Constraint-Based Abstraction of a Model Checker for Infinite State Systems*

Gourinath Banda[1] and John P. Gallagher[1,2]

[1] Roskilde University, Denmark
[2] IMDEA Software, Madrid
Email: {gnbanda,jpg}@ruc.dk

Abstract. Abstract interpretation-based model checking provides an approach to verifying properties of infinite-state systems. In practice, most previous work on abstract model checking is either restricted to verifying universal properties, or develops special techniques for temporal logics such as modal transition systems or other dual transition systems. By contrast we apply completely standard techniques for constructing abstract interpretations to the abstraction of a CTL semantic function, without restricting the kind of properties that can be verified. Furthermore we show that this leads directly to implementation of abstract model checking algorithms for abstract domains based on constraints, making use of an SMT solver.

1 Introduction

Model Checking is a widely used technique for verifying properties of reactive systems expressed as formulas in a temporal logic, but is restricted to finite-state systems. Abstraction is an effective technique for handling infinite state spaces, where a finite or infinite number of *original states* are collectively represented with a single *abstract state*.

The theory of *abstract interpretation* formalises abstraction techniques. In abstract interpretation-based analyses, an *abstract domain* is first constructed and then a *Galois connection* between the original (or *concrete*) domain and the abstract domain is defined. Computations over the concrete domain are abstractly interpreted over the abstract domain. Due to the Galois connection, the result from abstract computation will always be an *over approximation* of the actual result, had the actual analysis been possible.

The present work is part of an attempt to develop a uniform CLP-based formal modelling and verification framework for verifying infinite state reactive systems. The modelling part of this framework was covered in [2] where it is shown (i) how to model linear hybrid automata (LHA) specifications in CLP; (ii) how standard program transformation tools of CLP can be applied to extract the underlying state transition system semantics; and (iii) how to abstract the (infinite) reachable state space with the domain of linear constraints. In this extended abstract, we show the verification part of the framework in which abstract interpretation and model checking are integrated.

* Work partly supported by the Danish Natural Science Research Council project *SAFT: Static Analysis Using Finite Tree Automata*.

The contributions of this work are threefold. First, we present the definition of a CTL-semantics function in a suitable form, using only monotonic functions and fixpoint operators. Second, we apply the standard abstract interpretation framework to get a precise abstraction of the CTL-semantics function. We do not construct an abstract transition system, which turns out to be an unnecessary restriction. Finally, we show how a constraint-based abstraction can be directly implemented from the abstract semantic function and show how *satisfiability modulo theories* (SMT)-technology can be exploited to improve the performance of our abstract model checker.

The structure of this paper is as follows. Section 2 introduces the syntax and semantics of CTL, and outlines the theory of abstract interpretation. Section 3 describes abstract model checking, that is, abstract interpretation of the CTL semantics function. Section 4 shows how the framework can be applied in practice to an abstraction based on linear constraints. Section 6 gives some experimental results, and we conclude in Section 8.

2 CTL, Model Checking and Abstract Interpretation

CTL is a formal language used to specify temporal properties. A well formed formula in CTL is constructed from one or more atomic propositions and eight CTL-operators. The syntax of CTL is given below.

Definition 1 (CTL Syntax). *The set of CTL formulas* ϕ *in* negation normal form *is inductively defined by the following grammar:*

$$\begin{aligned}\phi ::= & \mathit{true} \mid p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \mid AX\phi \mid EX\phi \mid AF\phi \\ & \mid EF\phi \mid AG\phi \mid EG\phi \mid AU[\phi_1,\phi_2] \mid EU[\phi_1,\phi_2] \mid AR[\phi_1,\phi_2] \mid ER[\phi_1,\phi_2]\end{aligned}$$

where p ranges over a set of atomic formulas $\mathcal{P}$.

A CTL-formula is in negation normal form (NNF) if and only if the negations are placed in front of the atomic propositions. Any formula not in NNF can be transformed into NNF by moving negations inwards using equivalence-preserving transformations.. The *unary operators* AF, AG, AX and the binary operators AU, AR are called *universal CTL operators*; while the unary EF, EG, EX and binary ER, EU operators are called *existential CTL operators*.

2.1 CTL Semantics

The semantics of CTL formulas is defined with respect to a Kripke structure, which is a state transition system whose states are labelled with atomic propositions that are true in that state.

Definition 2 (Kripke structure). *A Kripke structure is a tuple* $\langle S, \Delta, I, L, \mathcal{P}\rangle$ *where S is the set of states,* $\Delta \subseteq S \times S$ *is the transition relation,* $I \subseteq S$ *is the set of initial states,* $\mathcal{P}$ *is the set of propositions and* $L : S \rightarrow 2^{\mathcal{P}}$ *is the labelling function which returns the set of propositions that are true in each state. The set of atomic propositions is closed under negation.*

Given the *Kripke structure* $\langle S, \Delta, I, L, \mathcal{P} \rangle$, the meaning of a formula is the set of states in S where the formula holds; this is itself an abstraction of a more detailed trace-based semantics [9]. We define a function $[\![\cdot]\!] : CTL \to 2^S$ that returns the set of states where the formula holds. This function is called the*CTL-semantics function*.

Definition 3 (CTL-semantics function). *Given a Kripke structure $K = \langle S, \Delta, I, L, \mathcal{P} \rangle$, the semantic function $[\![\cdot]\!] : CTL \to 2^S$ is defined as follows.*

$$
\begin{array}{ll|ll}
[\![true]\!] & = S & [\![false]\!] & = \emptyset \\
[\![p]\!] & = \mathsf{states}(p) & [\![\neg p]\!] & = \mathsf{states}(\neg p) \\
[\![EX\phi]\!] & = \mathsf{pred}_\exists([\![\phi]\!]) & [\![\phi_1 \vee \phi_2]\!] & = [\![\phi_1]\!] \cup [\![\phi_2]\!] \\
[\![AX\phi]\!] & = \mathsf{pred}_\forall([\![\phi]\!]) & [\![\phi_1 \wedge \phi_2]\!] & = [\![\phi_1]\!] \cap [\![\phi_2]\!] \\
[\![EF\phi]\!] & = \mu Z.([\![\phi]\!] \cup \mathsf{pred}_\exists(Z)) & [\![ER[\phi_1, \phi_2]]\!] & = \nu Z.([\![\phi_2]\!] \cap ([\![\phi_1]\!] \cup \mathsf{pred}_\exists(Z)) \\
[\![AF\phi]\!] & = \mu Z.([\![\phi]\!] \cup \mathsf{pred}_\forall(Z)) & [\![AU[\phi_1, \phi_2]]\!] & = \mu Z.([\![\phi_2]\!] \cup ([\![\phi_1]\!] \cap \mathsf{pred}_\forall(Z)) \\
[\![AG\phi]\!] & = \nu Z.([\![\phi]\!] \cap \mathsf{pred}_\forall(Z)) & [\![EU[\phi_1, \phi_2]]\!] & = \mu Z.([\![\phi_2]\!] \cup ([\![\phi_1]\!] \cap \mathsf{pred}_\exists(Z)) \\
[\![EG\phi]\!] & = \nu Z.([\![\phi]\!] \cap \mathsf{pred}_\exists(Z)) & [\![AR[\phi_1, \phi_2]]\!] & = \nu Z.([\![\phi_2]\!] \cap ([\![\phi_1]\!] \cup \mathsf{pred}_\forall(Z))
\end{array}
$$

This semantics function makes use of three subsidiary functions $\mathsf{pred}_\exists : 2^S \to 2^S$, $\mathsf{pred}_\forall : 2^S \to 2^S$ and $\mathsf{states} : \mathcal{P} \to 2^S$ called the *existential predecessor function*, the *universal predecessor function* and *allocating function* respectively. These three functions are specific for a given Kripke structure K and are monotonic.

Definition 4. *Given a Kripke structure $K = \langle S, \Delta, I, L, \mathcal{P} \rangle$ we define functions $\mathsf{pred}_\exists : 2^S \to 2^S$, $\mathsf{pred}_\forall : 2^S \to 2^S$ and $\mathsf{states} : \mathcal{P} \to 2^S$ as follows.*

- $\mathsf{pred}_\exists(S') = \{s \mid \exists s' \in S' : (s, s') \in \Delta\}$ *returns the set of states having at least one of their successors in the set $S' \subseteq S$;*
- $\mathsf{pred}_\forall(S') = \mathsf{pred}_\exists(S') \backslash \mathsf{pred}_\exists(\mathsf{compl}(S'))$ *returns the set of states all of whose successors are in the set $S' \subseteq S$; the function $\mathsf{compl}(X) = S \setminus X$.*
- $\mathsf{states}(p) = \{s \in S \mid p \in L(s)\}$ *returns the set of states where $p \in \mathcal{P}$ holds.*

In the CTL semantic definition, $\mu Z.(F(Z))$ (resp. $\nu Z.(F(Z))$) stands for the least fixed point (resp. greatest fixed point) of the function $\lambda Z.F(Z)$. The Knaster-Tarski fixed point theorem [28] guarantees the existence of least and greatest fixed points for a monotonic function on a complete lattice. All the expressions $F(Z)$ occurring in $\mu Z.(F(Z))$ and $\nu Z.(F(Z))$ in the CTL semantic functions are functions $2^S \to 2^S$ on the complete lattice $\langle 2^S, \subseteq, \cup, \cap, S, \emptyset \rangle$. They are constructed with *monotonic operators* ($\cup$,$\cap$) and monotonic functions ($\mathsf{pred}_\exists, \mathsf{pred}_\forall$). Thus the CTL semantics function is well-defined.

2.2 Model Checking

Model checking is based on checking that the Kripke structure $K = \langle S, \Delta, I, L, \mathcal{P} \rangle$ possesses a property ϕ, written $K \models \phi$. This is defined to be true iff $I \subseteq [\![\phi]\!]$, or equivalently, that $I \cap [\![\neg\phi]\!] = \emptyset$. (Note that $\neg\phi$ should be converted to negation normal form). Thus model-checking requires implementing the CTL-semantics function which in essence is a *fixed point computation*. When the state-space S is finite the greatest and least fixed

point expressions can be evaluated as the limits of Kleene sequences. But when S is infinite, the fixed point computations might not terminate and hence the model checking of infinite state systems becomes undecidable. In this case we try to approximate $[\![.]\!]$ using the theory of *abstract interpretation*.

2.3 Abstract Interpretation

In abstract interpretation we replace the "concrete" semantic function by an abstract semantic function, developed systematically from the concrete semantics with respect to a Galois connection. We present the formal framework briefly.

Definition 5 (Galois Connection). $\langle L, \sqsubseteq_L \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq_M \rangle$ *is a* Galois Connection *between the lattices* $\langle L, \sqsubseteq_L \rangle$ *and* $\langle M, \sqsubseteq_M \rangle$ *if and only if* $\alpha : L \to M$ *and* $\gamma : M \to L$ *are monotonic and* $\forall l \in L, m \in M, \alpha(l) \sqsubseteq_M m \leftrightarrow l \sqsubseteq_L \gamma(m)$.

In abstract interpretation, $\langle L, \sqsubseteq_L \rangle$ and $\langle M, \sqsubseteq_M \rangle$ are the concrete and abstract semantic domains respectively. Given a Galois connection $\langle L, \sqsubseteq_L \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq_M \rangle$ and a monotonic concrete semantics function $f : L \to L$, then we define an abstract semantic function $f^\sharp : M \to M$ such that for all $m \in M$, $(\alpha \circ f \circ \gamma)(m) \sqsubseteq_M f^\sharp(m)$. Furthermore it can be shown that $\mathsf{lfp}(f) \sqsubseteq_L \gamma(\mathsf{lfp}(f^\sharp))$ and that $\mathsf{gfp}(f) \sqsubseteq_L \gamma(\mathsf{gfp}(f^\sharp))$.

Thus the abstract function $f^\sharp$ can be used to compute over-approximations of f, which can be interpreted using the γ function. The case where the abstract semantic function is defined as $f^\sharp = (\alpha \circ f \circ \gamma)$ gives the most precise approximation.

If M is a finite-height lattice then the non-terminating fixed point computations of $\mathsf{lfp}(f)$ and $\mathsf{gfp}(f)$ over L are approximated with a terminating fixed point computation over the finite lattice M.

We next apply this general framework to abstraction of the CTL semantic function, and illustrate with a specific abstraction in Section 4.

3 Abstract Interpretation of the CTL-Semantic function

In this section we consider abstractions based on Galois connections of the form $\langle 2^S, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq \rangle$, where the abstract domain 2^A consists of sets of abstract states. In fact the abstract domain could be any lattice but for the purposes of this paper we consider such state-based abstractions, which will be further discussed in Section 4.

Definition 6. *Let* $\mathsf{pred}_\exists : 2^S \to 2^S$, $\mathsf{pred}_\forall : 2^S \to 2^S$, *and* $\mathsf{states} : \mathcal{P} \to 2^S$ *be the functions defined in Definition 4 and used in the CTL semantic function. Given a Galois connection* $\langle 2^S, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq \rangle$, *we define* $\mathsf{apred}_\exists : 2^A \to 2^A$, $\mathsf{apred}_\forall : 2^A \to 2^A$ *and* $\mathsf{astates} : \mathcal{P} \to 2^A$ *as*

$$\mathsf{apred}_\exists = \alpha \circ \mathsf{pred}_\exists \circ \gamma \qquad \mathsf{apred}_\forall = \alpha \circ \mathsf{pred}_\forall \circ \gamma \qquad \mathsf{astates} = \alpha \circ \mathsf{states}$$

It follows directly from the properties of Galois connections that for all $S' \subseteq S$, $\alpha(\mathsf{pred}_\exists(S')) \subseteq \mathsf{apred}_\exists(\alpha(S'))$ and $\alpha(\mathsf{pred}_\forall(S')) \subseteq \mathsf{apred}_\forall(\alpha(S'))$.

Definition 7 (Abstract *CTL* semantics function). *Given a Galois connection* $\langle 2^S, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq \rangle$*, the abstract CTL semantic function* $[\![\cdot]\!]^a : CTL \to 2^A$ *is defined as follows.*

$$
\begin{array}{llll}
[\![true]\!]^a & = A & [\![false]\!]^a & = \emptyset \\
[\![p]\!]^a & = \mathsf{astates}(p) & [\![\neg p]\!]^a & = \mathsf{astates}(\neg p) \\
[\![EX\phi]\!]^a & = \mathsf{apred}_\exists([\![\phi]\!]^a) & [\![\phi_1 \vee \phi_2]\!]^a & = [\![\phi_1]\!]^a \cup [\![\phi_2]\!]^a \\
[\![AX\phi]\!]^a & = \mathsf{apred}_\forall([\![\phi]\!]^a) & [\![\phi_1 \wedge \phi_2]\!]^a & = [\![\phi_1]\!]^a \cap [\![\phi_2]\!]^a \\
[\![EF\phi]\!]^a & = \mu Z.([\![\phi]\!]^a \cup \mathsf{apred}_\exists(Z)) & [\![ER[\phi_1,\phi_2]]\!]^a & = \nu Z.([\![\phi_2]\!]^a \cap ([\![\phi_1]\!]^a \cup \mathsf{apred}_\exists(Z)) \\
[\![AF\phi]\!]^a & = \mu Z.([\![\phi]\!]^a \cup \mathsf{apred}_\forall(Z)) & [\![AU[\phi_1,\phi_2]]\!]^a & = \mu Z.([\![\phi_2]\!]^a \cup ([\![\phi_1]\!]^a \cap \mathsf{apred}_\forall(Z)) \\
[\![AG\phi]\!]^a & = \nu Z.([\![\phi]\!]^a \cap \mathsf{apred}_\forall(Z)) & [\![EU[\phi_1,\phi_2]]\!]^a & = \mu Z.([\![\phi_2]\!]^a \cup ([\![\phi_1]\!]^a \cap \mathsf{apred}_\exists(Z)) \\
[\![EG\phi]\!]^a & = \nu Z.([\![\phi]\!]^a \cap \mathsf{apred}_\exists(Z)) & [\![AR[\phi_1,\phi_2]]\!]^a & = \nu Z.([\![\phi_2]\!]^a \cap ([\![\phi_1]\!]^a \cup \mathsf{apred}_\forall(Z))
\end{array}
$$

Since all the operators appearing in the abstract CTL-semantic are monotonic, the fixpoint expressions and hence the abstract semantic function is well defined. The following soundness theorem is the basis of our abstract model checking approach.

Theorem 1 (Safety of Abstract CTL Semantics). *Let* $K = \langle S, \Delta, I, L, \mathcal{P} \rangle$ *be a Kripke structure,* $\langle 2^S, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq \rangle$ *be a Galois connection and* ϕ *any CTL-formula in negation normal form. Then* $\alpha([\![\phi]\!]) \subseteq [\![\phi]\!]^a$ *and* $\gamma([\![\phi]\!]^a) \supseteq [\![\phi]\!]$.

The proof follows from the fact that α is a join-morphism: that is, that $\alpha(S_1 \cup S_2) = \alpha(S_1) \cup \alpha(S_2)$ and the fact that $\alpha(S_1 \cap S_2) \subseteq \alpha(S_1) \cap \alpha(S_2)$.

This theorem provides us with a sound abstract model checking procedure for any CTL formula ϕ. As noted previously, $K \models \phi$ iff $[\![\neg\phi]\!] \cap I = \emptyset$ (where $\neg\phi$ is converted to negation normal form). It follows from Theorem 1 that this follows if $\gamma([\![\neg\phi]\!]^a) \cap I = \emptyset$. Of course, if $\gamma([\![\neg\phi]\!]^a) \cap I \supseteq \emptyset$ nothing can be concluded.

4 Abstract Model Checking in Constraint-based Domains

The abstract semantics given in Section 3 is not always implementable in practice for a given Galois connection $\langle 2^S, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq \rangle$. In particular, the function γ yields a value in the concrete domain, which is typically an infinite object such as an infinite set. Thus evaluating the functions $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$ and $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$ might not be feasible.

In this section we show that the construction is implementable for transition systems and abstract domains expressed using linear constraints.

4.1 Constraint Representation of Transition Systems

We consider the set of linear arithmetic constraints (hereafter simply called constraints) over the real numbers.

$$c ::= t_1 \leq t_2 \mid t_1 < t_2 \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \mid \neg c$$

where t_1, t_2 are linear arithmetic terms built from real constants, variables and the operators $+$, $*$ and $-$. The constraint $t_1 = t_2$ is an abbreviation for $t_1 \leq t_2 \wedge t_2 \leq t_1$. Note

that $\neg(t_1 \leq t_2) \equiv t_2 < t_2$ and $\neg(t_1 < t_2) \equiv t_2 \leq t_2$, and so the negation symbol $\neg$ can be eliminated from constraints if desired by moving negations inwards by Boolean transformations and then applying this equivalence.

A constraint is *satisfied* by an assignment of real numbers to its variables if the constraint evaluates to *true* under this assignment, and is *satisfiable* if there exists some assignment that satisfies it. A constraint can be identified with the set of assignments that satisfy it. Thus a constraint over n real variable represents a set of points in R^n.

A constraint can be projected onto a subset of its variables. Denote by $\mathsf{proj}_V(c)$ the projection of c onto the set of variables X.

Let us consider a transition system defined over the state-space R^n. Let $\bar{x}, \bar{x}_1, \bar{x}_2$ etc. represent n-tuples of distinct variables, and $\bar{r}, \bar{r}_1, \bar{r}_2$ etc. represent tuples of real numbers. Let $\bar{x}/\bar{r}$ represent the assignment of values $\bar{r}$ to the respective variables $\bar{x}$. We consider transition systems in which the transitions can be represented as a finite set of *transition rules* of the form $\bar{x}_1 \xrightarrow{c(\bar{x}_1,\bar{x}_2)} \bar{x}_2$. This represents the set of all transitions from state $\bar{r}_1$ to state $\bar{r}_2$ in which the constraint $c(\bar{x}_1,\bar{x}_2)$ is satisfied by the assignment $\bar{x}_1/\bar{r}_1, \bar{x}_2/\bar{r}_2$. Such transition systems can be used to model real-time control systems [18, 2].

4.2 Computation of the CTL semantic function using constraints

A constraint representation of a transition system allows a constraint solver to be used to compute the functions $\mathsf{pred}_\exists$, $\mathsf{pred}_\forall$ and states in the CTL semantics. Let T be a finite set of transition rules. Let $c'(\bar{y})$ be a constraint over variables $\bar{y}$. It is assumed that the set of propositions in the Kripke structure used in the semantics is the set of linear constraints.

$$\mathsf{pred}_\exists(c'(\bar{y})) = \bigvee\{\mathsf{proj}_{\bar{x}}(c'(\bar{y}) \wedge c(\bar{x},\bar{y})) \mid \bar{x} \xrightarrow{c(\bar{x},\bar{x})} \bar{y} \in T\}$$
$$\mathsf{pred}_\forall(c'(\bar{y})) = \mathsf{pred}_\exists(c'(\bar{y})) \wedge \neg(\mathsf{pred}_\exists(\neg c'(\bar{y})))$$
$$\mathsf{states}(p) = p$$

In the definition of states we use p both as the proposition (the argument of states) and as a set of points (the result).

4.3 Abstract Domains Based on a Disjoint State-Space Partition

Suppose we have a transition system with n state variables; we take as the *concrete domain* the complete lattice $\langle 2^C, \subseteq\rangle$ where $C \subseteq 2^{R^n}$ is some nonempty, possibly infinite set of n-tuples including all the reachable states of the system.

We build an abstraction of the state space based on a disjoint partition of C say $A = \{d_1, \ldots, d_k\}$ such that $\bigcup A = C$. Such a partition can itself be constructed by an abstract interpretation of the transition relation [2]. Define a representation function $\beta : C \to 2^A$, such that $\beta(\bar{x}) = \{d \in A \mid x \in d\}$. We extend the representation function to sets of points, obtaining the abstraction function $\alpha; 2^C \to 2^A$ given by $\alpha(S) = \bigcup\{\beta(x) \mid x \in S\}$. Define the concretisation function $\gamma : 2^A \to 2^C$, as $\gamma(V) = \{x \in C \mid \beta(x) \subseteq V\}$. As shown in [24, 7], $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \subseteq)$ is a Galois connection. Because the partition A is disjoint the value of $\beta(x)$ is a singleton for all x, and the γ function can be written as $\gamma(V) = \bigcup\{\gamma(\{d\}) \mid d \in V\}$.

4.4 Representation of Abstraction Using Constraints

A constraint can be identified with the set of points that satisfies it. Suppose that each element d of the partition A is representable as a linear constraint c_d over the variables $x_1, \ldots x_n$. The β function can be rewritten as $\beta(x) = \{d \mid x \text{ satisfies } c_d\}$. Assuming that we apply α to sets of points represented by a linear constraint over $x_1, \ldots x_n$, we can rewrite the α and γ functions as follows.

$$\alpha(c) = \{d \mid \mathsf{SAT}(c_d \wedge c)\} \qquad \gamma(V) = \bigvee\{c_d \mid d \in V\}$$

4.5 Computation of α and γ functions using constraint solvers

The constraint formulations of the α and γ functions allows them to be effectively computed. The expression $\mathsf{SAT}(c_d \wedge c)$ occurring in the α function means "$(c_d \wedge c)$ is satisfiable" and can be checked by an SMT solver. In our experiments we use the SMT solver Yices [13]. The γ function simply collects a disjunction of the constraints associated with the given set of partitions; no solver is required.

4.6 Implementation of constraint-based abstract semantics

Combining the constraint-based evaluation of the functions $\mathsf{pred}_\exists$ and $\mathsf{pred}_\forall$ with the constraint-based evaluation of the α and γ functions gives us (in principle) a method of computing the abstract semantic counterparts of $\mathsf{pred}_\exists$ and $\mathsf{pred}_\forall$, namely $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$ and $(\alpha \circ \mathsf{pred}_\forall \circ \gamma)$. This gives us a sound abstract semantics for CTL as discussed previously. The question we now address is the feasibility of this approach. Taken naively, the evaluation of these constraint-based functions (in particular $\mathsf{pred}_\forall$) does not scale up. We now show how we can transform these definitions to a form which can be computed much more efficiently, with the help of an SMT solver.

Consider the evaluation of $(\alpha \circ \mathsf{pred}_\forall \circ \gamma)(V)$ where $V \in 2^A$ is a set of disjoint partitions represented by constraints.

$$\begin{aligned}(\alpha \circ \mathsf{pred}_\forall \circ \gamma)(V) &= (\alpha \circ \mathsf{pred}_\forall)(\bigvee\{c_d \mid d \in V\}) \\ &= \alpha(\mathsf{pred}_\exists(\bigvee\{c_d \mid d \in V\}) \wedge \neg(\mathsf{pred}_\exists(\neg(\bigvee\{c_d \mid d \in V\})))) \\ &= \alpha(\mathsf{pred}_\exists(\bigvee\{c_d \mid d \in V\}) \wedge \neg(\mathsf{pred}_\exists(\bigvee\{c_d \in A \setminus V\}))\end{aligned}$$

In the last step, we use the equivalence $\neg(\bigvee\{c_d \mid d \in V\}) \leftrightarrow \bigvee\{c_d \in A \setminus V\})$, which is justified since the abstract domain A is a disjoint partition of the concrete domain; thus $A \setminus V$ represents the negation of V restricted to the state space of the system. The computation of $\mathsf{pred}_\exists(\bigvee\{c_d \in A \setminus V\})$ is much easier to compute (with available tools) than $\mathsf{pred}_\exists(\neg(\bigvee\{c_d \mid d \in V\}))$. The latter requires the projection operations proj to be applied to complex expressions of the form $\mathsf{proj}_{\bar{x}}(\neg(c_1(\bar{y}) \vee \cdots \vee c_k(\bar{y})) \wedge c(\bar{x}, \bar{y}))$, which involves expanding the expression (to d.n.f. for example); by contrast the former requires evaluation of simpler expressions of the form $\mathsf{proj}_{\bar{x}}(c_d(\bar{y}) \wedge c(\bar{x}, \bar{y}))$.

4.7 Further Optimisation by Pre-Computing Predecessor Constraints

We now show that we can improve the computation of the abstract function $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$. Let $\{c_i\}$ be a set of constraints, each of which represents a set of points. It can easily seen that $\mathsf{pred}_\exists(\bigvee\{c_i\}) = \bigvee\{\mathsf{pred}_\exists(c_i)\}$. Consider the evaluation of $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)(V)$ where $V \in 2^A$ is a set of disjoint partitions represented by constraints.

$$\begin{aligned}(\alpha \circ \mathsf{pred}_\exists \circ \gamma)(V) &= (\alpha \circ \mathsf{pred}_\exists)(\bigvee\{c_d \mid d \in V\}) \\ &= \alpha(\bigvee\{\mathsf{pred}_\exists(c_d) \mid d \in V\})\end{aligned}$$

Give a finite partition A, we pre-compute the constraint $\mathsf{pred}_\exists(c_d)$ for all $d \in A$. Let $Pre(d)$ be the stored predecessor constraint for partition element d. The results can be stored as a table, and whenever it is required to compute $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)(V)$ where $V \in 2^A$, we simply evaluate $\alpha(\bigvee\{Pre(d) \mid d \in V\})$. The abstraction function α is evaluated efficiently using the SMT solver, as already discussed.

Note that expressions of the form $\alpha(\mathsf{pred}_\exists(\bigvee\{\cdots\}))$ occur in the transformed expression for $(\alpha \circ \mathsf{pred}_\forall \circ \gamma)(V)$ above. The same optimisation can be applied here too. Our experiments show that this usually yields a considerable speedup (2-3 times faster) compared to dynamically computing the $\mathsf{pred}_\exists$ function during model checking.

5 Implementation

The abstract CTL semantic function was implemented directly in Prolog without serious attempt at optimisation of the algorithm. The function $[\![\phi]\!]^a$ yielding a set S is represented by the predicate `absCtl(Phi,S)`, where `Phi` is a suitable representation of a CTL formula. Thus for example, the rule for evaluating a formula $AG\phi$, namely

$$[\![AG\phi]\!]^a = \nu Z.([\![\phi]\!]^a \cap \mathsf{apred}_\forall(Z))$$

is rendered in Prolog by the clauses

```
absCtl(ag(F),States) :-
  absCtl(F,FStates),
  gfp_ag(FStates,States).

gfp_ag(F,S) :-
  gfp('$VAR'('Z'), intersect(F,predForall('$VAR'('Z'))), S).
```

The predicate `gfp(Z,F,S)` computes the greatest fixed point of the function $\lambda Z.(F(Z))$, and is implemented naively as shown below by computing successive iterations $A, F(A)$, $F(F(A)), \ldots$ until $F^j(A) = F^{j+1}(A)$ for some j. Here, A is the set of all abstract regions for the system under consideration. There are improved fixpoint algorithms in the literature which could be applied, e.g. [4].

```
gfp(Z,E, S1) :-
  allStates(S),
  gfp_iteration(S, Z, E, S1).

gfp_iteration(Prev, Z, E, Fix) :-
  apply_arg(Z,Prev,E,E1),
  evalExpr(E1,Next),
  gfp_checkfix(Next,Prev,Z,E,Fix).

gfp_checkfix(E1,Prev,_,Fix) :-
  subset(Prev,E1),
  !,
  returnfixpoint(E1,Fix).
gfp_checkfix(E1,_,Z,E,Fix) :-
  gfp_iteration(E1, Z, E, Fix).

returnfixpoint(X,X).
```

The most relevant aspect of the prototype implementation is the interface to external libraries to perform constraint-solving functions. In implementing the $\mathsf{pred}_\exists$ operation we make use of a Ciao-Prolog interface to the PPL library [1]. In particular, this is used to compute the proj function. The α function is implemented using the SMT solver Yices [13]. We implemented an interface predicate `yicessat(C,Xs)`, where `C` is a constraint and `Xs` is the set of variables in `C`. This predicate simply translates `C` to the syntax of Yices, and succeeds if and only if Yices finds that the constraint is satisfiable. Using this predicate the definition of α, that is $\alpha(c) = \{d \mid \mathsf{SAT}(c_d \wedge c)\}$ can be implemented directly as defined.

6 Experiments Using an SMT Constraint Solver

Figure 1 shows the transitions of a water-level controller taken from [18]. The transitions are represented as constraint logic program clauses generated automatically from a Linear Hybrid Automaton specification of the controller, as explained in detail in [2]. The state variables in an atomic formula of form `rState(X,W,T,L)` represent the rate of flow (`X`), the water-level (`W`), the elapsed time (`T`) and the location identifier (`L`). The meaning of a clause of form

```
rState(X,W,T,L) :- c(X,W,T,L,X1,W1,T1,L1), rState(X1,W1,T1,L1)
```

is a transition rule $(\mathtt{X1},\mathtt{W1},\mathtt{T1},\mathtt{L1}) \xrightarrow{c(\mathtt{X},\mathtt{W},\mathtt{T},\mathtt{L},\mathtt{X1},\mathtt{W1},\mathtt{T1},\mathtt{L1})} (\mathtt{X},\mathtt{W},\mathtt{T},\mathtt{L})$. The initial state is given by the clause `rState(0,1,1)`. Note that there are transitions both from one location to another, and also from a location to itself, since the controller can remain in a location so long as an invariant is satisfied.

Figure 2 shows the result of an analysis of the reachable states of the system, based on computing an approximation of the minimal model of the constraint program in Figure 1. There are 8 *regions*, which cover the reachable states of the controller starting in the initial state (which is region 1). The term `v(N, rState(A,B,C,D),[])` means

```
rState(0,1,1).
rState(A,B,C,1) :-A<D, B = E+A-D, C = F+A-D, 10<
                 rState(D,E,F,1).
rState(0,A,B,1) :-D<E, E = F-2*(D-C), G = H+D-C, G=2, B = G, A=E,
                 rState(C,F,H,4).
rState(A,B,C,2) :-A<D, B = E+A-D, C = F+A-D, 2<
                 rState(D,E,F,2).
rState(0,A,B,2) :-D<E, E = F+D-C, = G+D-C, E=10, B = 0, A=E,
                 rState(C,F,G,1).
rState(A,B,C,3) :-A<D, B = E-2*(A-D), C = F+A-D,5<B>
                 rState(D,E,F,3).
rState(0,A,B,3) :-D<E, E = F+D-C, G = H+D-C, G=2, B = G, A=E,
                 rState(C,F,H,2).
rState(A,B,C,4) :-A<D, B = E-2*(A-D), C = F+A-D,2<C<
                 rState(D,E,F,4).
rState(0,A,B,4) :-D<E, E = F-2*(D-C), = G+D-C, E=5, B = 0, A=E,
                 rState(C,F,G,3).
```

Fig. 1. The Water-Level Controller

```
v(1, rState(A,B,C,D), [1*A=0,1*B=1,D=1]).
v(2, rState(A,B,C,D), [-1*B> -10,1*B>1,1*A+ -1*B= -1,D=1]).
v(3, rState(A,B,C,D), [1*B=10,1*A=0,1*C=0,D=2]).
v(4, rState(A,B,C,D), [-1*C> -2,1*C>0,1*A+ -1*C=0,1*B+ -1*C=10,D=2]).
v(5, rState(A,B,C,D), [1*B=12,1*A=0,1*C=2,D=3]).
v(6, rState(A,B,C,D), [-2*C> -11,1*C>2,1*A+ -1*C= -2,1*B+2*C=16,D=3]).
v(7, rState(A,B,C,D), [1*B=5,1*A=0,1*C=0,D=4]).
v(8, rState(A,B,C,D), [-1*C> -2,1*C>0,1*A+ -1*C=0,1*B+2*C=5,D=4]).
```

Fig. 2. Disjoint Regions of the Water-Level Controller

that the region labelled `N` is defined by the constraint in the third argument, with constraint variables `A,B,C,D` corresponding to the given state variables. The 8 regions are disjoint. We use this partition to construct the abstract domain as described in Section 4.3.

Our implementation of the abstract semantics function is in Ciao-Prolog with external interfaces to the Parma Polyhedra Library [1] and the Yices SMT solver [13]. Our prototype implementation of the fixpoint computations is very naive. Nonetheless we successfully checked many CTL formulas including those with CTL operators nested in various ways, which in general is not allowed in either UPPAAL [3] or HyTech [19].

Table 1 gives the results of abstract model checking two systems, namely, a water level monitor and a task scheduler. Both of these systems are taken from [18]. In the table: (i) the columns *System* and *Formula* indicate the system and the formula being checked; (ii) the columns A and Δ, respectively, indicate the number of abstract regions and original transitions in a system and (iii) the column *time* indicates the computation

time to check a formula on the computer with an Intel XEON CPU running at 2.66GHz and with 4GB RAM.

6.1 Water level controller

For the water level system, which has 4 state variables, no formula that we have tried to evaluate takes longer than 0.2 seconds to check. Since verifying certain properties requires finer abstractions (as discussed shortly in Section 6.3), we consider two variants, a coarse one with eight and a more refined abstraction with twelve abstract regions.

The formula $EF(W = 10)$ means "there exists a path along which eventually the water level (W) reaches 10", while $AG(W = 10 \rightarrow EF(W \neq 10))$ means "on every path it is possible for the water level not to get stuck at 10". The formula $EF(AG(min \leq W \leq max))$ where $min, max \in R$ states "possibly the water level stabilises and fluctuates between a minimum *min* and maximum *max*". Using this formula, we can check whether the system reaches a stable region with the given bounds on the water level.

6.2 Scheduler

For the scheduler system that has 8 state variables, 42 abstract regions and 12 transitions, the checking time increases. Here, formulas can take up to 5-6 seconds to check in our prototype implementation. We proved a number of safety and liveness properties, again successfully checking properties of a form beyond the capability of other model checkers. For example the formula $AG(K2 > 0 \rightarrow EF(K2 = 0))$, containing an EF nested within an AG, means that the tasks of high priority (whose presence is indicated by a strictly positive value of $K2$) do not get starved (that is, the value of $K2$ eventually returns to zero).

6.3 Increasing precision by property-specific refinements

The property $EF(W = 3)$ in the water level controller should hold on the system. But this formula cannot be verified when the state space is abstracted with 8 abstract regions. Because of the coarse abstraction, we cannot distinguish $W = 3$ from $W \neq 3$. The negation of the formula, namely $AG(W > 3 \vee W < 3)$, holds in the abstract initial state since there are infinite paths from region 1 which always stay in regions that are consistent with $W \neq 3$.

One approach to solving such cases is to make a property-specific refinement to the abstraction. Each region is split into three regions by adding $W = 3$, $W > 3$ and $W < 3$ respectively to each region. Consequently, since there are 8 regions in the current abstraction (shown in 2), we get a new abstraction with 24 abstract regions, of which only 12 are satisfiable. Only the satisfiable regions need to be retained, giving a total of 12 regions in this example. With this refined abstraction, the property $EF(W = 3)$ can then successfully be checked.

System	*Formula*	A	Δ	*Time (secs.)*
Waterlevel Monitor	$EF(W = 10)$	8	8	0.08
	$AG(W = 10 \rightarrow EF(W \neq 10))$	8	8	0.06
	$EF(AG(1 \leq W \leq 12))$	8	8	0.04
	$AG(W \geq 1))$	8	8	0.01
	$EF(W = 3)$	12	8	0.16
Task Scheduler	$EF(K2 = 1)$	42	14	5.51
	$AG(K2 > 0 \rightarrow EF(K2 = 0))$	42	14	3.49
	$AG(K2 \leq 1)$	42	14	3.49

Table 1. Experimental Results

6.4 Limitations implied by our modelling technique

We cannot always successfully check formulas of the form $AF\phi$, due to an abstraction introduced into our model of continuous behaviour (rather than the abstraction induced by the Galois connection). The reason for this is that the transitions of the system in general include additional self-transitions that were not intended in the original system. Such transitions with the *same location* for the successor state as well as predecessor state are those which do not respect the continuity of the physical system. For example, the transition rules of the water level controller allow a transition within location 1 directly from $W = 1$ to $W = 5$ without passing through $W = 3$. When trying to prove a formula of the form $AF\phi$, we need to refute the formula $\neg(AF\phi)$ i.e. $EG\neg\phi$. Because of the extra self transitions, there might exist a path from the initial state on which $\neg\phi$ holds forever. Thus refutation might not be possible. Other modelling techniques are needed to capture continuity.

7 Related Work

The topic of model-checking infinite state systems using some form of abstraction has been already widely studied. Abstract model checking is described by Clarke *et al.* [6]. In this approach a state-based abstraction is defined where an abstract state is a set of concrete states. A state abstraction together with a concrete transition relation Δ induces an *abstract transition relation* Δ_{abs}. Specifically, if X_1, X_2 are abstract states, $(X_1, X_2) \in \Delta_{abs}$ iff $\exists x_1 \in X_1, x_2 \in X_2$ such that $(x_1, x_2) \in \Delta$. From this basis an abstract Kripke structure can be built; the initial states of the abstract Kripke structure are the abstract states that contain a concrete initial state, and the property labelling function of the abstract Kripke structure is induced straightforwardly as well. Model checking over the abstract Kripke structure is correct for *universal* temporal formulas (ACTL), that is, formulas that do not contain operators EX, EF, EG or EU. Intuitively, the set of paths in the abstract Kripke structure represents a superset of the paths of the concrete Kripke structure. Hence, any property that holds for all paths of the abstract Kripke structure also holds in the concrete structure. If there is a finite number of abstract states, then the abstract transition relation is also finite and thus a standard (finite-state) model checker can be used to perform model-checking of ACTL properties. However,

if an ACTL property does not hold in the abstract structure, nothing can be concluded about the concrete structure, and furthermore checking properties containing existential path quantifiers is not sound in such an approach.

This technique for abstract model checking can be reproduced in our approach, although we do not explicitly use an abstract Kripke structure. Checking an ACTL formula is done by negating the formula and transforming it to negation normal form, yielding an *existential* temporal formula (ECTL formula). Checking such a formula using our semantic function makes use of the $\mathsf{pred}_\exists$ function but not the $\mathsf{pred}_\forall$ function. It can be shown that the composition $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$ gives the $\mathsf{pred}_\exists$ function for the abstract transition relation defined by Clarke *et al.* Note that whereas abstract model checking the ACTL formula with an abstract Kripke structure yields an under-approximation of the set of states where the formula holds, our approach yields the complement, namely an over-approximation of the set of states where the negation of the formula holds.

There have been different techniques proposed in order to overcome the restriction to ACTL formulas. Dams *et al.* [10] present a framework for constructing abstract interpretations for transition systems. This involves constructing a *mixed transition system* containing two kinds of transition relations, the so-called free and constrained transitions. Godefroid *et al.* [16] proposed the use of *modal transition systems* [22] which consist of two components, namely *must*-transitions and *may*-transitions. In both [10] and [16], given a state abstraction together with a concrete transition system, a mixed transition system, or an (abstract) modal transition system respectively, is automatically generated. Following this, a modified model-checking algorithm is defined in which any formula can be checked with respect to the dual transition relations. There are certainly similarities between these approaches and ours, though more study of the precise relationship is needed. The *may*-transitions are captured by the abstract transitions defined by Clarke *et al.* [6] and hence by our abstract function $(\alpha \circ \mathsf{pred}_\exists \circ \gamma)$, as discussed above. We conjecture that the *must*-transitions are closely related to the abstract function $(\alpha \circ \mathsf{pred}_\forall \circ \gamma)$. We argue that the construction of abstract transition systems, and the consequent need to define different transitions preserving universal and existential properties, is an avoidable complication, and that our approach is conceptually simpler. Probably the main motivation for the definition of abstract transition systems is to re-use existing model checkers, as remarked by Cousot and Cousot [9].

The application of the theory of abstract interpretation to temporal logic, including abstract model checking, is thoroughly discussed by Cousot and Cousot [8, 9]. Our abstract semantics is inspired by these works, in that we also proceed by direct abstraction of a concrete semantic function using a Galois connection, without constructing any abstract transition relations. The technique of constructing abstract functions based on the pattern $(\alpha \circ f \circ \gamma)$, while completely standard in abstract interpretation [7], is not discussed explicitly in the temporal logic context. We focus only on state-based abstractions (Section 9 of [9]) and we ignore abstraction of traces. Our contribution compared to these works is to work out the abstract semantics for a specific class of constraint-based abstractions, and point the way to effective abstract model checking implementations using SMT solvers. Kelb [21] develops a related abstract model checking algo-

rithm based on abstraction of universal and existential predecessor functions which are essentially the same as our $\mathsf{pred}_\forall$ and $\mathsf{pred}_\exists$ functions.

Giacobazzi and Quintarelli [15] discuss abstraction of temporal logic and their refinement, but deal only with checking universal properties.

Our technique for modelling and verifying real time and concurrent systems using constraint logic programs builds on the work of a number of other authors, including Gupta and Pontelli [17], Jaffar *et al.* [20] and Delzanno and Podelski [11]. However we take a different direction from them in our approach to abstraction and checking of CTL formulas, in that we use abstract CLP program semantics when abstracting the state space (only briefly covered in the present work), but then apply this abstraction in a temporal logic framework, which is the topic of this work. Other authors have encoded both the transition systems and CTL semantics as constraint logic programs [5, 23, 25, 12, 14, 26, 27]. However none of these develops a comprehensive approach to abstract semantics when dealing with infinite-state systems. Perhaps a unified CLP-based approach to abstract CTL semantics could be constructed based on these works.

8 Conclusion

We have demonstrated a practical approach to abstract model checking, by constructing an abstract semantic function for CTL based on a Galois connection. Most previous work on abstract model checking is restricted to verifying *universal properties* and requires the construction of an abstract transition system. In other approaches in which arbitrary properties can be checked [16, 10], a dual abstract transition system is constructed. Like Cousot and Cousot [9] we do not find it necessary to construct any abstract transition system, but abstract the concrete semantic function systematically. Using abstract domains based on constraints we are able to implement the semantics directly. The use of an SMT solver adds greatly to the effectiveness of the approach.

Acknowledgements. We gratefully acknowledge discussions with Dennis Dams and César Sánchez.

References

1. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *SAS 2002*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2002.
2. G. Banda and J. P. Gallagher. Analysis of Linear Hybrid Systems in CLP. In M. Hanus, editor, *LOPSTR 2008*, volume 5438 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2009.
3. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *SFM-RT 2004*, number 3185 in Lecture Notes in Computer Science, pages 200–236. Springer, September 2004.
4. A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997.
5. C. Brzoska. Temporal logic programming in dense time. In *ILPS*, pages 303–317. MIT Press, 1995.

6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
7. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL'79*, pages 269–282. ACM Press, New York, U.S.A., 1979.
8. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Autom. Softw. Eng.*, 6(1):69–95, 1999.
9. P. Cousot and R. Cousot. Temporal abstract interpretation. In *POPL'2000*, pages 12–25, 2000.
10. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.
11. G. Delzanno and A. Podelski. Model checking in CLP. In *TACAS*, pages 223–239, 1999.
12. X. Du, C. R. Ramakrishnan, and S. A. Smolka. Real-time verification techniques for untimed systems. *Electr. Notes Theor. Comput. Sci.*, 39(3), 2000.
13. B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. B. Jones, editors, *CAV 2006*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.
14. F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite-state systems by specializing constraint logic programs. In M. Leuschel, A. Podelski, C. Ramakrishnan, and U. Ultes-Nitsche, editors, *Proceedings of the Second International Workshop on Verification and Computational Logic (VCL'2001)*, pages 85–96. Tech. Report DSSE-TR-2001-3, University of Southampton, 2001.
15. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples, and refinements in abstract model-checking. In P. Cousot, editor, *Static Analysis, 8th International Symposium, SAS 2001, Paris, France, July 16-18, 2001, Proceedings*, volume 2126 of *Lecture Notes in Computer Science*, pages 356–373. Springer, 2001.
16. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In K. G. Larsen and M. Nielsen, editors, *CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2001.
17. G. Gupta and E. Pontelli. A constraint-based approach for specification and verification of real-time systems. In *IEEE Real-Time Systems Symposium*, pages 230–239, 1997.
18. N. Halbwachs, Y. E. Proy, and P. Raymound. Verification of linear hybrid systems by means of convex approximations. In *SAS'94*, volume 864 of *Lecture Notes in Computer Science*, pages 223–237, 1994.
19. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer, 1997.
20. J. Jaffar, A. E. Santosa, and R. Voicu. A CLP proof method for timed automata. In J. Anderson and J. Sztipanovits, editors, *The 25th IEEE International Real-Time Systems Symposium*, pages 175–186. IEEE Computer Society, 2004.
21. P. Kelb. Model checking and abstraction: A framework preserving both truth and failure information. Technical report, Carl yon Ossietzky Univ. of Oldenburg, Oldenburg, Germany, 1994.
22. K. G. Larsen and B. Thomsen. A modal process logic. In *Proceedings, Third Annual Symposium on Logic in Computer Science, 5-8 July 1988, Edinburgh, Scotland, UK*, pages 203–210. IEEE Computer Society, 1988.
23. M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'99)*, volume 1817 of *Springer-Verlag Lecture Notes in Computer Science*, pages 63–82, April 2000.
24. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., 1999.

25. U. Nilsson and J. Lübcke. Constraint logic programming for local and symbolic model-checking. In *Computational Logic*, volume 1861 of *LNCS*, pages 384–398, 2000.
26. G. Pemmasani, C. R. Ramakrishnan, and I. V. Ramakrishnan. Efficient real-time model checking using tabled logic programming and constraints. In *ICLP*, volume 2401 of *Lecture Notes in Computer Science*, pages 100–114, 2002.
27. J. C. Peralta and J. P. Gallagher. Convex hull abstractions in specialization of CLP programs. In M. Leuschel, editor, *Logic Based Program Synthesis and Tranformation, 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17-20,2002, Revised Selected Papers*, pages 90–108, 2002.
28. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.