $\boldsymbol{v} \in [\![\Phi \wedge \neg T(p(\boldsymbol{x}),\Psi)]\!] = [\![T(\neg p(\boldsymbol{x}),\Phi)]\!]$.

For the $\supseteq$ direction (soundness of constructive negation), suppose $\boldsymbol{v} \in [\![T(\neg p(\boldsymbol{x}),\Phi)]\!]$. Then the entry $T(\neg p(\boldsymbol{x}),\Phi)$ is nonempty, and by the termination assumption the table must contain an index $\langle p(\boldsymbol{x}),\Psi\rangle$ such that $\Phi \models \Psi$ and $[\![T(\neg p(\boldsymbol{x}),\Phi)]\!] = [\![\Phi \wedge \neg T(p(\boldsymbol{x},\Psi)]\!]$. Then $\boldsymbol{v} \in \Phi$ and $\boldsymbol{v} \notin [\![T(p(\boldsymbol{x},\Psi)]\!]$. By the induction hypothesis, $[\![T(p(\boldsymbol{x}),\Psi)]\!] = [\![p]\!]_{pm(\mathcal{P})} \cap [\![\Psi]\!]$. Since $\Phi \models \Psi$, we have $[\![\Phi]\!] \subseteq [\![\Psi]\!]$ and $[\![p]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!] \subseteq [\![p]\!]_{pm(\mathcal{P})} \cap [\![\Psi]\!]$. Therefore $\boldsymbol{v} \notin [\![p]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!]$. But $\boldsymbol{v} \in [\![\Phi]\!]$, so $\boldsymbol{v} \notin [\![p]\!]_{pm(\mathcal{P})}$. Thus $\boldsymbol{v} \in [\![\neg p]\!]_{pm(\mathcal{P})} \cap [\![\Phi]\!]$.

Case 2. $l(\boldsymbol{x})$ is positive literal $p(\boldsymbol{x})$.

We start with the $\subseteq$ direction (completeness for the positive part). Here the proof goes by the induction on the number of iterations of the $T_{\mathcal{P}}^{m+1}$ operator. We show by an induction on n that for all predicates p of level $m+1$ and all constraints Φ such that $\langle p(\boldsymbol{x}),\Phi\rangle$ is an index in the table T

$$[\![p]\!]_{(T_{\mathcal{P}}^{m+1})^n(M_m)} \cap [\![\Phi]\!] \subseteq [\![T(p(\boldsymbol{x}),\Phi)]\!]$$

For $n=0$ (the induction basis) this is trivially satisfied since $(T_{\mathcal{P}}^{m+1})^0(M_m) = M_m$ and M_m does not contain any atoms of level greater than m.

Now consider the case $n>0$. Let $\boldsymbol{v} \in [\![p]\!]_{(T_{\mathcal{P}}^{m+1})^n(M_m)} \cap [\![\Phi]\!]$. By the definition of the $T_{\mathcal{P}}^{m+1}$ operator, there exists a clause $C \in \mathcal{P}$ of the form $p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge \Phi'$ and tuples of elements $\boldsymbol{v}_i$ from the domain $\mathcal{D}$ such that $\boldsymbol{v},\{\boldsymbol{v}_i\}_i \models \Phi \wedge \Phi'$ and $(T_{\mathcal{P}}^{m+1})^{n-1}(M_m) \models \bigwedge_i l_i(\boldsymbol{v}_i)$. By several applications of the positive extension rules (each for the clause C and some of $l_i(\boldsymbol{y}_i)$) we have that there are indexes $\langle l_i(\boldsymbol{y}_i),\Psi_i\rangle$ in the table T such that $\boldsymbol{v}_i \models \Psi_i$. By the induction hypothesis $\boldsymbol{v}_i \in [\![T(l_i(\boldsymbol{y}_i),\Psi_i)]\!]$. Hence, there exist constraints $\Phi_i \in T(l_i(\boldsymbol{y}_i),\Psi_i)$ such that $v_i \models \Phi_i$. Then $\boldsymbol{v},\{\boldsymbol{v}_i\}_i \models \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi'$, and thus $\boldsymbol{v} \models \exists_{-\boldsymbol{x}} \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi'$. Now an application of the positive completion rule shows that $\boldsymbol{v} \in [\![T(p(\boldsymbol{x}),\Phi)]\!]$.

The $\supseteq$ direction (soundness of the positive part) is proved in a quite similar way, by induction on the number of applications of the positive completion rule to the atoms of level $m+1$.

We prove that for all $n \geq 0$

$$[\![p]\!]_{pm(\mathcal{P}))} \cap [\![\Phi]\!] \supseteq [\![T_{(n)}(p(\boldsymbol{x}),\Phi)]\!]$$

where $T_{(n)}$ is the table T after n applications of the positive completion rule to a predicate of level $m+1$.

For $n = 0$ this is trivially satisfied, since $T_{(0)}(p(\boldsymbol{x}), \Phi)$ is empty.

For $n > 0$, suppose $\Psi \in T_{(n)}(p(\boldsymbol{x}), \Phi)$ and $\boldsymbol{v} \in [\![\Psi]\!]$. Since Φ is a conjunct in Ψ, we have $\boldsymbol{v} \in [\![\Phi]\!]$, so we only have to prove that $\boldsymbol{v} \in [\![p]\!]_{pm(\mathcal{P}))}$. If $\Psi \in T_{(n-1)}(p(\boldsymbol{x}), \Phi)$ then we are done by the induction hypothesis, so suppose $\Psi \notin T_{(n-1)}(p(\boldsymbol{x}), \Phi)$. Then there exists a clause $p(\boldsymbol{x}) \leftarrow \bigwedge_i l_i(\boldsymbol{y}_i) \wedge \Phi'$ and constraints Φ_i, Ψ_i such that $\Phi_i \in T_{(n-1)}(l_i(\boldsymbol{y}_i), \Psi_i)$ and $\Psi = \exists_{-\boldsymbol{x}} \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi'$. Since Ψ is satisfiable, we can find $\boldsymbol{v}_i$ such that $\boldsymbol{v}, \{\boldsymbol{v}_i\}_i \models \bigwedge_i \Phi_i \wedge \Phi \wedge \Phi'$. By the induction hypothesis (since all l_i are of level $\leq m+1$), $M_{m+1} \models l_i(\boldsymbol{v}_i)$. Therefore $p(\boldsymbol{v}) \in T_{\mathcal{P}}^{m+1}(M_{m+1}) \subseteq pm(\mathcal{P})$. □

5 Related Work, Future Work

The notion of forward analysis for infinite-state systems is different from the notion of 'symbolic forward state traversal' in [25]. This is already clear by the fact that the property $\mathsf{EF}(p \wedge \mathsf{EF}(q) \wedge \mathsf{EF}(r))$ is not computable by symbolic forward state traversal [25] (but since it is an Sμ property, it is computable by forward analysis). In either sense, a forward procedure is a least-fixpoint iteration involving the direct-successor operator *post* applied to a constraint representation of a set of states. In [25], however, a constraint is viewed monolithically (this accounts for the setting where a finite set of states is represented by a BDD), whereas our procedure can operate on each single disjunct in a disjunctive constraint.

The logic-programming approach to verification is an active topic of research; see e.g. [10, 16, 20–22, 30, 31, 33]. The work that is directly related to ours is [33] and [30], where the authors use logic programming and tabulation to compute CTL properties of *finite*-state systems. One may characterize the difference between that work and ours by stating that we trade the loss of expressiveness of the considered temporal logic (Sμ vs. CTL) with the gain in generality of the systems to be verified (infinite vs. finite).

In [22], Gottlob, Grädel and Veith consider two linear-time, P-complete versions of Datalog, Datalog LIT and its Extension Datalog LITE, and translate formulas of the Alternation-Free μ-calculus AFμ into Datalog LITE in such a way that backward model checking amounts to bottom-up query evaluation. No top-down query evalution procedure is known for Datalog LITE over infinite models. Sμ is exactly the fragment of AFμ such that the translation of [22] falls into Datalog LIT; their translation then becomes the one presented in Section 3.

In a quite extensive literature on the subject, we found many tabulation procedures that are superficially related (see e.g. [8, 11, 12, 17, 19, 34, 35, 38, 37]); since we did not not find one that fits exactly, we felt that we had to present it here.

Two seemingly unrelated algorithms for detecting convergence of timed automata have been proposed. One is based on backward analysis [24], the other on forward analysis [39]. Once we have observed that convergence can be expressed in Sμ, in fact already EF-Logic (for every timed automaton $\mathcal{A}$, one can construct a timed automaton $\mathcal{A}'$ (in linear time) such that $\mathcal{A}$ is convergent if and only if the EF-Logic-formula $\mathsf{EF}\neg\mathsf{EF}(y > 1)$ is true for the automaton $\mathcal{A}'$), it is not difficult to see that the two algorithms are ad-hoc solutions for an instance of the problem of computing a perfect model of a stratified constraint logic program, and that they correspond to the bottom-up resp. top-down approach.

We leave it for future work to make precise our intuition that Sμ captures the boundary in expressiveness of temporal logic properties for which constraint-based forward methods exist. Also, it is open whether the equality STL=CTL$\cap$Sμ holds wrt. expressiveness (i.e. not only syntactically via the embedding of CTL and STL into AFμ).

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real time. *Information and Computation*, 104(2):2–34, 1993.
2. R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.
3. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.
4. R. Alur and T. A. Henzinger. Modularity for timed and hybrid systems. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR'97: Concurrency Theory*, volume 1243 of *LNCS*, pages 74–88. Springer-Verlag, 1997.
5. R. Alur and T. A. Henzinger. Computer-aided verification: An introduction to model building and model checking for concurrent systems, 1999. Book in preparation.
6. K. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–214. Morgan Kaufmann, 1988.
7. J. Bengtsson, K. G. Larsen, F. Larsson, P. Petersson, and W. Yi. Uppaal in 1995. In T. Margaria and B. Steffen, editors, *TACAS*, LNCS 1055, pages 431–434. Springer-Verlag, 1996.
8. D. Chan. Constructive negation based on the completed database. In R. A. Kowalski and K. A. Bowen, editors, *the fifth International Symposium on Logic Programming*, pages 111–125. MIT Press, 1988.
9. W. Charatonik, D. McAllester, D. Niwinski, A.Podelski, and I. Walukiewicz. The Horn mu-calculus. In V. Pratt, editor, *The 13th IEEE Annual Symposium on Logic in Computer Science*, 1998.
10. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 358–375, Lisbon, Portugal, March-April 1998. Springer-Verlag.
11. W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *JACM*, 43(1):20–74, 1996.
12. P. Codognet. A tabulation method for constraint logic programming. Proceedings of INAP'95, Industrial Applications of Prolog, Tokyo, Japan, 1995.

13. H. Comon and Y. Jurski. Multiple Counters Automata, Safety Analysis, and Presburger Arithmetics. In A. J. Hu and M. Y. Vardi, editors, *CAV'98: Computer Aided Verification*, volume 1427 of *LNCS*, pages 268–279. Springer-Verlag, 1998.
14. B. Cui, Y. Dong, X. Du, K. N. Kumar, C. R. Ramakrishnan, I. V. Ramakrishnan, A. Roychoudhury, S. A. Smolka, and D. S. Warren. Logic programming and model checking. In *PLAP/ALP98*, volume 1490 of *LNCS*, pages 1–20. Springer-Verlag, 1998.
15. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In B. Steffen, editor, *TACAS98: Tools and Algorithms for the Construction of Systems*, LNCS 1384, pages 313–329. Springer-Verlag, March/April 1998.
16. G. Delzanno and A. Podelski. Model Checking in CLP. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239, Amsterdam, The Netherlands, January 1999. Springer-Verlag.
17. X. Du, C. R. Ramakrishnan, and S. A. Smolka. Tabled resolution + constraints: A recipe for model checking real-time systems. In *Proceedings of RTSS 2000*, 2000.
18. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.
19. F. Fages. Constructive negation by prunning. *Journal of Logic Programming*, 32(8):85–118, August 1997.
20. L. Fribourg and M. V. Peixoto. Concurrent constraint automata. In D. Miller, editor, *Logic Programming - Proceedings of the 1993 International Symposium*, page 656, Vancouver, Canada, 1993. The MIT Press.
21. L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. In J. P. Gallagher, editor, *LOPSTR'96: Logic Based Program Synthesis and Transformation*, volume 1207 of *LNCS*, pages 20–37. Springer-Verlag, 1996.
22. G. Gottlob, E. Grädel, and H. Veith. Linear Time Datalog for Branching Time Logic. In J. Minker, editor, *Logic-Based Artificial Intelligence*, chapter 19. Kluwer, 2000.
23. G. Gupta and E. Pontelli. A constraint-based approach for the specification and verification of real-time systems. In K.-J. Lin, editor, *IEEE Real-Time Systems Symposium*, pages 230–239. IEEE Press, 1997.
24. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. Special issue for LICS 92.
25. T. A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. In A. J. Hu and M. Y. Vardi, editors, *CAV'98: Computer-aided Verification*, LNCS 1427, pages 195–206. Springer-Verlag, 1998.
26. Y. Kesten, Z. Manna, and A. Pnueli. Verifying clocked transition systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *LNCS*, pages 13–40. Springer-Verlag, 1996.
27. K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.
28. S. Mukhopadhyay and A. Podelski. Beyond region graphs: Symbolic forward analysis of timed automata. In C. Pandurangan, V. Raman, and R. Ramanujam, editors, *19th International Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 233–245, December 1999.

29. S. Mukhopadhyay and A. Podelski. Model checking for timed logic processes. In *Proceedings of the First International Conference on Computational Logic (CL)*, volume 1861 of *LNAI*, pages 598–612. Springer, 2000.
30. U. Nilsson and J. Lübke. Constraint logic programming for local and symbolic model checking. In *Proceedings of the First International Conference on Computational Logic (CL)*, volume 1861 of *LNAI*, pages 384–398. Springer, 2000.
31. G. Pemmasani, C. Ramakrishnan, and I. Ramakrishnan. Efficient model checking of real time systems using tabled logic programming and constraints. In *This proceedings*, 2002.
32. T. Przymusinski. On the semantics of stratified deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.
33. Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. Smolka, T. Swift, and D. Warren. Efficient model checking using tabled resolution. In *Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*. Springer-Verlag, June 1997.
34. H. Seki and H. Ito. A query evaluation method for stratified programs under the extended cwa. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the fifth International Conference and Symposium on Logic Programming*, pages 195–211. MIT Press, 1988.
35. P. Stuckey. Negation for constraint logic programming. *Information and Computation*, 118(1), 1995.
36. P. J. Stuckey. Constructive negation for constraint logic programming. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 328–339. IEEE Computer Society Press, 1991.
37. H. Tamaki and T. Sato. OLD resolution with tabulation. In E. Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, volume 225 of *LNCS*, pages 84–98, London, 1986. Springer-Verlag.
38. D. Toman. Top-down beats bottom-up for constraint extensions of datalog. In J. Lloyd, editor, *Proceedings of the International Symposium on Logic Programming*, pages 98–114, Cambridge, Dec. 4–7 1995. MIT Press.
39. S. Tripakis. Verifying progress in timed systems. In J. P. Katoen, editor, *Formal Methods in Real time and Probablistic Systems: 5th International AMAST Workshop (ARTS99)*, volume 1601 of *LNCS*, pages 299–314. Springer-Verlag, 1999.