Symbolic Model Checking for Rectangular Hybrid Systems*

Thomas A. Henzinger and Rupak Majumdar

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley, CA 94720
{tah,rupak}@eecs.berkeley.edu

Abstract. An important case of hybrid systems are the rectangular automata. First, rectangular dynamics can naturally and arbitrarily closely approximate more general, nonlinear dynamics. Second, rectangular automata are the most general type of hybrid systems for which model checking –in particular, LTL model checking– is decidable. However, on one hand, the original proofs of decidability did not suggest practical algorithms and, on the other hand, practical symbolic model-checking procedures –such as those implemented in HYTECH– were not known to terminate on rectangular automata. We remedy this unsatisfactory situation: we present a symbolic method for LTL model checking which can be performed by HYTECH and is guaranteed to terminate on all rectangular automata. We do so by proving that our method for symbolic LTL model checking terminates on an infinite-state transition system if the trace-equivalence relation of the system has finite index, which is the case for all rectangular automata.

1 Introduction

The *hybrid automaton* [1] is a mathematical model for dynamical systems with mixed discrete-continuous dynamics. Model checking has been successfully applied to hybrid automaton specifications in automotive [30,32], aerospace [28,29], consumer electronics [26], plant control [25], and robotics [11] applications.

The maximal class of hybrid automata with a decidable model-checking problem is the class of *rectangular automata*[1]: in [22] it is shown that linear temporal logic (LTL) requirements can be checked for rectangular automata, while various minor generalizations of rectangular automata have formally undecidable reachability problems. The rectangular-automaton case is of practical significance, as hybrid systems with very general dynamics can be locally approximated arbitrarily closely using rectangular dynamics [20], which has the form $\dot{\mathbf{x}} \in \prod_{i=0}^{n} [a_i, b_i]$,

* This research was supported in part by the DARPA (NASA) grant NAG2-1214, the DARPA (Wright-Patterson AFB) grant F33615-C-98-3614, the MARCO grant 98-DT-660, the ARO MURI grant DAAH-04-96-1-0341, and the NSF CAREER award CCR-9501708.

[1] In this paper, we refer as "rectangular automata" to the *initialized* rectangular automata of [22].

S. Graf and M. Schwartzbach (Eds.): TACAS/ETAPS 2000, LNCS 1785, pp. 142–156, 2000.

constraining the time derivative $\dot{\mathbf{x}}$ of a state in $\mathbb{R}^n$ to the n-dimensional rectangle $\prod_{i=0}^{n}[a_i, b_i]$ with rational corner points. The decidability proof of [22], however, does not yield a practical LTL model-checking algorithm (and has never been implemented), because it involves a reduction from a rectangular automaton of dimension n to a timed automaton of dimension $2n$, and dimension (i.e., number of clocks) is the most common bottleneck in timed analysis [13].

For practical applications, the tool HYTECH [19] can be used for checking LTL requirements of rectangular automata. Instead of translating a given rectangular automaton H into a timed automaton, HYTECH performs a symbolic computation directly on the n-dimensional state space of H. However, the symbolic procedures employed by HYTECH may not terminate, and thus do not qualify as decision procedures. In this paper, we resolve the gap between theory ([22]) and practice (HYTECH) by showing how given a rectangular automaton H and an LTL formula φ, we can run a symbolic procedure on the state space of H (using the primitives of HYTECH) which is guaranteed to terminate and, upon termination, returns the states of H that satisfy φ. We thus obtain a *symbolic* (rather than reductive) model-checking *algorithm* (rather than semi-algorithm) for LTL requirements of rectangular automata.

We obtain our result by first studying symbolic procedures for LTL model checking in a very general setting (Section 2, 3 and 4), namely, for arbitrary (infinite-state) transition systems with a computable *Pre* operator, which given a set of states, returns the set of predecessor states. We identify a symbolic LTL model-checking procedure based on the *Pre* operator, and a structural (syntax-independent) condition for transition systems (finite trace equivalence) which guarantees termination of the procedure. Since trace equivalence has finite index for all rectangular automata [22], we conclude that symbolic LTL model-checking terminates for rectangular automata. We illustrate our algorithm as applied to a rectangular automaton specifying a physical scheduling problem (Section 5).

Our symbolic LTL model-checking procedure executes a μ-calculus expression which is obtained from a given LTL formula. It is well-known that LTL can be translated into the μ-calculus [6,12,16], and it has been observed that the resulting μ-calculus expressions have a special form [15]: each conjunction has at least one argument which is atomic and constant (i.e., contains no fixpoint operators or variables). This leads us to define the following procedure, called *observation refinement* ($\mathcal{A}_{OR}$): starting from a finite initial partition of the state space, iteratively compute new sets of states by applying either the *Pre* operator, or intersection with an initial set. We show that $\mathcal{A}_{OR}$ terminates on a transition system (i.e., finds only a *finite* number of sets) if and only if the system has a trace-equivalence relation of finite index. Moreover, $\mathcal{A}_{OR}$ termination is a sufficient condition for termination of the μ-calculus based symbolic model-checking algorithm for LTL. Finally, we show that the μ-calculus based algorithm is, in a strong sense, equivalent to the standard, product-automaton based algorithm for symbolic LTL model checking [9].

Thus, $\mathcal{A}_{OR}$ plays with respect to LTL a role that is similar to the role of partition refinement ($\mathcal{A}_{PR}$), which iterates *Pre*, (unrestricted) intersection, and

set difference, with respect to branching-time logics: the termination of $\mathcal{A}_{PR}$ on a transition system guarantees that symbolic model checking for the full μ-calculus (or CTL, CTL*) also terminates. This is because $\mathcal{A}_{PR}$ computes the bisimilarity quotient [27]. While $\mathcal{A}_{PR}$ is known to terminate on timed automata [2], whose time-abstract bisimilarity quotients are finite, there are rectangular automata on which $\mathcal{A}_{PR}$ does not terminate [17]. However, since rectangular automata have finite trace-equivalence quotients, $\mathcal{A}_{OR}$ terminates on every rectangular automaton, thus enabling symbolic LTL model checking.

2 Symbolic Model Checking for Infinite-State Systems

2.1 Transition Structures

A *transition structure* $\mathcal{K} = (Q, \Pi, \langle\langle\cdot\rangle\rangle, \delta)$ consists of a (possibly infinite) set Q of states, a finite set Π of observables, an observation function $\langle\langle\cdot\rangle\rangle\colon Q \to 2^\Pi$ which maps each state to a set of observables, and a transition function $\delta\colon Q \to 2^Q$ which maps each state to a nonempty set of possible successor states. We say that an observable π *holds* at a state q if $\pi \in \langle\langle q \rangle\rangle$. A state q is a *successor* of a state p if $q \in \delta(p)$. A *source-q_0 run* of $\mathcal{K}$ is an infinite sequence $r = q_0 q_1 q_2 \ldots$ of states such that q_{i+1} is a successor of q_i for all $i \geq 0$. The run r induces a *trace*, denoted $\langle\langle r \rangle\rangle$, which is the infinite sequence $\langle\langle q_0 \rangle\rangle \langle\langle q_1 \rangle\rangle \langle\langle q_2 \rangle\rangle \ldots$ of observable sets. For a state $q \in Q$, the *outcome* L^q from q is the set of all runs of $\mathcal{K}$ with source q. For a set L of runs, we write $\langle\langle L \rangle\rangle$ for the set $\{\langle\langle r \rangle\rangle \mid r \in L\}$ of corresponding traces.

A binary relation $\preceq^l \subseteq Q \times Q$ is a *trace containment* if $p \preceq^l q$ implies $\langle\langle L^p \rangle\rangle \subseteq \langle\langle L^q \rangle\rangle$. Define $p \preceq^L q$ if there exists a trace containment $\preceq^l$ with $p \preceq^l q$. Define the *trace-equivalence* relation $\cong^L$ as $p \cong^L q$ if both $p \preceq^L q$ and $q \preceq^L p$. A binary relation $\preceq^s \subseteq Q \times Q$ is a *simulation* if $p \preceq^s q$ implies (1) $\langle\langle p \rangle\rangle = \langle\langle q \rangle\rangle$, and (2) for all states $p' \in \delta(p)$, there exists a state $q' \in \delta(q)$ such that $p' \preceq^s q'$. A binary relation $\cong^b$ on Q is a *bisimulation* if $\cong^b$ is a symmetric simulation. Define $p \cong^B q$ if there is a bisimulation $\cong^b$ with $p \cong^b q$. The equivalence relation $\cong^B$ is called *bisimilarity*.

The observables induce an equivalence relation $\cong^A$, called *atomic equivalence*, on the states Q, with $p \cong^A q$ iff $\langle\langle p \rangle\rangle = \langle\langle q \rangle\rangle$. The equivalence classes of $\cong^A$ are called *atomic regions*. Let A denote the set of atomic regions. For an equivalence $\cong$ on the states Q which refines $\cong^A$, define $\mathcal{K}/_\cong = (Q/_\cong, \Pi, \langle\langle\cdot\rangle\rangle_\cong, \delta_\cong)$, the *quotient structure of $\mathcal{K}$ with respect to $\cong$*, as follows. The states in $Q/_\cong$ are the equivalence classes of $\cong$. The observables are the same as those of $\mathcal{K}$. Define the observation function $\langle\langle\cdot\rangle\rangle_\cong$ as $\pi \in \langle\langle R \rangle\rangle_\cong$ if $\pi \in \langle\langle q \rangle\rangle$ for any/all states $q \in R$. Define the transition function $\delta_\cong$ as $R \in \delta_\cong(P)$ if there are a state $p \in P$ and a state $q \in R$ such that $q \in \delta(p)$.

2.2 Symbolic Semi-algorithms

Let $\mathcal{K}$ be a transition structure. A *region* is a (possibly infinite) set of states of $\mathcal{K}$. If the state space of $\mathcal{K}$ is infinite, any algorithm that traverses the state

space must represent regions implicitly, as formulas in some constraint system. With a transition structure $\mathcal{K}$ we associate a *symbolic theory* [23], which consists of (1) a set Σ of *region representatives* containing finite representations of some regions of $\mathcal{K}$, and (2) an extension function $\ulcorner\cdot\urcorner : \Sigma \to 2^Q$ which maps each region representative to the region it represents, such that the following conditions are satisfied:

- For every atomic region $R \in A$, there is a region representative $\sigma_R \in \Sigma$ such that $\ulcorner\sigma_R\urcorner = R$. Let $\Sigma_A = \{\sigma_R \mid R \in A\}$ denote the set of region representatives for the atomic regions.
- For every region representative $\sigma \in \Sigma$, there is a region representative $\mathit{Pre}(\sigma) \in \Sigma$ such that $\ulcorner \mathit{Pre}(\sigma)\urcorner = \{q \in Q \mid \delta(q) \cap \ulcorner\sigma\urcorner \neq \emptyset\}$; furthermore, the function Pre: $\Sigma \to \Sigma$ can be computed algorithmically.
- For every pair of region representatives $\sigma, \tau \in \Sigma$, there are region representatives $\mathit{And}(\sigma,\tau), \mathit{Diff}\,(\sigma,\tau) \in \Sigma$ such that $\ulcorner \mathit{And}(\sigma,\tau)\urcorner = \ulcorner\sigma\urcorner \cap \ulcorner\tau\urcorner$ and $\ulcorner \mathit{Diff}\,(\sigma,\tau)\urcorner = \ulcorner\sigma\urcorner \setminus \ulcorner\tau\urcorner$; furthermore, the functions And, Diff: $\Sigma \times \Sigma \to \Sigma$ can be computed algorithmically.
- The emptiness of a region representative is decidable; that is, there is a computable function Empty: $\Sigma \to \mathbb{B}$ such that $\mathit{Empty}(\sigma)$ iff $\ulcorner\sigma\urcorner = \emptyset$.
- The membership problem for a state and a region representative is decidable; that is, given a state q and a region representative σ, it can be decided if $q \in \ulcorner\sigma\urcorner$.

A *symbolic semi-algorithm* takes as input the symbolic theory for a transition structure $\mathcal{K}$, and generates region representatives in Σ by applying the operations Pre, And, Diff, and Empty to the atomic region representatives in Σ_A. The expression "semi-algorithm" indicates that, while each operation is computable, the iteration of operations may or may not terminate. Two examples of symbolic semi-algorithms are well-known. The first is *backward reachability*, denoted $\mathcal{A}_\Diamond$. Given an atomic region representative $\alpha \in \Sigma_A$, the symbolic semi-algorithm $\mathcal{A}_\Diamond$ starts from $\sigma_0 = \alpha$ and computes inductively the region representatives $\sigma_{i+1} = \mathit{Pre}(\sigma_i)$. The semi-algorithm terminates if there is a k such that $\bigcup_{0\le i\le k+1} \ulcorner\sigma_i\urcorner \subseteq \bigcup_{0\le i\le k} \ulcorner\sigma_i\urcorner$; that is, no new state is encountered. Termination can be detected using the operations Diff and Empty [23]. Upon termination, a state q can reach the atomic region $\ulcorner\alpha\urcorner$ iff $q \in \ulcorner\sigma_i\urcorner$ for some $1 \le i \le k$.

The second example is *partition refinement* [7,27], denoted $\mathcal{A}_{PR}$. The symbolic semi-algorithm $\mathcal{A}_{PR}$ starts from the finite set $\mathcal{S}_0 = \Sigma_A$ of atomic region representatives and computes inductively the finite sets

$$\mathcal{S}_{i+1} \;=\; \mathcal{S}_i \cup \{\mathit{Pre}(\sigma), \mathit{And}(\sigma,\tau), \mathit{Diff}\,(\sigma,\tau) \mid \sigma,\tau \in \mathcal{S}_i\}$$

of region representatives. The semi-algorithm terminates if there is a k such that $\{\ulcorner\sigma\urcorner \mid \sigma \in \mathcal{S}_{k+1}\} \subseteq \{\ulcorner\sigma\urcorner \mid \sigma \in \mathcal{S}_k\}$; that is, no new region is encountered. Termination can be detected using the operations Diff and Empty: for each region representative $\sigma \in \mathcal{S}_{k+1}$ check that there is a region representative $\tau \in \mathcal{S}_k$ such that both $\mathit{Empty}(\mathit{Diff}\,(\sigma,\tau))$ and $\mathit{Empty}(\mathit{Diff}\,(\tau,\sigma))$. Upon termination, two states p and q are bisimilar iff for all region representatives $\sigma \in \mathcal{S}_k$, we have

$p \in \ulcorner\sigma\urcorner$ iff $q \in \ulcorner\sigma\urcorner$. Thus, the symbolic semi-algorithm $\mathcal{A}_{PR}$ terminates iff the bisimilarity relation $\cong^B$ has finite index [18], as is the case, for instance, for timed automata [2].

2.3 Symbolic Model Checking

A *state logic* $\mathcal{L}$ is a logic whose formulas are interpreted over the states of transition structures. For a formula φ of $\mathcal{L}$ and a transition structure $\mathcal{K}$, let $[\![\varphi]\!]_\mathcal{K}$ be the set of states of $\mathcal{K}$ that satisfy φ. The $\mathcal{L}$ *model-checking problem* asks, given an $\mathcal{L}$-formula φ, a transition structure $\mathcal{K}$, and a state q of $\mathcal{K}$, whether $q \in [\![\varphi]\!]_\mathcal{K}$. A logic $\mathcal{L}$ induces an equivalence relation $\cong^\mathcal{L}$ on states: for all states p and q of a transition structure $\mathcal{K}$, define $p \cong^\mathcal{L} q$ if for all $\mathcal{L}$-formulas φ, we have $p \in [\![\varphi]\!]_\mathcal{K}$ iff $q \in [\![\varphi]\!]_\mathcal{K}$. Thus, two states p and q of a transition structure $\mathcal{K}$ are equivalent with respect to $\cong^\mathcal{L}$ iff there is no formula in the logic $\mathcal{L}$ that can distinguish p from q. Two formulas φ and ψ of state logics are *equivalent* if $[\![\varphi]\!]_\mathcal{K} = [\![\psi]\!]_\mathcal{K}$ for all transition structures $\mathcal{K}$. The logic $\mathcal{L}_1$ is *as expressive as* the logic $\mathcal{L}_2$ if for every formula ψ of $\mathcal{L}_2$, there exists a formula φ of $\mathcal{L}_1$ equivalent to ψ. The logics $\mathcal{L}_1$ and $\mathcal{L}_2$ are *equally expressive* if $\mathcal{L}_1$ is as expressive as $\mathcal{L}_2$, and $\mathcal{L}_2$ is as expressive as $\mathcal{L}_1$.

A state logic $\mathcal{L}$ *admits abstraction* if for every equivalence relation $\cong$ that refines $\cong^\mathcal{L}$, for every $\mathcal{L}$-formula φ, and for every transition structure $\mathcal{K}$, the region $[\![\varphi]\!]_\mathcal{K}$ is $\bigcup [\![\varphi]\!]_{\mathcal{K}/\cong}$. If $\mathcal{L}$ admits abstraction, and $\cong$ refines $\cong^\mathcal{L}$, then $\cong$ is called an *abstract semantics* for $\mathcal{L}$; if $\mathcal{L}$ admits abstraction, then $\cong^\mathcal{L}$ is the *fully abstract* semantics for $\mathcal{L}$. Let $\mathcal{L}$ be a logic that admits abstraction, and let $\cong$ be an abstract semantics for $\mathcal{L}$. Then a state p of $\mathcal{K}$ satisfies an $\mathcal{L}$-formula φ iff the $\cong$-equivalence class containing p satisfies φ in the quotient structure $\mathcal{K}/\cong$. This means that instead of model checking the structure $\mathcal{K}$, we can model check the quotient structure $\mathcal{K}/\cong$. In case the equivalence relation $\cong$ has finite index, we can so reduce model-checking questions over an infinite-state structure to model-checking questions over a finite-state structure.

A simple state logic of interest is the logic EFL, which contains all formulas of the form $\exists\Diamond\varphi$, where φ is a boolean combination of observables. The formula $\exists\Diamond\varphi$ *holds* at a state q of a transition structure $\mathcal{K}$ if there exists a source-q run r of $\mathcal{K}$, and a state p in r, such that φ holds in p. A model-checking algorithm for the logic EFL is easily derived from the symbolic semi-algorithm $\mathcal{A}_\Diamond$ (backward reachability). In particular, if $\mathcal{A}_\Diamond$ terminates on every atomic region representative of $\mathcal{K}$, then EFL model checking can be decided over $\mathcal{K}$. The logic EFL can express reachability (or dually, safety) properties. To express more interesting properties, we define the *μ-calculus*, which can encode temporal logics such as LTL, CTL, and CTL* [14]. The formulas of the μ-calculus are given by the grammar

$$\varphi \ ::= \ \pi \mid \neg\pi \mid X \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \exists\bigcirc\varphi \mid \forall\bigcirc\varphi \mid \mu X.\,\varphi \mid \nu X.\,\varphi,$$

where π is an observable, X is a propositional variable, μ is the least-fixpoint operator, and ν is the greatest-fixpoint operator. We interpret closed formulas

over states in the standard way [14]. For example, the EFL formula $\exists\Diamond\pi$ is equivalent to the μ-calculus formula $\mu X.(\pi \vee \exists\bigcirc X)$.

The μ-calculus admits abstraction, and bisimilarity $\cong^B$ is a fully abstract semantics for the μ-calculus. Thus, if an infinite-state transition structure $\mathcal{K}$ with a symbolic theory has a finite bisimilarity quotient $\mathcal{K}/_{\cong^B}$, or equivalently, if the symbolic semi-algorithm $\mathcal{A}_{PR}$ (partition refinement) terminates on $\mathcal{K}$, then μ-calculus model checking can be decided over $\mathcal{K}$: first, use partition refinement to compute the finite-state structure $\mathcal{K}/_{\cong^B}$; then, model check over $\mathcal{K}/_{\cong^B}$. There is, however, also a more direct, more efficient way of μ-calculus model checking over infinite-state transition structures with symbolic theories: we can attempt to compute fixpoints by successive approximation [8], using the operations *Pre*, *And*, and *Diff* [18]. If the successive approximation of each fixpoint subformula of a μ-calculus formula φ terminates in a finite number of steps, then we arrive at the region $[\![\varphi]\!]_{\mathcal{K}}$ in a finite number of applications of *Pre*, *And*, and *Diff*. Clearly, a sufficient condition is the termination of $\mathcal{A}_{PR}$, which applies all possible combinations of the three operations. The symbolic semi-algorithm for μ-calculus model checking, which performs only the subset of operations of $\mathcal{A}_{PR}$ called for by the input formula, is denoted $\mathcal{A}_{\mu}$. For example, for the EFL formula $\exists\Diamond\pi$, the semi-algorithm $\mathcal{A}_{\mu}$ is identical to $\mathcal{A}_{\Diamond}$.

3 A Symbolic Characterization of Trace Equivalence

3.1 Observation Refinement

We define a symbolic semi-algorithm, called *observation refinement* and denoted $\mathcal{A}_{OR}$, which repeatedly applies the two operations *Pre* and intersection with atomic region representatives, until no new regions can be generated. The symbolic semi-algorithm $\mathcal{A}_{OR}$ starts from the finite set $\mathcal{S}_0 = \Sigma_A$ of atomic region representatives and computes inductively the finite sets

$$\mathcal{S}_{i+1} \;=\; \mathcal{S}_i \cup \{\mathit{Pre}(\sigma), \mathit{And}(\sigma,\tau) \mid \sigma \in \mathcal{S}_i \text{ and } \tau \in \Sigma_A\}$$

of region representatives. Note that only a restricted form of intersection (intersection with atomic regions) is allowed. The semi-algorithm terminates if there is a k such that $\{\ulcorner\sigma\urcorner \mid \sigma \in \mathcal{S}_{k+1}\} \subseteq \{\ulcorner\sigma\urcorner \mid \sigma \in \mathcal{S}_k\}$; this is checked as in the case of $\mathcal{A}_{PR}$ (partition refinement). Observation refinement will typically produce more region representatives than $\mathcal{A}_{\Diamond}$ (backward reachability), but fewer than $\mathcal{A}_{PR}$. In particular, there are infinite-state transition structures on which backward reachability terminates, but not observation refinement; and structures on which observation refinement terminates, but not partition refinement.

3.2 The Guarded Fragment of the μ-Calculus

For a logical characterization of the regions computed by observation refinement, we define Gμ, the *guarded fragment of the μ-calculus*, as the set of formulas given by the following rules:

1. All observables and propositional variables are formulas of Gμ.
2. If π is an observable, then $\neg\pi$ is a formula of Gμ.
3. If φ_1 and φ_2 are formulas of Gμ, then
   (a) $\varphi_1 \vee \varphi_2$, $\exists\bigcirc \varphi_1$, $\mu X.\ \varphi_1$, and $\nu X.\ \varphi_1$ are formulas of Gμ.
   (b) $\varphi_1 \wedge \varphi_2$ is a formula of Gμ provided at least one of φ_1 and φ_2 is a boolean combination of observables.

This definition is similar to the definition of L_1 in [5,15].[2] Over finite-state transition structures, there is a fast, $O(mnk)$ model-checking algorithm for Gμ, where m is the size of the transition structure, n is the size of the formula, and k is the alternation depth of the formula [5].

Proposition 1. *The guarded fragment of the μ-calculus admits abstraction.*

The equivalence relation $\cong^{\mathrm{G}\mu}$ induced by the guarded fragment of the μ-calculus is characterized operationally by observation refinement. By induction, each region computed in step i of the symbolic semi-algorithm $\mathcal{A}_{OR}$ is a block (i.e., a union of equivalence classes) of $\cong^{\mathrm{G}\mu}$. Thus, if $\cong^{\mathrm{G}\mu}$ has finite index, then $\mathcal{A}_{OR}$ terminates. Conversely, suppose that $\mathcal{A}_{OR}$ terminates with $\mathcal{S}_{k+1} = \mathcal{S}_k$. We can show that if two states are distinguished by a formula in Gμ, then there is a region constructed by $\mathcal{A}_{OR}$ that separates them. Define the state equivalence $\cong^{OR}$ as $p \cong^{OR} q$ iff for each region representative $\sigma \in \mathcal{S}_k$, we have $p \in \ulcorner\sigma\urcorner$ iff $q \in \ulcorner\sigma\urcorner$. It follows that $p \cong^{OR} q$ implies $p \cong^{\mathrm{G}\mu} q$.

Proposition 2. *Observation refinement ($\mathcal{A}_{OR}$) terminates on the symbolic theory of a transition structure $\mathcal{K}$ iff the equivalence relation $\cong^{\mathrm{G}\mu}$ induced by the guarded fragment of the μ-calculus on $\mathcal{K}$ has finite index.*

3.3 Expressiveness of the Guarded Fragment

We can alternatively characterize the expressiveness of Gμ using a linear-time logic (without path quantifiers). A *Büchi automaton* $\mathcal{B}$ is a tuple $(S, \Phi, \rightarrow, s_0, F)$, where S is a finite set of states, Φ is a finite input alphabet, $\rightarrow \subseteq S \times \Phi \times S$ is the transition relation, $s_0 \in S$ is the start state, and $F \subseteq S$ is the set of Büchi accepting states. An *execution* of $\mathcal{B}$ on an ω-word $w = w_0 w_1 \ldots \in \Phi^\omega$ is an infinite sequence $r = s_0 s_1 \ldots$ of states in S, starting from the initial state s_0, such that $s_i \xrightarrow{w_i} s_{i+1}$ for all $i \geq 0$. The execution r is *accepting* if some state in F occurs infinitely often in r. The automaton $\mathcal{B}$ *accepts* the word w if it has an accepting execution on w. The *language* $L(\mathcal{B}) \subseteq \Phi^\omega$ is the set of ω-words accepted by $\mathcal{B}$.

Let EBL be the state logic whose formulas have the form $\exists\mathcal{B}$, where $\mathcal{B}$ is a Büchi automaton whose input alphabet are sets of observables; that is, $\Phi = 2^{\Pi}$.

[2] In [15], condition (2) of our definition is changed to "If φ is an L_1 formula that does not contain any variables, then $\neg\varphi$ is in L_1," and condition (3b) is changed to "If φ_1 and φ_2 are L_1 formulas such that at most one of them contains any variables, then $\varphi_1 \wedge \varphi_2$ is in L_1." It can be shown that L_1 and Gμ are equally expressive.

The formula $[\![\exists\mathcal{B}]\!]_{\mathcal{K}}$ *holds* at a state q of a transition structure $\mathcal{K}$ if there exists a source-q run r of $\mathcal{K}$ such that $\langle\!\langle r \rangle\!\rangle \in L(\mathcal{B})$. The logic EBL admits abstraction, and trace equivalence is a fully abstract semantics for EBL. Now we show that EBL is as expressive as the guarded fragment of the μ-calculus. This result is implicit in a proof in [15], although it is never explicitly stated.

Lemma 1. *Given a* Gμ *formula* φ *(over the observables* Π*), we can construct a Büchi automaton* $\mathcal{B}_\varphi$ *(on the alphabet* 2^Π*) such that* $[\![\exists\mathcal{B}_\varphi]\!]_{\mathcal{K}} = [\![\varphi]\!]_{\mathcal{K}}$ *for all transition structures* $\mathcal{K}$*. Conversely, given an* EBL *formula* $\exists\mathcal{B}$ *(over the alphabet* 2^Π*), we can construct an* Gμ *formula* $\varphi_\mathcal{B}$ *(over the observables* Π*) so that* $[\![\varphi_\mathcal{B}]\!]_{\mathcal{K}} = [\![\exists\mathcal{B}]\!]_{\mathcal{K}}$ *for all transition structures* $\mathcal{K}$*.*

The proof of the first part of the lemma proceeds by induction on the structure of the Gμ formula. For the converse claim, let $\mathcal{B}$ be a Büchi automaton. We construct a guarded μ-calculus formula that is equivalent to the formula $\exists\mathcal{B}$. For notational convenience, we present the formula in equational form [10]; it can be easily converted to the standard representation by unrolling the equations, and binding variables with μ or ν-fixpoints. For each set $R \in 2^\Pi$, let ψ_R abbreviate the formula $\bigwedge R \wedge \bigwedge\{\neg\pi \mid \pi \in \Pi\backslash R\}$. For each state s of $\mathcal{B}$, we introduce a propositional variable X_s. The equation for X_s is

$$X_s \;=_\lambda\; \bigvee\{\psi_R \wedge \exists\bigcirc X_{s'} \mid s \xrightarrow{R} s'\},$$

where $\lambda = \nu$ if $s \in F$ is an accepting state, and $\lambda = \mu$ otherwise. The top-level variable is X_{s_0}, where s_0 is the initial state. The correctness of the procedure follows from [6]. An equivalent construction is given in [12].

Theorem 1. *The logics* Gμ *and* EBL *are equally expressive.*

From Proposition 2, and since the fully abstract semantics of EBL is trace equivalence, we conclude the following.

Corollary 1. *Observation refinement (*$\mathcal{A}_{OR}$*) terminates on the symbolic theory of a transition structure* $\mathcal{K}$ *iff the trace-equivalence relation* $\cong^L$ *of* $\mathcal{K}$ *has finite index.*

When the symbolic semi-algorithm $\mathcal{A}_\mu$ for μ-calculus model checking is applied to an input in guarded form, then it computes only regions also computed by observation refinement. It follows that symbolic model checking for the guarded fragment of the μ-calculus terminates on all transition structures with finite trace-equivalence quotients.

4 Symbolic LTL Model Checking

4.1 Mu-Calculus Based Symbolic Model Checking for LTL

The formulas of *linear temporal logic* (LTL) are generated inductively by the grammar

$$\varphi \;::=\; \pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2,$$

where π is an observable, $\bigcirc$ is the *next* operator, and $\mathcal{U}$ is the *until* operator. From these operators, additional operators such as $\Diamond\varphi \stackrel{\Delta}{=} (true \mathcal{U} \varphi)$ and $\Box\varphi \stackrel{\Delta}{=} \neg\Diamond\neg\varphi$ can be defined as usual. Formulas of LTL are interpreted over traces [14]. We extend the interpretation to states *existentially*: an LTL formula φ *holds* at a state q of a transition structure $\mathcal{K}$ if there is a source-q run r such that the trace $\langle\langle r \rangle\rangle$ satisfies the formula φ. The logic LTL admits abstraction, and trace equivalence is a fully abstract semantics for LTL. The expressiveness of LTL lies strictly between EBL and EFL. In particular, for every LTL formula φ, we can construct a Büchi automaton $\mathcal{B}_\varphi$ such that $[\![\varphi]\!]_\mathcal{K} = [\![\exists \mathcal{B}_\varphi]\!]_\mathcal{K}$ for all transition structures $\mathcal{K}$ [31]. We call $\mathcal{B}_\varphi$ the *tableau automaton* of φ.

This suggests the following symbolic semi-algorithm $\mathcal{A}^1_{\text{LTL}}$, the *$\mu$-calculus based algorithm for* LTL *model checking*: given an LTL formula φ, first construct the tableau automaton $\mathcal{B}_\varphi$, then convert $\exists\mathcal{B}_\varphi$ into the guarded fragment of the μ-calculus (using the procedure described above), and finally evaluate the resulting Gμ formula on the given transition structure (using $\mathcal{A}_\mu$). The final step requires only *Pre* operations and intersections with atomic regions.

Theorem 2. *For a transition structure $\mathcal{K}$ with a symbolic theory, and an* LTL *formula φ, the symbolic semi-algorithm $\mathcal{A}^1_{\text{LTL}}$ terminates and computes $[\![\varphi]\!]_\mathcal{K}$ if the trace-equivalence relation $\cong^L$ of $\mathcal{K}$ has finite index.*

4.2 Product-Automaton Based Symbolic Model Checking for LTL

Traditionally, a different method is used for symbolic model checking of LTL formulas [9]. Given a state q of a finite-state transition structure $\mathcal{K}$, and an LTL formula φ, the question if $q \in [\![\varphi]\!]_\mathcal{K}$ can be answered by constructing the product of $\mathcal{K}$ with the tableau automaton $\mathcal{B}_\varphi$, and then checking the nonemptiness of a Büchi condition on the product structure. A Büchi condition is an LTL formula of the form $\Box\Diamond\psi$, where ψ is a disjunction of observables; therefore nonemptiness can be checked symbolically by evaluating the equivalent formula

$$\chi \;=\; \nu X_1.\, \mu X_2.\, (\exists\bigcirc X_2 \vee (\psi \wedge \exists\bigcirc X_1))$$

of the guarded fragment of the μ-calculus.

To extend this method to infinite-state structures, we need to be more formal. Let $\mathcal{K} = (Q, \Pi, \langle\langle\cdot\rangle\rangle, \delta)$ be a transition structure and let $\mathcal{B}_\varphi = (S, 2^\Pi, \rightarrow, s_0, F)$ be a tableau automaton. The *product structure* $\mathcal{K}_\varphi = (S \times Q, S \times \Pi, \langle\langle\cdot\rangle\rangle_\varphi, \delta_\varphi)$ is defined as follows. Define $(s', \pi) \in \langle\langle s, q\rangle\rangle_\varphi$ iff $s' = s$ and $\pi \in \langle\langle q \rangle\rangle$; that is, the state of the tableau automaton is observable. Define $(s', q') \in \delta_\varphi(s, q)$ iff $s \stackrel{R}{\rightarrow} s'$ and $q' \in \delta(q)$ and $\langle\langle q \rangle\rangle = R$. Then $q \in [\![\varphi]\!]_\mathcal{K}$, for $q \in Q$, iff $(s_0, q) \in [\![\Box\Diamond\psi]\!]_{\mathcal{K}_\varphi}$, where $\psi = \bigvee_{s\in F, \pi\in\Pi}(s, \pi)$. To perform symbolic model checking on the product structure, we need to ensure that from a symbolic theory for $\mathcal{K}$ we can obtain a symbolic theory for $\mathcal{K}_\varphi$. Let $(\Sigma, \ulcorner\cdot\urcorner)$ be a symbolic theory for $\mathcal{K}$. We choose as region representatives for the product structure $\mathcal{K}_\varphi$ the pairs of the form (s, σ), where s is a state of $\mathcal{B}_\varphi$ and σ is a region representative for $\mathcal{K}$; that is, $\Sigma_\varphi = S \times \Sigma$. Define $\ulcorner s, \sigma \urcorner_\varphi = \{(s, q) \mid q \in \ulcorner\sigma\urcorner\}$. Since the tableau automaton

$\mathcal{B}_\varphi$ is finite, it is easy to check that $(\Sigma_\varphi, \ulcorner\cdot\urcorner_\varphi)$ is a symbolic theory for $\mathcal{K}_\varphi$. Let $\mathcal{A}^2_{\text{LTL}}$ be the *product-automaton based algorithm for* LTL *model checking* which, given an LTL formula φ and a transition structure $\mathcal{K}$, evaluates the Gμ formula χ (representing a Büchi condition) on the product structure $\mathcal{K}_\varphi$ (using $\mathcal{A}_\mu$). It is not difficult to see that if observation refinement terminates on $\mathcal{K}$ in k steps, then it also terminates on $\mathcal{K}_\varphi$ in k steps (if $\mathcal{A}_{OR}$ generates m regions on $\mathcal{K}$, then it generates at most $m \cdot |S|$ regions on $\mathcal{K}_\varphi$).

Corollary 2. *For a transition structure $\mathcal{K}$ with a symbolic theory, and an* LTL *formula φ, the symbolic semi-algorithm $\mathcal{A}^2_{\text{LTL}}$ terminates and computes $[\![\varphi]\!]_\mathcal{K}$ if the trace-equivalence relation $\cong^L$ of $\mathcal{K}$ has finite index.*

Indeed, by induction on the construction of regions, one can show that for each region representative (s, σ) computed in the product-automaton based algorithm, the variable X_s in the μ-calculus based algorithm represents the region $\ulcorner\sigma\urcorner$ at some stage of the computation, and conversely, for each valuation R of the variable X_s in the μ-calculus based algorithm, a region representative of $\{s\} \times R$ is computed in the product-automaton based algorithm. Thus, the two methods are equivalent in the regions they generate.

5 Rectangular Hybrid Automata

5.1 Definitions

Let $\mathbb{R}^n$ be the n-dimensional Euclidean space. A *rectangle* $\mathfrak{r}$ *of dimension* n is a subset of $\mathbb{R}^n$ which is a cartesian product of (possibly unbounded) intervals, all of whose finite end-points are integral[3]. The projection of a rectangle $\mathfrak{r}$ on its ith coordinate is denoted $\mathfrak{r}_i$, so that $\mathfrak{r} = \prod_{i=1}^n \mathfrak{r}_i$. The set of all n-dimensional rectangles is denoted $\Re^n$.

An *n-dimensional rectangular automaton* H consists of a finite directed multi-graph (V, E), three vertex labeling functions *init*: $V \to \Re^n$, *inv*: $V \to \Re^n$, and *flow*: $V \to \Re^n$, and three edge labeling functions *pre*: $E \to \Re^n$, *post*: $E \to \Re^n$, and *jump*: $E \to 2^{\{1,\cdots,n\}}$ [22]. The vertices $\ell \in V$ specify the discrete states of the automaton; the edges $e \in E$ specify the discrete transitions. The initialization function *init* specifies the possible initial states of the automaton. If the automaton starts in vertex ℓ, then its continuous state must be in $\mathit{init}(\ell)$. The invariant function *inv* and the flow function *flow* constrain the continuous time evolution of the automaton. In vertex ℓ, the continuous state nondeterministically follows a smooth trajectory within the invariant region $\mathit{inv}(\ell)$. At each point, the derivative of the trajectory must lie within the flow region $\mathit{flow}(\ell)$. The edges are constrained by the pre-guard function *pre*, the post-guard function *post*, and the jump function *jump*. The edge $e = (\ell, \ell')$ may be traversed when the current vertex is ℓ and the continuous state lies within $\mathit{pre}(e)$. For each $i \in \mathit{jump}(e)$, the ith coordinate of the continuous state is nondeterministically assigned a new value in the postguard interval $\mathit{post}(e)_i$. For each coordinate $i \notin \mathit{jump}(e)$, the

[3] It is straightforward to permit intervals with rational end-points.

continuous state is not changed, and must lie within $post(e)_i$. We require that for every edge $e = (\ell, \ell')$, and every coordinate $i = 1, \ldots, n$, if $flow(\ell)_i \neq flow(\ell')_i$, then $i \in jump(e)$. This condition is called *initialization* in [22], and it is shown there that it is necessary for simple reachability questions to be decidable.

With a rectangular automaton H, we associate an infinite-state transition structure $\mathcal{K}_H = (Q, V, \langle\!\langle \cdot \rangle\!\rangle, \delta)$ as follows. The states in Q are pairs $(\ell, \mathbf{x})$ consisting of a discrete part $\ell \in V$ and a continuous part $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{x} \in inv(v)$. The observables are the vertices, and $\langle\!\langle \ell, \mathbf{x} \rangle\!\rangle = \ell$. We have $(\ell', \mathbf{x}') \in \delta((\ell, \mathbf{x}))$ iff either (1) [time transition of duration t and slope $\mathbf{d}$] $\ell' = \ell$, and $\mathbf{x}' = \mathbf{x} + t \cdot \mathbf{d}$ for some real vector $\mathbf{d} \in flow(\ell)$ and some real $t \geq 0$ such that for all $0 \leq t' \leq t$, $(\mathbf{x} + t' \cdot \mathbf{d}) \in inv(\ell)$; or (2) [discrete transition along edge e] $e = (\ell, \ell') \in E$, and $(\ell, \mathbf{x}) \in pre(e)$, and $\mathbf{x}'_i \in post(e)_i$ for all $i \in jump(e)$, and $\mathbf{x}'_i = \mathbf{x}_i$ for all $i \notin jump(e)$. The runs and traces of H are inherited from the underlying transition structure $\mathcal{K}_H$.

A natural symbolic theory for the rectangular automaton H is the following. Regions are represented as sets $\Sigma = \{(\ell, f) \mid \ell \in V\}$ of pairs, where ℓ is a vertex and f is a quantifier-free formula in the theory of reals with addition, $\mathrm{Th}(\mathbb{R}, 0, 1, +, \leq)$, over the n variables $x_1, \ldots, x_n$. The atomic sentences in this theory are the linear inequalities; thus the continuous part of a region is represented by a boolean combination of linear inequalities. The *Pre* operation can be described in the theory using quantifiers, and since the theory permits quantifier elimination, the quantifier-free formulas suffice as region representatives [4]. The emptiness and membership checks are also decidable. The tool HyTech [19] implements symbolic semi-algorithms for analyzing rectangular (and more general hybrid) automata using this symbolic theory. In particular, the symbolic semi-algorithms $\mathcal{A}_{PR}$, $\mathcal{A}_{OR}$, and $\mathcal{A}_{\mu}$ are all readily programmable using the scripting facility of HyTech.

The variable x_i of the rectangular automaton H is a *finite-slope variable* if for each vertex $\ell \in V$, the interval $flow(\ell)_i$ is a singleton. If $flow(\ell)_i = [1, 1]$ for all vertices ℓ, then x_i is called a *clock*. The rectangular automaton H has *deterministic jumps* if for each edge $e \in E$, and each coordinate $i \in jump(e)$, the interval $post(e)_i$ is a singleton. If H has deterministic jumps and $x_1, \ldots, x_n$ are all finite-slope variables, then H is a *singular automaton*. If H has deterministic jumps and $x_1, \ldots, x_n$ are all clocks, then H is called a *timed automaton* [2].

5.2 Symbolic Model Checking

It can be shown that the bisimilarity relation of the transition structure $\mathcal{K}_H$ has finite index for every singular automaton H [2,1]. This implies that we can check μ-calculus properties symbolically on timed and singular automata [24]. For rectangular automata, dimension 2 is enough to have bisimilarity degenerate into equality on states [17]. However, the results of [22] show that the trace-equivalence quotient of $\mathcal{K}_H$ is finite for every rectangular automaton H. From Corollary 1, we get the following result.

Theorem 3. *Observation refinement ($\mathcal{A}_{OR}$) terminates when applied to the symbolic theory of a rectangular automaton.*

Corollary 3. *Symbolic* Gμ *model checking,* μ*-calculus based symbolic* LTL *model checking, and product-automaton based symbolic* LTL *model checking all terminate for rectangular automata.*

In practice, we are interested in the *divergent runs* of a rectangular automaton, i.e., those runs on which time advances beyond any bound. Formally, a run $(\ell_0, \mathbf{x}_0)(\ell_1, \mathbf{x}_1)\ldots$ of a rectangular automaton is *divergent* if the infinite sum

$$\sum \{t_i \mid i \geq 0 \text{ and } (\ell_{i+1}, \mathbf{x}_{i+1}) \in \delta(\ell_i, \mathbf{x}_i) \text{ is a time step of duration } t_i\}$$

diverges. To restrict our attention to divergent runs, we can modify an n-dimensional rectangular automaton H in a standard way [3]. We add an additional clock variable at coordinate $n+1$, so that the dimension becomes $n+1$. For each vertex $\ell \in V$, we introduce a new vertex ℓ_{tick} and two edges $e = (\ell, \ell_{tick})$ and $e' = (\ell_{tick}, \ell)$. Define $pre(e)_i = post(e) = pre(e')_i = post(e') = \mathbb{R}$ for $1 \leq i \leq n$; $pre(e)_{n+1} = post(e)_{n+1} = pre(e')_{n+1} = 1$, $jump(e)_{n+1} = \emptyset$, $jump(e') = \{n+1\}$, and $post(e')_{n+1} = 0$. This construction ensures that the added clock is reset to 0 every time its value reaches 1. Then, the divergent runs are those for which the formula $\psi = \bigvee_{\ell \in V} \ell_{tick}$ is true infinitely often. To check if an LTL formula φ holds on some divergent run of H, we instead check that the LTL formula $(\Box\Diamond\psi) \wedge \varphi$ holds on any run of the extended automaton.

In [22], the proof that the trace-equivalence relation has finite index for every rectangular automaton H proceeds in two steps. First, the authors construct a singular automaton H' which forward simulates H, and is backward simulated by H. This implies that the trace-equivalence quotient for *finite* traces has finite index. In a second, involved step, they prove that a finite trace-equivalence quotient for finite traces implies a finite trace-equivalence quotient for infinite traces as well. The results of this paper allow a more direct proof, which immediately gives the desired result for infinite traces. It suffices to show that observation refinement ($\mathcal{A}_{OR}$) terminates on the transition structure of H. As in [22], it can be argued that every *Pre* operation on $\mathcal{K}_H$ corresponds in a precise sense to a *Pre* operation on $\mathcal{K}_{H'}$. Since the bisimilarity quotient of $\mathcal{K}_{H'}$ is finite, this implies that *Pre* operations on $\mathcal{K}_H$ can also generate only a finite number of distinct regions. There is only a finite number of observables (corresponding to the vertices of H). So the intersection of a region with an atomic region is simply the projection of the region onto a discrete part. Thus, the restricted intersection operation of $\mathcal{A}_{OR}$ does not give rise to any new continuous parts of regions beyond the ones already computed. It follows that $\mathcal{A}_{OR}$ terminates on $\mathcal{K}_H$. This result is sharp: for simple extensions to the model of rectangular automata, even backward reachability ($\mathcal{A}_{\Diamond}$) may not terminate [22].

5.3 Example: Assembly Line Scheduler

We describe an assembly line scheduler that must assign elements from an incoming stream to one of two assembly lines [21]. The stream has an inter-arrival time of four minutes. The lines process the parts at different speeds: on the first

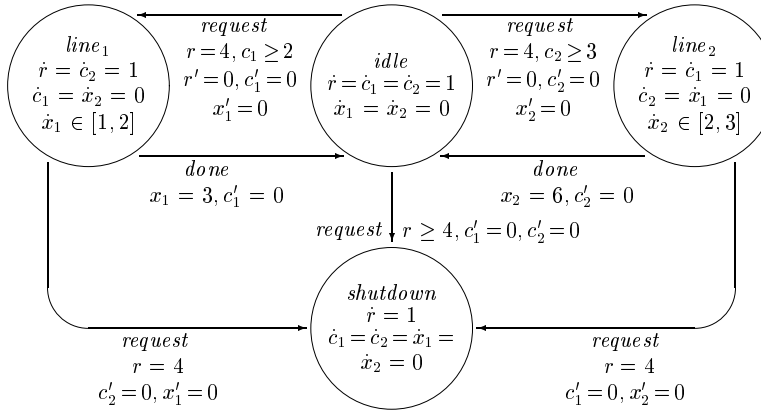

Fig. 1. Two assembly lines modeled as a rectangular automaton

line, jobs travel between one and two meters per minute, while on the second, jobs travel between two and three meters per minute. The first line is three meters long and the second line is six meters long. Once a line finishes processing a job, it enters a clean-up phase, and no jobs may be assigned to it while it cleans up. The clean-up time is two minutes for the first line and three minutes for the second line. The system may accept a job if both lines are free, and at most one is cleaning up. If the system is unable to accept a job, it shuts down.

The system is modeled by a rectangular automaton as shown in Figure 1. There are four discrete states: in *idle*, no jobs are being processed; in $line_1$ ($line_2$), line-1 (respectively, line-2) is processing a job, and in *shutdown*, the system is shut down. The variable x_1 (respectively, x_2) measures the distance a job has traveled along line-1 (respectively, line-2). The variable c_1 (c_2) tracks the amount of time line-1 (line-2) has spent cleaning up after its last job. Finally, the variable r measures the elapsed time since the last arrival of a job.

We modeled the system in HyTech. Backward reachability ($\mathcal{A}_\Diamond$) terminates in set of states that can reach the unsafe vertex *shutdown*. We added a preprocessor to HyTech which takes an Ltl formula and generates a script to evaluate an equivalent Gμ formula. We then considered the property that any feasible schedule must choose line-1 infinitely often. To establish this requirement, we checked that the formula $(\Diamond\Box\neg line_1) \wedge (\Box\neg shutdown)$ does not hold on any divergent run from the vertex *idle* (if this formula were to hold on some divergent run, then there would be a schedule that assigns jobs to $line_1$ only finitely many times, and still enforces that the system never shuts down). This required 0.39 seconds of CPU time on a DEC alpha with 2G RAM.

Acknowledgments

We thank Luca de Alfaro for several useful discussions.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995. 142, 152
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. 144, 146, 152
3. R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer*, 1:86–109, 1997. 153
4. R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual Real-time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993. 152
5. G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *TACAS96: Tools and Algorithms for Construction and Analysis of Systems*, LNCS 1055, pages 107–126. Springer-Verlag, 1996. 148
6. G. Bhat and R. Cleaveland. Efficient model checking via the equational μ-calculus. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 304–312. IEEE Computer Society Press, 1996. 143, 149
7. A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In *CAV 90: Computer-aided Verification*, LNCS 663, pages 197–203. Springer-Verlag, 1990. 145
8. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992. 147
9. E.M. Clarke, O. Grumberg, and D.E. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency: Reflections and Perspectives*, LNCS 803, pages. Springer-Verlag, 1994. 143, 150
10. R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal μ-calculus. In *CAV 92: Computer-aided Verification*, LNCS 663, pages 410–422. Springer-Verlag, 1993. 149
11. J.C. Corbett. Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering*, 22:461–483, 1996. 142
12. M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994. 143, 149
13. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS98: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1384, pages 313–329. Springer-Verlag, 1998. 143
14. E.A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990. 146, 147, 150
15. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model checking for fragments of μ-calculus. In *CAV 93: Computer-aided Verification*, LNCS 697, pages 385–396. Springer-Verlag, 1993. 143, 148, 149
16. E.A. Emerson and C. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science*, pages 267–278. IEEE Computer Society Press, 1986. 143
17. T.A. Henzinger. Hybrid automata with finite bisimulations. In *ICALP 95: Automata, Languages, and Programming*, LNCS 944, pages 324–335. Springer-Verlag, 1995. 144, 152

18. T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996. 146, 147
19. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997. 143, 152
20. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998. 142
21. T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *CONCUR 99: Concurrency Theory*, LNCS 1664, pages 320–335, 1999. 153
22. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998. 142, 143, 151, 152, 153
23. T.A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In *STACS 2000: Theoretical Aspects of Computer Science*, LNCS. Springer-Verlag, 2000. 145
24. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science*, pages 394–406. IEEE Computer Society Press, 1992. 152
25. T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, pages 265–282. Springer-Verlag, 1996. 142
26. P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In *CAV 95: Computer-aided Verification*, LNCS 939, pages 381–394. Springer-Verlag, 1995. 142
27. P.C. Kanellakis and S.A. Smolka. Ccs expressions, finite-state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990. 144, 145
28. J. Kosecka, C.J. Tomlin, G. Pappas, and S. Sastry. Generation of conflict resolution manoeuvres for air traffic management. In *IROS 97: International Conference on Intelligent Robot and Systems*, volume 3, pages 1598–1603. IEEE Press, 1997. 142
29. S. Nadjm-Tehrani and J.-E. Strömberg. Proving dynamic properties in an aerospace application. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 2–10. IEEE Computer Society Press, 1995. 142
30. T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *HART 97: Hybrid and Real-time Systems*, LNCS 1201, pages 139–153. Springer-Verlag, 1997. 142
31. W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990. 150
32. T. Villa, H. Wong-Toi, A. Balluchi, J. Preußig, A. Sangiovanni-Vincentelli, and Y. Watanabe. Formal verification of an automotive engine controller in cutoff mode. In *Proceedings of the 37th Conference on Decision and Control*. IEEE Press, 1998. 142