Constraint-Based Abstract Semantics for Temporal Logic

A Direct Approach to Design and Implementation

Banda, Gourinath; Gallagher, John Patrick

Published in:
Logic for Programming, Artificial Intelligence, and Reasoning

Publication date:
2010

Document Version
Early version, also known as pre-print

Citation for published version (APA):
Banda, G., & Gallagher, J. P. (2010). Constraint-Based Abstract Semantics for Temporal Logic: A Direct Approach to Design and Implementation. In E. M. Clarke, & A. Voronkov (Eds.), *Logic for Programming, Artificial Intelligence, and Reasoning: 16th International Conference, LPAR-16, Dakar, Senegal,* (pp. 27-45). Springer.

Constraint-Based Abstract Semantics for Temporal Logic: A Direct Approach to Design and Implementation *

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 provides a practical approach to verifying properties of infinite-state systems. We apply the framework of abstract interpretation to derive an abstract semantic function for the modal μ-calculus, which is the basis for abstract model checking. The abstract semantic function is constructed directly from the standard concrete semantics together with a Galois connection between the concrete state-space and an abstract domain. There is no need for mixed or modal transition systems to abstract arbitrary temporal properties, as in previous work in the area of abstract model checking. Using the modal μ-calculus to implement CTL, the abstract semantics gives an over-approximation of the set of states in which an arbitrary CTL formula holds. Then we show that this leads directly to an effective implementation of an abstract model checking algorithm for CTL using abstract domains based on linear constraints. The implementation of the abstract semantic function makes use of an SMT solver. We describe an implemented system for proving properties of linear hybrid automata and give some experimental results.

1 Introduction

In this paper we apply the framework of abstract interpretation [12] to design and implement an abstraction of temporal logic, based on linear constraints. We emphasise firstly that abstraction of the concrete semantics of a language such as the modal μ-calculus or CTL gives safe approximations for arbitrary formulas. Some other previous approaches handle only universal formulas (e.g. [11]). Secondly, we do not need to introduce extra conceptual apparatus in the semantics such as mixed or modal transition systems (e.g. [16, 25]) in order to approximate the meaning of arbitrary formulas. Thirdly we show that the abstract semantics can be directly implemented, for domains based on constraints, using a constraint solver (a convex polyhedra library) and satisfiability checker

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

(an SMT solver) and applied to prove properties of real-time systems modelled as linear hybrid automata.

The present work is part of an attempt to develop a uniform constraint-based formal modelling and verification framework for verifying infinite state reactive systems. The modelling part of this framework was considered in [3] where it was shown how to model linear hybrid automata (LHA) specifications as constraint logic programs. However the techniques described in the present work are not limited to constraint-based models and properties but applies to any abstraction in the framework of abstract interpretation. This work is also orthogonal to other highly interesting and relevant areas of abstract model checking such as abstract domain construction and the refinement of abstractions. We believe that basing our presentation based on completely standard semantics and abstract interpretation techniques will facilitate cross-fertilisation of techniques from abstract interpretation and model checking.

The structure of this paper is as follows. Section 2 reviews the syntax and semantics of the modal μ-calculus and recalls how this can be used to define the semantics of typical temporal property languages such as CTL. The theory of abstract interpretation is then outlined. Section 3 describes abstract interpretation of the μ-calculus semantic function, providing a basis for abstract model checking. Section 4 shows how to define an abstraction based on linear constraints. Section 5 describes an implementation of the constraint-based abstract semantics and Section 6 gives some experimental results. In Sections 7 and 8 we discuss related work and conclusions.

2 Preliminaries

The (propositional modal) μ-calculus "provides a single, simple and uniform framework subsuming most other logics of interest for reasoning about reactive systems" [20]. The set of μ-calculus formulas is defined by the following grammar.

$$\phi ::= p \mid \neg p \mid Z \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid AX\phi \mid EX\phi \mid \mu Z.\phi \mid \nu Z.\phi$$

where p ranges over a set of atomic formulas $\mathcal{P}$ and Z ranges over a set of propositional variables $\mathcal{V}$. Note that negations can appear only before propositions p. This form is called *negation normal form*; if we extend the grammar to allow expressions of the form $\neg\phi$ in which occurrences of Z in ϕ in formulas fall under an even number of negations, an equivalent formula in negated normal form can be obtained using rewrites (which are justified by the semantics below), namely $\neg\mu Z.\phi \Rightarrow \nu Z.\neg\phi$, $\neg\nu Z.\phi \Rightarrow \mu Z.\neg\phi$, $\neg EX\phi \Rightarrow AX\neg\phi$, $\neg AX\phi \Rightarrow EX\neg\phi$ together with De Morgan's laws and elimination of double negation.

2.1 Semantics of μ-calculus

There are various presentations of the semantics of the μ-calculus, e.g. [20, 30]. We restrict our attention here to state-based semantics, which means that given

a Kripke structure K, a μ-calculus formula evaluates to the set of states in K at which the formula holds. The semantics is presented in a functional style similar to that given in [15] Section 9, suitable for applying abstraction. A μ-calculus formula is interpreted 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 1 (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 a total transition relation (i.e. every state has a successor), $I \subseteq S$ is the set of initial states, $\mathcal{P}$ is the set of propositions and $L : S \to 2^{\mathcal{P}}$ 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.*

We first define some subsidiary functions that depend on K.

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

- *$pre(S') = \{s \mid \exists s' \in S' : (s, s') \in \Delta\}$ returns the set of states having at least one of their successors in the set $S' \subseteq S$;*
- *$\widetilde{pre}(S') = \mathsf{compl}(pre(\mathsf{compl}(S')))$ 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.*

The functions pre and $\widetilde{pre}$ are defined by several authors (e.g. [32, 15]) and are also used with other names by other authors (e.g. they are called $pre_\exists$ and $pre_\forall$ by Huth and Ryan [30]).

Lemma 1. *pre and $\widetilde{pre}$ are monotonic.*

Let Mu be the set of μ-calculus formulas, and $\mathcal{V} \to 2^S$ be the set of environments for the free variables. The meaning of a formula is a mapping from an environment giving values to its free variables to a set of states. The semantics function $[\![.]\!]_\mu : \mathsf{Mu} \to (\mathcal{V} \to 2^S) \to 2^S$ is defined as follows.

$$
\begin{array}{ll}
[\![Z]\!]_\mu\sigma = \sigma(Z) & \\
[\![p]\!]_\mu\sigma = \mathsf{states}(p) & [\![\neg p]\!]_\mu\sigma = \mathsf{states}(\neg p) \\
[\![EX\phi]\!]_\mu\sigma = pre([\![\phi]\!]_\mu\sigma) & [\![\phi_1 \vee \phi_2]\!]_\mu\sigma = [\![\phi_1]\!]_\mu\sigma \ \cup \ [\![\phi_2]\!]_\mu\sigma \\
[\![AX\phi]\!]_\mu\sigma = \widetilde{pre}([\![\phi]\!]_\mu\sigma) & [\![\phi_1 \wedge \phi_2]\!]_\mu\sigma = [\![\phi_1]\!]_\mu\sigma \ \cap \ [\![\phi_2]\!]_\mu\sigma \\
[\![\mu Z.\phi]\!]_\mu\sigma = \mathsf{lfp}(F) & [\![\nu Z.\phi]\!]_\mu\sigma = \mathsf{gfp}(F) \\
\quad \text{where } F(S') = [\![\phi]\!]_\mu\sigma[Z/S'] & \quad \text{where } F(S') = [\![\phi]\!]_\mu\sigma[Z/S']
\end{array}
$$

The expressions $\mathsf{lfp}(F)$ and $\mathsf{gfp}(F)$ return the least fixed point and greatest fixed point respectively of the monotonic function $F : 2^S \to 2^S$ on the lattice $\langle 2^S, \subseteq, \cup, \cap, S, \emptyset\rangle$. The Knaster-Tarski fixed point theorem [41] guarantees the existence of least and greatest fixed points for a monotonic function on a complete lattice, and also their constructive forms $\bigcup_{i=0}^{\infty} F^i(\emptyset)$ and $\bigcap_{i=0}^{\infty} F^i(S)$ respectively. The environment $\sigma[Z/S']$ above is such that $\sigma[Z/S']Y = S'$ if $Y = Z$ and $\sigma(Y)$ otherwise.The functions F above are monotonic due the restricted occurrence of negation symbols in negated normal form.

When a formula contains no free variables we can evaluate it in a trivial environment $\sigma_\emptyset$ in which all variables are mapped (say) to the empty set. If ϕ contains no free variables we thus define its meaning $[\![\phi]\!] = [\![\phi]\!]_\mu \sigma_\emptyset$.

2.2 CTL Syntax and Semantics

The set of CTL formulas ϕ in *negation normal form* is inductively defined by the following grammar:

$$\begin{aligned}\phi ::= & \; p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \\ & \mid AU(\phi_1, \phi_2) \mid EU(\phi_1, \phi_2) \mid AR(\phi_1, \phi_2) \mid ER(\phi_1, \phi_2)\end{aligned}$$

where p ranges over a set of atomic formulas $\mathcal{P}$. We assume familiarity with the intended meanings of the various temporal operators. Note that we sometimes refer to arbitrary negations $\neg\phi$ in what follows, but such formulas can be transformed to equivalent negation normal form formulas.

CTL is interpreted by translating to μ-calculus, using the following function C.

$$\begin{array}{ll|ll}
\mathsf{C}(p) & = p & \mathsf{C}(\neg p) & = \neg p \\
\mathsf{C}(EX\phi) & = EX\ \mathsf{C}(\phi) & \mathsf{C}(\phi_1 \vee \phi_2) & = \mathsf{C}(\phi_1) \vee \mathsf{C}(\phi_2) \\
\mathsf{C}(AX\phi) & = AX\ \mathsf{C}(\phi) & \mathsf{C}(\phi_1 \wedge \phi_2) & = \mathsf{C}(\phi_1) \wedge \mathsf{C}(\phi_2) \\
\mathsf{C}(EF\phi) & = \mu Z.(\mathsf{C}(\phi) \vee EX\ Z) & \mathsf{C}(ER(\phi_1, \phi_2)) & = \nu Z.(\mathsf{C}(\phi_2) \wedge (\mathsf{C}(\phi_1) \vee EX\ Z) \\
\mathsf{C}(AF\phi) & = \mu Z.(\mathsf{C}(\phi) \vee AX\ Z) & \mathsf{C}(AU(\phi_1, \phi_2)) & = \mu Z.(\mathsf{C}(\phi_2) \vee (\mathsf{C}(\phi_1) \wedge AX\ Z) \\
\mathsf{C}(AG\phi) & = \nu Z.(\mathsf{C}(\phi) \wedge AX\ Z) & \mathsf{C}(EU(\phi_1, \phi_2)) & = \mu Z.(\mathsf{C}(\phi_2) \vee (\mathsf{C}(\phi_1) \wedge EX\ Z) \\
\mathsf{C}(EG\phi) & = \nu Z.(\mathsf{C}(\phi) \wedge EX\ Z) & \mathsf{C}(AR(\phi_1, \phi_2)) & = \nu Z.(\mathsf{C}(\phi_2) \wedge (\mathsf{C}(\phi_1) \vee AX\ Z)
\end{array}$$

A semantic function for CTL is then obtained by composing the translation with the semantics of the μ-calculus. The translated formulas contain no free variables (all variables Z are introduced in the scope of a μ or ν). Thus we define the semantic function for a CTL formula ϕ as $[\![\phi]\!]_{CTL} = [\![\mathsf{C}(\phi)]\!]$.

It is of course possible to partially evaluate the translation function. In this way we obtain state semantics for CTL directly. For example, the meaning of $EF\phi$ and $AG\phi$ are:

$$\begin{aligned}[\![EF\phi]\!]_{CTL} &= \mathsf{lfp}(F) \text{ where } F(S') = [\![\phi]\!]_{CTL} \cup pre(S')) \\ [\![AG\phi]\!]_{CTL} &= \mathsf{gfp}(F) \text{ where } F(S') = [\![\phi]\!]_{CTL} \cap \widetilde{pre}(S'))\end{aligned}$$

We present the semantics via μ-calculus to emphasise the generality of the approach; we can now restrict our attention to considering μ-calculus semantics.

2.3 Model Checking

Model checking consists of checking whether the Kripke structure K possesses a property ϕ, written $K \models \phi$. This is defined to be true iff $I \subseteq [\![\phi]\!]$, where I is the set of initial states, 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 μ-calculus semantics function. Specifically, the implementation of

the expressions $\mathsf{lfp}(F)$ and $\mathsf{gfp}(F)$ is performed by computing a Kleene sequence $F^i(\emptyset)$ or $F^i(S)$ respectively, iterating until the values stabilise.

When the state-space powerset 2^S has infinite $\subseteq$-chains, these iterations 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*.

2.4 Abstract Interpretation

In abstract interpretation we develop an abstract semantic function systematically from the standard ("concrete") semantics with respect to a Galois connection. We present the formal framework briefly.

Definition 3 (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 $\mathsf{gfp}(f) \sqsubseteq_L \gamma(\mathsf{gfp}(f^\sharp))$ where $\mathsf{lfp}(f), \mathsf{gfp}(f)$ are the least and greatest fixed points respectively of f.

Thus the abstract function $f^\sharp$ can be used to compute over-approximations of f, and it 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 with respect to the Galois connection. We next apply this general framework to abstraction of the μ-calculus semantics, and illustrate with a specific abstraction in Section 4.

3 Abstract Interpretation of μ-calculus semantics

We consider abstractions based on Galois connections $\langle 2^S, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$, 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 4. *Let* $pre : 2^S \to 2^S$, $\widetilde{pre} : 2^S \to 2^S$, *and* $\mathsf{states} : \mathcal{P} \to 2^S$ *be the functions defined in Definition 2. Given a Galois connection* $\langle 2^S, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$, *we define* $apre : 2^A \to 2^A$, $\widetilde{apre} : 2^A \to 2^A$ *and* $\mathsf{astates} : \mathcal{P} \to 2^A$ *as*

$$apre = \alpha \circ pre \circ \gamma \qquad \widetilde{apre} = \alpha \circ \widetilde{pre} \circ \gamma \qquad \mathsf{astates} = \alpha \circ \mathsf{states}$$

The properties of Galois connections imply that for all $S' \subseteq S$, $\alpha(pre(S')) \subseteq apre(\alpha(S'))$ and $\alpha(\widetilde{pre}(S')) \subseteq \widetilde{apre}(\alpha(S'))$. We simply substitute $apre$, $\widetilde{apre}$ and $\mathsf{astates}$ for their concrete counterparts in the μ-calculus semantic function to obtain abstract semantics for the μ-calculus.

Given a Galois connection $\langle 2^S, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$, the abstract μ-calculus semantic function $[\![\cdot]\!]^a_\mu : \mathsf{Mu} \to (\mathcal{V} \to 2^A) \to 2^A$ is defined as follows.

$$
\begin{array}{ll}
[\![Z]\!]^a_\mu\sigma = \sigma(Z) & \\
[\![p]\!]^a_\mu\sigma = \mathsf{astates}(p) & [\![\neg p]\!]^a_\mu\sigma = \mathsf{astates}(\neg p) \\
[\![EX\phi]\!]^a_\mu\sigma = apre([\![\phi]\!]^a_\mu\sigma) & [\![\phi_1 \vee \phi_2]\!]^a_\mu\sigma = [\![\phi_1]\!]^a_\mu\sigma \cup [\![\phi_2]\!]^a_\mu\sigma \\
[\![AX\phi]\!]^a_\mu\sigma = \widetilde{apre}([\![\phi]\!]^a_\mu\sigma) & [\![\phi_1 \wedge \phi_2]\!]^a_\mu\sigma = [\![\phi_1]\!]^a_\mu\sigma \cap [\![\phi_2]\!]^a_\mu\sigma \\
[\![\mu Z.\phi]\!]^a_\mu\sigma = \mathsf{lfp}(F_a) & [\![\nu Z.\phi]\!]^a_\mu\sigma = \mathsf{gfp}(F_a) \\
\quad \text{where } F_a(A') = [\![\phi]\!]^a_\mu\sigma[Z/A'] & \quad \text{where } F_a(A') = [\![\phi]\!]^a_\mu\sigma[Z/A']
\end{array}
$$

As before, for formulas containing no free variables we define the function $[\![\phi]\!]^a = [\![\phi]\!]^a_\mu\sigma_\emptyset$. The abstract semantics for a CTL formula ϕ is $[\![\mathsf{C}(\phi)]\!]^a$.

The functions α and γ are extended to apply to environments $\sigma : \mathcal{V} \to A$. $\alpha(\sigma)$ is defined as $\alpha(\sigma)(Z) = \alpha(\sigma(Z))$ and $\gamma(\sigma)$ is defined as $\gamma(\sigma)(Z) = \gamma(\sigma(Z))$.

Theorem 1 (Safety of Abstract 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 μ-calculus formula in negation normal form. Then $\alpha([\![\phi]\!]_\mu\sigma) \subseteq [\![\phi]\!]^a_\mu\alpha(\sigma)$ and $\gamma([\![\phi]\!]^a_\mu\sigma) \supseteq [\![\phi]\!]_\mu\gamma(\sigma)$, for all environments σ.*

The proof is by structural induction on ϕ. First we establish a subsidiary result.

Lemma 2. *Let $F(S') = [\![\phi]\!]_\mu\sigma[Z/S']$ and $F_a(A') = [\![\phi]\!]^a_\mu\alpha(\sigma)[Z/A']$ and let $\langle 2^S, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$ be a Galois connection. Assume $\alpha([\![\phi]\!]_\mu\sigma) \subseteq [\![\phi]\!]^a_\mu\alpha(\sigma)$. Then for all $A' \subseteq A$, $(\alpha \circ F \circ \gamma)(A') \subseteq F_a(A')$.*

Proof.

$$
\begin{array}{lll}
(\alpha \circ F \circ \gamma)(A') = \alpha(F(\gamma(A'))) & & \\
= \alpha([\![\phi]\!]_\mu\sigma[Z/\gamma(A')]) & & \\
\subseteq [\![\phi]\!]^a_\mu\alpha(\sigma)[Z/\alpha(\gamma(A'))] & \text{by assumption that} & \\
& \alpha([\![\phi]\!]_\mu\sigma) \subseteq [\![\phi]\!]^a_\mu\alpha(\sigma) & \\
\subseteq [\![\phi]\!]^a_\mu\alpha(\sigma)[Z/A'] & \text{by properties of Galois connections} & \\
& \text{and monotonicity of } [\![\phi]\!]^a_\mu\alpha(\sigma) & \\
= F_a(A') & &
\end{array}
$$

The proof of Theorem 1 is just an exercise in applying the properties of Galois connections and monotonic functions. We show a few representative cases.

Proof. (Theorem 1). We show that $\alpha([\![\phi]\!]_\mu\sigma) \subseteq [\![\phi]\!]^a_\mu\alpha(\sigma)$ by structural induction on ϕ. The proof for $\gamma([\![\phi]\!]^a_\mu\sigma) \supseteq [\![\phi]\!]_\mu\gamma(\sigma)$ is similar.

Base Cases.

– $\phi = Z$.

$$\begin{aligned}\alpha([\![Z]\!]_\mu\sigma) &= \alpha(\sigma(Z)) \\ &= \alpha(\sigma)(Z) \\ &= [\![Z]\!]^a_\mu\alpha(\sigma)\end{aligned}$$

– $\phi = p'$ where $p' = p$ or $p' = \neg p$

$$\begin{aligned}\alpha([\![p']\!]_\mu\sigma) &= \alpha(\mathsf{states}(p')) \\ &= \mathsf{astates}(p') \\ &= [\![p']\!]^a_\mu\alpha(\sigma)\end{aligned}$$

Inductive Cases

– $\phi = EX\phi$.

$$\begin{aligned}\alpha([\![EX\phi]\!]_\mu\sigma) &= \alpha(pre([\![\phi]\!]_\mu\sigma)) \\ &\subseteq \alpha(pre(\gamma(\alpha([\![\phi]\!]_\mu\sigma)))) && \text{by Galois connection} \\ & && \text{and monotonicity of } pre, \alpha \\ &= apre(\alpha([\![\phi]\!]_\mu\sigma)) \\ &\subseteq apre([\![\phi]\!]^a_\mu\alpha(\sigma)) && \text{by ind. hyp.} \\ & && \text{and monotonicity of } apre \\ &= [\![EX\phi]\!]^a_\mu\alpha(\sigma)\end{aligned}$$

– $\phi = \mu Z.\phi$.

In this case, and the case for $\phi = \nu Z.\phi$, we make use of the general property of a Galois connection $\langle 2^S, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$ that if $f : 2^S \to 2^S$ and $f^\sharp : 2^A \to 2^A$ are such that $(\alpha \circ f \circ \gamma) \subseteq f^\sharp$, then $\alpha(\mathsf{lfp}(f)) \subseteq \mathsf{lfp}(f^\sharp)$ and $\alpha(\mathsf{gfp}(f)) \subseteq \mathsf{gfp}(f^\sharp)$. The relevant condition $(\alpha \circ f \circ \gamma) \subseteq f^\sharp$ is established in Lemma 2.

$$\begin{aligned}\alpha([\![\mu Z.\phi]\!]_\mu\sigma) &= \alpha(\mathsf{lfp}(F)) && \text{where } F(S') = [\![\phi]\!]_\mu\sigma[Z/S'] \\ &\subseteq \mathsf{lfp}(F_a) && \text{where } F_a(A') = [\![\phi]\!]^a_\mu\alpha(\sigma)[Z/A'] \\ & && \text{by Lemma 2, ind. hyp., and} \\ & && \text{properties of Galois connections} \\ &= [\![\mu Z.\phi]\!]^a_\mu\alpha(\sigma)\end{aligned}$$

Corollary 1. *Let $K = \langle S, \Delta, I, L, \mathcal{P}\rangle$ be a Kripke structure and ϕ be a μ-calculus formula with no free variables. Then if $\gamma([\![\neg\phi]\!]^a) \cap I = \emptyset$ then $K \models \phi$.*

Proof.

$$\begin{aligned}\gamma([\![\neg\phi]\!]^a_\mu) \cap I = \emptyset &\equiv \gamma([\![\neg\phi]\!]^a_\mu\sigma_\emptyset) \cap I = \emptyset \\ &\Rightarrow [\![\neg\phi]\!]_\mu\gamma(\sigma_\emptyset) \cap I = \emptyset && \text{by Theorem 1} \\ &\equiv [\![\neg\phi]\!] \cap I = \emptyset && \text{since } \gamma(\sigma_\emptyset) = \sigma_\emptyset \\ &\equiv I \supseteq [\![\phi]\!] \\ &K \models \phi\end{aligned}$$

This result provides us with a sound abstract model checking procedure for any μ-calculus formula ϕ. Of course, if $\gamma([\![\neg\phi]\!]^a) \cap I \supset \emptyset$ nothing can be concluded.

4 An Abstract Constraint-based Domain

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 $\alpha(\mathsf{states}(p))$, $(\alpha \circ pre \circ \gamma)$ and $(\alpha \circ \widetilde{pre} \circ \gamma)$ might not be feasible. In general in abstract interpretation one designs functions that safely approximate these constructions. For example one would design computable functions $apre'$ and $\widetilde{apre}'$ such that for all A, $apre'(A') \supseteq apre(A')$ and $\widetilde{apre}'(A') \supseteq \widetilde{apre}(A')$. In this section we show that the abstract semantics is implementable directly, without any further approximation, for transition systems and abstract domains expressed using linear constraints.

4.1 Abstract Domains Based on a State-Space Partitions

Consider transition systems whose states are n-tuples of real numbers; 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 finite partition of C say $A = \{d_1, \ldots, d_k\}$ such that $\bigcup A = C$. Such a partition could be obtained in various ways, including predicate abstraction or Moore closures (see [23] for a discussion). Define a representation function $\beta : C \to 2^A$, such that $\beta(\bar{x}) = \{d \in A \mid \bar{x} \in d\}$. We extend the representation function [36] to sets of points, obtaining the abstraction function $\alpha : 2^C \to 2^A$ given by $\alpha(S) = \bigcup\{\beta(\bar{x}) \mid \bar{x} \in S\}$. Define the concretisation function $\gamma : 2^A \to 2^C$, as $\gamma(V) = \{\bar{x} \in C \mid \beta(\bar{x}) \subseteq V\}$. As shown in [36, 13], $\langle 2^C, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle 2^A, \subseteq\rangle$ is a Galois connection. Because A is a partition the value of $\beta(\bar{x})$ is a singleton for all $\bar{x}$, and the γ function can be written as $\gamma(V) = \bigcup\{\gamma(\{d\}) \mid d \in V\}$.

4.2 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 c is *satisfied* by an assignment of real numbers to its variables if the constraint evaluates to *true* under this assignment, and is *satisfiable* (written $SAT(c)$) 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 variables 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 V.

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 [27, 3].

4.3 Constraint representation of the semantic functions

We consider abstract semantics based on linear partitions, so that each element d_i of the partition $A = \{d_1, \ldots, d_n\}$ is representable as a linear constraint c_{d_i}. We first provide definitions of the functions pre, $\widetilde{pre}$, α, γ and states in terms of constraint operations. In the next section the optimisation and effective implementation of these operations is considered. Let T be a finite set of transition rules. Let $c'(\bar{y})$ be a constraint over variables $\bar{y}$. We express the functions pre, $\widetilde{pre}$ and states using constraint operations as follows.

$$\begin{array}{l} pre(c'(\bar{y})) = \bigvee\{\mathsf{proj}_{\bar{x}}(c'(\bar{y}) \wedge c(\bar{x}, \bar{y})) \mid \bar{x} \xrightarrow{c(\bar{x},\bar{y})} \bar{y} \in T\} \\ \widetilde{pre}(c'(\bar{y})) = \neg(pre(\neg c'(\bar{y}))) \\ \mathsf{states}(p) = p \end{array}$$

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

The β function introduced in Section 4.1 can be rewritten as $\beta(\bar{x}) = \{d \in A \mid \bar{x} \text{ satisfies } c_d\}$. Assuming that we only need to apply α to sets of points represented by a linear constraint c, we can rewrite the α and γ functions as follows.

$$\alpha(c) = \{d \in A \mid \mathsf{SAT}(c_d \wedge c)\} \qquad \gamma(A') = \bigvee\{c_d \mid d \in A'\} \text{ for } A' \subseteq A$$

5 Implementation

The abstract μ-calculus semantic function $[\![.]\!]^a$ can be implemented directly as a recursive function following the definition in Section 3. The structure of the algorithm is independent of any particular abstraction. With standard iterative techniques for computing greatest and least fixpoints [7] the algorithm has the same complexity as a concrete model checker for μ-calculus. In practice the effectiveness of the algorithm as an abstract model checker depends on the effective implementation of the functions accessing the transition system and performing abstraction and concretisation, namely $pre, \widetilde{pre}, \alpha$ and γ.

5.1 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 [19]. The γ function simply collects a disjunction of the constraints associated with the given set of partitions; no solver is required.

5.2 Optimisation of constraint-based evaluation

Combining the constraint-based evaluation of the functions pre and $\widetilde{pre}$ with the constraint-based evaluation of the α and γ functions gives us (in principle) a method of computing the abstract semantic counterparts of pre and $\widetilde{pre}$, namely $(\alpha \circ pre \circ \gamma)$ and $(\alpha \circ \widetilde{pre} \circ \gamma)$. The question we now address is the feasibility of this approach. Taken naively, the evaluation of these constraint-based functions (in particular $\widetilde{pre}$) 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 $(\alpha \circ \widetilde{pre} \circ \gamma)(A')$ where $A' \in 2^A$ is a set of disjoint partitions represented by constraints.

$$\begin{aligned}(\alpha \circ \widetilde{pre} \circ \gamma)(A') &= (\alpha \circ \widetilde{pre})(\bigvee\{c_d \mid d \in A'\}) \\ &= \alpha(\neg(pre(\neg(\bigvee\{c_d \mid d \in A'\})))) \\ &= \alpha(\neg(pre(\bigvee\{c_d \in A \setminus A'\})))\end{aligned}$$

In the last step, we use the equivalence $\neg(\bigvee\{c_d \mid d \in A'\}) \leftrightarrow \bigvee\{c_d \in A \setminus A'\}$, which is justified since the abstract domain A is a disjoint partition of the concrete domain; thus $A \setminus A'$ represents the negation of A' restricted to the state space of the system. The computation of $pre(\bigvee\{c_d \in A \setminus A'\})$ is much easier to compute (with available tools) than $pre(\neg(\bigvee\{c_d \mid d \in A'\}))$. 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 $\mathsf{proj}_{\bar{x}}(c_d(\bar{y}) \wedge c(\bar{x}, \bar{y}))$.

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

$$\begin{aligned}(\alpha \circ pre \circ \gamma)(A') &= (\alpha \circ pre)(\bigvee\{c_d \mid d \in A'\}) \\ &= \alpha(\bigvee\{pre(c_d) \mid d \in A'\})\end{aligned}$$

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

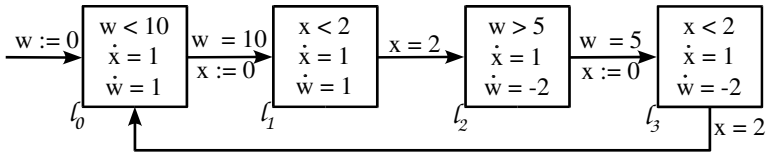

Fig. 1. A Water-level Monitor [27]

```
rState1(A,B,C,D):- rState4(E,F,G,H),
                   D=1,H=4,G≤I,1*J=1*E+1*(I-G),1*K=1*F+ -2*(I-G),
                   J=2,L=J,M=K,0≤C,1*A=1*L+1*(C-0),1*B=1*M+1*(C-0),B≤10.
rState1(A,B,C,D):- D=1,0≤C,1*A=1*0+1*(C-0),1*B=1*1+1*(C-0),B≤10.
rState2(A,B,C,D):- rState1(E,F,G,H),
                   D=2,H=1,G≤I,1*J=1*E+1*(I-G),1*K=1*F+1*(I-G),
                   K=10,L=0,M=K,0≤C,1*A=1*L+1*(C-0),1*B=1*M+1*(C-0),A≤2.
rState3(A,B,C,D):- rState2(E,F,G,H),
                   D=3,H=2,G≤I,1*J=1*E+1*(I-G),1*K=1*F+1*(I-G),J=2,
                   L=J,M=K,0≤C,1*A=1*L+1*(C-0),1*B=1*M+ -2*(C-0),B>=5.
rState4(A,B,C,D):- rState3(E,F,G,H),
                   D=4,H=3,G≤I,1*J=1*E+1*(I-G),1*K=1*F+ -2*(I-G),K=5,
                   L=0,M=K,0≤C,1*A=1*L+1*(C-0),1*B=1*M+ -2*(C-0),A≤2.
```

Fig. 2. The Water-Level Controller transition rules, automatically generated from Figure 1 [4]

Note that expressions of the form $\alpha(pre(\bigvee\{\cdots\}))$ occur in the transformed expression for $(\alpha \circ \widetilde{pre} \circ \gamma)(A')$ 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 *pre* function during model checking.

Our implementation of the abstract μ-calculus semantic function $[\![.]\!]^a_{CTL}$ was in Prolog, with interfaces to external libraries to perform constraint-solving functions. In implementing the *pre* operation we make use of a Ciao-Prolog interface to the PPL library [2]. In particular, this is used to compute the proj function. The α function is implemented using the SMT solver Yices [19]. We implemented an interface predicate `yices_sat(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

A small example is shown with all details to illustrate the process of proving a property. In Figure 1 is shown a linear hybrid automaton (LHA) for a water

```
region(1,rState1(A,B,C,D),[D=1,-1*A≥ -9,1*A≥0,1*A-1*B= -1,1*A-1*C=0]).
region(2,rState2(A,B,C,D),[D=2,-1*C≥ -2,1*C≥0,1*A-1*C=0,1*B-1*C=10]).
region(3,rState3(A,B,C,D),[D=3,-2*C≥ -7,1*C≥0,1*A-1*C=2,1*B+2*C=12]).
region(4,rState4(A,B,C,D),[D=4,-1*C≥ -2,1*C≥0,1*A-1*C=0,1*B+2*C=5]).
region(5,rState1(A,B,C,D),[D=1,-1*C≥ -9,1*C≥0,1*A-1*C=2,1*B-1*C=1]).
```

Fig. 3. Disjoint Regions of the Water-Level Controller States

level controller taken from [27]. Figure 2 shows transition rules represented as constraint logic program (CLP) clauses generated automatically from the LHA in Figure 1, as explained in detail in [3]. 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) :- rState(X1,W1,T1,L1),` c`(X,W,T,L,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(0,0,_,1)`.

Figure 3 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 2. This approximation is obtained by a tool for approximating the minimal model of an arbitrary CLP program [6, 28]. There are 5 *regions*, which cover the reachable states of the controller starting in the initial state (which is region 1). The term `region(N, rState(A,B,C,D),[...])` means 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 5 regions are disjoint. We use this partition to construct the abstract domain as described in Section 4.1.

Our implementation of the abstract semantics function is in Ciao-Prolog with external interfaces to the Parma Polyhedra Library [2] and the Yices SMT solver [19][3] Using this prototype implementation we successfully checked many CTL formulas including those with CTL operators nested in various ways, which in general is not allowed in UPPAAL [5], HYTECH [29] or PHAVER [22] (though special-purpose constructs such as a "leads-to" operator can be used to handle some cases).

Table 1 gives the results of proving properties using abstract model checking two systems, namely, a water level monitor and a task scheduler. Both of these systems are taken from [27]. In the table: (i) the columns *System* and *Property* 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 prove a formula on the computer with an Intel XEON CPU running at 2.66GHz and with 4GB RAM.

Water level controller. The water level system has 4 state variables and 4 transition rules. A variety of different properties that were proved is shown in Table

[3] Prolog code available at `http://akira.ruc.dk/~jpg/Software/amc_all.pl`.

1. In some cases we derived a more precise partition than the one originally returned for the transition system, using the technique described below in Section 6.1.

The formula $AF(W \geq 10)$ means "on all paths the water level (W) reaches at least 10", while $AG(W = 10 \rightarrow AF(W < 10 \vee W > 10))$ means "on every path the water level cannot get stuck at 10". The formula $AG(0 \leq W \wedge W \leq 12)$ is a global safety property stating that the water level remains within the bounds 0 and 12. A more precise invariant is reached some time after the initial state and this is proved by the property $AF(AG(1 \leq W \wedge W \leq 12))$, in other words "on all paths, eventually the water level remains between 1 and 12". Since for all ϕ, $AG(\phi) \equiv AG(AG(\phi))$ we are able to prove $AG^5(0 \leq W \wedge W \leq 12)$ which is only shown in order to indicate the capability of the prover to handle arbitrarily deeply nested formulas. The formula $EF(W = 10)$ shows that the water level can reach exactly 10. Finally, the formula $EU(W < 12, AU(W < 12, W \geq 12))$ shows another example of a verified progress property (which could be formulated more simply but is shown just to exercise the prover's capabilities).

Scheduler. The scheduler system for two processes has 8 state variables, 18 abstract regions and 12 transition rules. 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 nested formula $AG(K2 > 0 \rightarrow AF(K2 = 0))$ is a critical correctness property meaning that 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 on all paths).

6.1 Property-specific refinements

The topic of refinement is a very relevant and much-studied area in proof by abstraction. Refinement is applied when a given abstraction is too coarse to prove some property. In this case we seek to derive a more precise refinement, that is somehow more relevant to the property being proved. Refinement is not the main focus in this paper, but we discuss briefly the use of a property-specific refinement that we can apply to increase the number of provable properties. Consider the property $EF(W = 10)$ in the water level controller, which holds on the system. But this formula cannot be verified when the state space is abstracted with regions that cannot distinguish states where $W = 10$. The negation of the formula, namely $AG(W > 10 \vee W < 10)$, holds in the abstract initial state since there are infinite paths from the initial region which always stay in regions that are consistent with $W \neq 10$.

One approach to solving such cases is to make a property-specific refinement to the abstraction. For a given constraint property p each region is split into further regions by adding the constraints p and $\neg p$ respectively to each region. Only the satisfiable regions need to be retained. With this refined abstraction using $(W = 10)$ for p, the property $EF(W = 10)$ can then successfully be checked.

System	*Property*	A	Δ	*Time (secs.)*
Waterlevel	$AF(W \geq 10)$	5	4	0.02
Monitor	$AG(0 \leq W \wedge W \leq 12)$	5	4	0.01
	$AF(AG(1 \leq W \wedge W \leq 12))$	5	4	0.02
	$AG(W = 10 \rightarrow AF(W < 10 \vee W > 10))$	10	4	0.05
	$AG(AG(AG(AG(AG(0 \leq W \wedge W \leq 12)))))$	5	4	0.02
	$EF(W = 10)$	10	4	0.01
	$EU(W < 12, AU(W < 12, W \geq 12))$	7	4	0.04
Task Scheduler	$EF(K2 = 1)$	18	12	0.53
	$AG(K2 > 0 \rightarrow AF(K2 = 0))$	18	12	0.30
	$AG(K2 \leq 1)$	18	12	0.04

Table 1. Experimental Results

The CEGAR approach to refinement [9] is quite compatible with our approach, but we are also interested in investigating more general forms of refinement investigated in the theory of abstract interpretation [23].

7 Related Work

The topic of *model-checking infinite state systems* [4] and using some form of abstraction has been already widely studied. Abstract model checking is described by Clarke *et al.* [10, 11]. 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 CTL properties 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. Checking properties containing existential path quantifiers is not sound in such an approach.

[4] When we say model checking of (continuous) infinite state systems, it means *model-checking the discrete abstractions of infinite state systems*. In [1], it is established that hybrid systems can be safely abstracted with discrete systems preserving all the temporal properties expressed in branching-time temporal logics as well as linear-time temporal logics.

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 *pre* function but not the $\widetilde{pre}$ function. For this kind of abstraction the relation on abstract states $s \rightarrow s'$ defined as $s \in (\alpha \circ pre \circ \gamma)(\{s'\})$ is identical to 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 universal formulas. Dams *et al.* [16] present a framework for constructing abstract interpretations for μ-calculus properties in 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.* [25] proposed the use of *modal transition systems* [33] which consist of two components, namely *must*-transitions and *may*-transitions. In both [16] and [25], given an 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. Our approach by contrast is based on the standard semantics of the μ-calculus. The *may*-transitions and the *must*-transitions of [25] could be obtained from the functions $(\alpha \circ pre \circ \gamma)$ and $(\alpha \circ \widetilde{pre} \circ \gamma)$ respectively. For the case of an abstraction given by a partition $A = \{d_1, \ldots, d_n\}$ it seems that an abstract modal transition system could be constructed with set of states A such that there is a *may*-transition $d_i \rightarrow d_j$ iff $d_i \in (\alpha \circ pre \circ \gamma)(\{d'_j\}$ and a *must*-transition $d_i \rightarrow d_j$ iff $d_i \in (\alpha \circ \widetilde{pre} \circ \gamma)(\{d_j\}$. However the two approaches are not interchangeable; in [25] a concrete modal transition system has the same set of *must*-transitions and *may*-transitions, but applying the above constructions to the concrete state-space (with α and γ as the identity function) does not yield the same sets of *must*- and *may*-transitions (unless the transition system is deterministic). We have shown that the construction of abstract transition systems as in [10, 11], and abstract modal transition systems in particular [16, 25] is an avoidable complication in abstraction. Probably the main motivation for the definition of abstract transition systems is to re-use existing model checkers, as remarked by Cousot and Cousot [15] (though this argument does not apply to modal or mixed transition systems in any case).

Property-preserving abstraction using Galois connections was applied in a μ-calculus setting by Loiseaux *et al.* [35]. Our aim and approach are similar, but are both more general and more direct. The cited work develops sounds abstractions for universal properties only whereas we handle arbitrary properties. On the other hand it uses Galois connections to develop a simulation relation

between concrete and abstract systems, which goes beyond the scope of the current work. The application of the theory of abstract interpretation to temporal logic, including abstract model checking, is thoroughly discussed by Cousot and Cousot [14, 15]. Our abstract semantics is inspired by their approach, 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 [13], is not discussed explicitly in the temporal logic context. We focus only on state-based abstractions (Section 9 of [15]) 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 [32] develops a related abstract model checking algorithm based on abstraction of universal and existential predecessor functions.

Giacobazzi and Quintarelli [24] discuss abstraction of temporal logic and their refinement, but deal only with checking universal properties. Saïdi and Shankar [40] also develop an abstract model checking algorithm integrated with a theorem proving system for handling property-based abstractions. Their approach also uses abstract interpretation but develops a framework that uses both over- and under-approximations for handling different kinds of formula.

Our technique for modelling and verifying real time and concurrent systems using constraint logic programs [3] builds on the work of a number of other authors, including Gupta and Pontelli [26], Jaffar *et al.* [31] and Delzanno and Podelski [17]. However we take a different direction from them in our approach to abstraction and checking of temporal properties, in that we use abstract CLP program semantics when abstracting the state space (only briefly mentioned 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 [8, 34, 37, 18, 21, 38, 39]. 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, but sound abstraction of negation in logic programming remains a significant complication in such an approach.

8 Conclusion

We have demonstrated a practical approach to abstract model checking, by constructing an abstract semantic function for the μ-calculus based on a Galois connection. Much 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 [25, 16], a dual abstract transition system is constructed. Like Cousot and Cousot [15] we do not find it necessary to construct any abstract transition system, but

rather 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, César Sánchez, Kim Guldstrand Larsen and suggestions by the LPAR-16 referees.

References

1. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*, pages 971–984, 2000.
2. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
3. 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.
4. G. Banda and J. P. Gallagher. Constraint-based abstraction of a model checker for infinite state systems. In A. Wolf and U. Geske, editors, *Proceedings of the 23rd Workshop on (Constraint) Logic Programming*. University of Potsdam (online Technical Report series), 2009.
5. 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.
6. F. Benoy and A. King. Inferring argument size relationships with CLP(R). In J. P. Gallagher, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'96)*, volume 1207 of *Springer-Verlag Lecture Notes in Computer Science*, pages 204–223, August 1996.
7. 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.
8. C. Brzoska. Temporal logic programming in dense time. In *ILPS*, pages 303–317. MIT Press, 1995.
9. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
10. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *POPL*, pages 342–354, 1992.
11. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
12. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles*, pages 238–252, 1977.
13. 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.

14. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Autom. Softw. Eng.*, 6(1):69–95, 1999.
15. P. Cousot and R. Cousot. Temporal abstract interpretation. In *POPL'2000*, pages 12–25, 2000.
16. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.
17. G. Delzanno and A. Podelski. Model checking in CLP. In *TACAS*, pages 223–239, 1999.
18. X. Du, C. R. Ramakrishnan, and S. A. Smolka. Real-time verification techniques for untimed systems. *Electr. Notes Theor. Comput. Sci.*, 39(3), 2000.
19. 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.
20. E. A. Emerson. Model checking and the mu-calculus. In N. Immerman and P. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 185–214. American Mathematical Society, 1996.
21. 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.
22. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.
23. P. Ganty. *The Fixpoint Checking Problem: An Abstraction Refinement Perspective.* PhD thesis, Université Libre de Bruxelles, Département d'Informatique, 2007.
24. 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.
25. 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.
26. 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.
27. 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.
28. K. S. Henriksen, G. Banda, and J. P. Gallagher. Experiments with a convex polyhedral analysis tool for logic programs. In *Workshop on Logic Programming Environments, Porto*, 2007.
29. 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.
30. M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge University Press, 2000.

31. 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.
32. 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.
33. 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.
34. 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.
35. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
36. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., 1999.
37. 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.
38. 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.
39. 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.
40. H. Saïdi and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 1999.
41. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.