Real-Time Verification Techniques for Untimed Systems

Xiaoqun Du, C.R. Ramakrishnan, Scott A. Smolka

Department of Computer Science, SUNY at Stony Brook
Stony Brook, NY 11794–4400, USA
{vicdu,cram,sas}@cs.sunysb.edu

Abstract

We show that verification techniques for timed automata based on the Alur and Dill region-graph construction can be applied to much more general kinds of systems, including asynchronous untimed systems over unbounded integer variables. We follow this approach in proving that the model-checking problem for the n-process Bakery algorithm is decidable, for any fixed n. We believe this is the first decidability proof for this problem to appear in the literature.

1 Introduction

In their seminal paper, Alur and Dill [2] showed how the language-emptiness problem for timed automata can be decided by partitioning the infinite state space underlying a timed automaton into a finite collection of *regions*. The resulting structure is referred to as a *region graph*, and the bound on the number of regions in a region graph is determined solely on the basis of the maximal constants appearing in the timed-automaton specification.

The key to the success of their method is the observation that timed-automata transitions *preserve the partition induced by a region graph*. That is, for any two points s_1 and s_2 in a given region R, s_1 has a transition into a region R' if and only if s_2 does. In other words, *regions are not split* on the basis of the transitions exhibited by any two points in a region. Alur and Dill show in this case that trace semantics is preserved by region-graph abstractions, and this provides the basis for their decidability result for the language-emptiness problem.

Taking a closer look, timed automata are a kind of guarded-command automaton. Variables are real-valued, representing clocks or timers, and guards are logical combinations of expressions of the form "x *op* c", where x is a

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

variable, c is an integer constant, and op is a standard comparison operator.[1] Timed-automaton transitions are of two kinds: "discrete transitions", which reset a prescribed subset of the variables to zero; and "delay transitions", which uniformly increment all variables by a constant δ. Given this restricted nature of timed automata and their transitions, it is possible to prove, as Alur and Dill succeeded in doing, that regions indeed are not split by transitions.

In this paper, we show that the region-graph technique can be applied to a much more general class of systems than for which it was originally intended. In particular, we consider Extended Automata (EA), the variables of which can range over dense as well as discrete domains such as the integers. EA guards are as in timed automata, while transitions may update variables independently and arbitrarily via assignments of the form: (i) $x := c$, (ii) $x := y + c$, or (iii) $V := V+ ?$, for any integer c. An assignment of form (iii) is a "random assignment" meaning that there is a constant d in the domain such that each variable in the set V of variables of the EA is incremented by d. This form of assignment can be used to simulate updates to clock variables caused by the progress of time in timed automata. With it, it is easy to see that the class EA contains timed automata.

It turns out that the region-graph technique can easily be extended to handle assignments of form (i), and we show how to do this. The same cannot be said, however, for assignments of form (ii) since, in general, such assignments may split regions. But for the subclass of EA whose form-(ii) assignments do *not* split regions, we can show that the region-graph technique is once again applicable. We refer to this class of systems as SPPA, for Strongly Partition Preserving Automata. In particular, we prove that, for SPPA, bisimulation semantics is preserved under the region-graph abstraction, and hence model checking of modal mu-calculus formulas is decidable.

Although the proof of the decidability result for SPPA is straightforward, this class is still of significant interest. First, it represents a class of very general automata—which, besides timed automata, includes infinite-state, asynchronous untimed systems over integer variables—for which the region-graph technique is applicable. Secondly, we show that the n-process Bakery algorithm is in SPPA, for any fixed n, and hence decidable. To our knowledge, this is the first decidability result for this oft-studied problem to appear in the literature.

The rest of the paper is organized as follows. Extended Automata and Strongly Partition Preserving Automata are defined in Sections 2 and 3, respectively. Section 3 also establishes the decidability of SPPA, while Section 4 shows that the n-process Bakery algorithm, for any fixed n,

[1] We also consider guards of the form "x op $y + c$", where x and y are variables and c is an integer constant.

falls in the class SPPA and its model-checking problem is hence decidable. Section 5 discusses related work, while our concluding remarks are given in Section 6. Due to space limitations, proofs are either sketched or omitted. Full proofs are available in the online version of the paper at http://www.cs.sunysb.edu/~vicdu/sppa.ps.

2 Extended Automata

2.1 Syntax and Semantics of Extended Automata

Extended automata (EA) is a general formalism for the representation of infinite-state systems composed of a finite control and a finite set of variables. Variables may range over discrete or dense domains, and, in contrast to timed automata, may be updated independently and in different ways.

Given a set of variables V and a domain $\mathcal{D}$, a *variable valuation* π maps each variable in V to a value in $\mathcal{D}$. For simplicity of presentation, we restrict our attention to domains containing only non-negative numbers, although our result applies to domains containing negative numbers as well. We use Π to represent the set of all variable valuations, and use π, π_1, π_2, etc. to range over Π. A *guarded command* is a statement of the form $g \rightarrow asgn$ for guard g and assignment $asgn$. A guard is a boolean combination of expressions of the form $x \ \# \ c$ or $x \ \# \ y + c$, where $\# \in \{<, \leq, =, \geq, >\}$, $x, y \in V$, and c is an integer constant (possibly negative). An assignment can be a finite combination of *basic assignments* of the following forms, where c is an integer constant and d is an arbitrary constant in $\mathcal{D}$:

(1) $x := y + c$ for $(y + c) \in \mathcal{D}$ (2) $x := c$

(3) $V := V + ?$ (4) $x := ?$

The meaning of basic assignments of types (1) and (2) is clear. Basic assignments of type (3) mean that all variables in the EA are incremented by the same amount d, for some $d \in \mathcal{D}$. Basic assignments of type (4) mean that x is assigned an arbitrary value in $\mathcal{D}$. Note that in assignments of type (1), x and y may refer to the same variable. Finite combinations of basic assignments, such as those allowed in guarded commands, represent *simultaneous* assignments to a subset of variables in V. We shall write $\pi \models g$ if $\pi \in \Pi$ satisfies guard g, and $\pi_2 = \pi_1[asgn]$ if π_2 is the result of applying $asgn$ to π_1. Notice that given π_1, assignments of types (3) and (4) lead to infinitely many π_2's.

Definition 2.1 An *extended automaton* $\mathcal{A}$ is a 5-tuple $\langle \mathcal{L}, l_0, V, Act, \delta \rangle$, where $\mathcal{L}$ is a set of locations, l_0 is the initial location, V is a set of variables ranging over a possibly infinite domain $\mathcal{D}$, Act is a finite set of actions, and δ is a finite

transition relation. A *transition* in δ is a 4-tuple $\langle l, a, \alpha, l' \rangle$, where $l, l' \in \mathcal{L}$, $a \in Act$, and α is a guarded command.

The semantics of extended automata is given in terms of labeled transition systems (LTSs).

Definition 2.2 An extended automaton $\mathcal{A} = \langle \mathcal{L}, l_0, V, Act, \delta \rangle$ *induces the LTS* $\mathcal{T} = \langle \mathcal{S}_\mathcal{T}, Act, \rightarrow_\mathcal{T}, \langle l_0, \pi_0 \rangle \rangle$ where:

- $\mathcal{S}_\mathcal{T} = \{\langle l, \pi \rangle \mid l \in \mathcal{L}, \pi \in \Pi\}$ is the set of *states*.
- Act is the set of *labels* of $\mathcal{T}$.
- $\langle l_0, \pi_0 \rangle$ is the *initial state*, with π_0 being the valuation where all variables have the value 0.
- $\rightarrow_\mathcal{T}$ is the *transition relation* of $\mathcal{T}$. $\langle l_1, \pi_1 \rangle \xrightarrow{a}_\mathcal{T} \langle l_2, \pi_2 \rangle$ if $\langle l_1, a, (g \rightarrow asgn), l_2 \rangle \in \delta$, $\pi_1 \models g$, and $\pi_2 = \pi_1[asgn]$.

Labels are included in the definition to make our model compatible with systems that have labeled transitions, and with the modal mu-calculus which has labels in formulas. We use the modal mu-calculus introduced by Kozen in [11]. In particular, the only forms of atomic formulas allowed are tt and ff. Given an extended automaton $\mathcal{A}$ and its induced LTS $\mathcal{T}$, a modal mu-calculus formula $\mathcal{F}$ defines a set of states $[\![\mathcal{F}]\!] \in \mathcal{S}_\mathcal{T}$ over which $\mathcal{F}$ is true. The *model-checking problem for extended automata* is, given an EA $\mathcal{A}$ and a modal mu-calculus formula $\mathcal{F}$, determine whether or not $\langle l_0, \pi_0 \rangle \in [\![\mathcal{F}]\!]$.

2.2 Region Graphs for Extended Automata

We now define an equivalence relation $\sim$ on the set of variable valuations of an extended automaton. $\sim$ leads to a finite partition of the state space, and region graphs à la Alur & Dill [2] can be constructed based on this partition.

Given an extended automaton $\mathcal{A}$, let c_{max} be the largest absolute value of the constants appearing in $\mathcal{A}$, and let b_{max} be the largest absolute value of the constants appearing in comparisons involving two variables in $\mathcal{A}$. Both c_{max} and b_{max} are non-negative integers. Further, we shall write $\lfloor d \rfloor$ for the largest integer less than or equal to d, $frac(d)$ for $d - \lfloor d \rfloor$, and $\pi(x)$ for the value of x in valuation π.

Definition 2.3 For all $\pi, \pi' \in \Pi$, $\pi \sim \pi'$ if the following holds:

(i) $\forall x \in V$, either $\lfloor \pi(x) \rfloor = \lfloor \pi'(x) \rfloor$ or both $\pi(x)$ and $\pi'(x)$ are greater than $c_{max} + b_{max}$.

(ii) $\forall x \in V$, if $\pi(x) \leq c_{max} + b_{max}$, then $frac(\pi(x)) = 0$ if and only if $frac(\pi'(x)) = 0$.

(iii) $\forall x, y \in V$, if $\pi(x) \leq c_{max}+b_{max}$ and $\pi(y) \leq c_{max}+b_{max}$, then $frac(\pi(x)) \leq frac(\pi(y))$ if and only if $frac(\pi'(x)) \leq frac(\pi'(y))$.

(iv) $\forall x, y \in V$, either $\lfloor(\pi(x) - \pi(y))\rfloor$ is equal to $\lfloor(\pi'(x) - \pi'(y))\rfloor$, or both $(\pi(x) - \pi(y))$ and $(\pi'(x) - \pi'(y))$ are greater than b_{max}, or both $(\pi(x) - \pi(y))$ and $(\pi'(x) - \pi'(y))$ are smaller than $-b_{max}$.

It is obvious that $\sim$ is an equivalence relation. We shall refer to a block of the partition induced by $\sim$ as a *region*, and use R, R_0, R_1, etc. to range over regions. There are only a finite number of regions for an extended automaton: each region is formed by a finite conjunction of constraints of the form $x = c$, $c < x < c+1$, $x = c_{max} + b_{max}$, $x > c_{max} + b_{max}$, $x - y = e$, $e < x - y < e+1$, $x - y = b_{max}$, and $x - y > b_{max}$, where c is an integer such that $0 \leq c < c_{max} + b_{max}$, and e is an integer such that $0 \leq e < b_{max}$.

Notice that the definition of $\sim$ extends the equivalence relation defined for timed automata in [2] by taking into consideration comparisons involving two variables and assignments of non-zero constants to variables. In particular, Condition (4) ensures that comparisons involving two variables do not distinguish variable valuations belonging to the same region. We distinguish between c_{max} and b_{max} to ensure that an assignment of the form $x := c$ where $c \neq 0$ does not split regions.

Definition 2.4 Given an extended automaton $\mathcal{A}$ and its induced LTS $\mathcal{T}$, the *region graph* $\mathcal{G}$ of $\mathcal{T}$ is the LTS $\langle \mathcal{S}_\mathcal{G}, Act, \rightarrow_\mathcal{G}, \langle l_0, R_0\rangle\rangle$ where:

- $\mathcal{S}_\mathcal{G}$, the set of *states* of $\mathcal{G}$, is such that $\forall$ regions R and locations l, $\langle l, R\rangle \in \mathcal{S}_\mathcal{G}$.
- The set of *labels* Act of $\mathcal{T}$ is also the set of labels for $\mathcal{G}$.
- The *transition relation* $\rightarrow_\mathcal{G}$ of $\mathcal{G}$ is defined as follows: $\langle l_1, R_1\rangle \stackrel{a}{\rightarrow}_\mathcal{G} \langle l_2, R_2\rangle$ if $\exists \pi_1 \in R_1$ and $\pi_2 \in R_2$ such that $\langle l_1, \pi_1\rangle \stackrel{a}{\rightarrow}_\mathcal{T} \langle l_2, \pi_2\rangle$ is a transition of $\mathcal{T}$.
- $\langle l_0, R_0\rangle$ is the *start state* of $\mathcal{G}$, where $\langle l_0, \pi_0\rangle$ is the start state of $\mathcal{T}$ and $\pi_0 \in R_0$.

Since the number of regions, locations, and action labels are all finite, the region graph of an extended automaton is finite.

3 Strongly Partition Preserving Automata

Extended automata can be used to simulate two-counter machines. Therefore, in general the model-checking problem for extended automata is undecidable. We now define a subclass of extended automata for which region graphs can be used to yield decidable model-checking problems.

Definition 3.1 Let $\mathcal{A}$ be an extended automaton and $\mathcal{T}$ its induced LTS with states $\mathcal{S}_\mathcal{T}$. $\mathcal{A}$ is a *strongly partition preserving automaton* (SPPA) if the follow-

ing holds: $\forall\langle l,\pi_1\rangle,\langle l,\pi_2\rangle \in \mathcal{S}_\mathcal{T}$ such that $\pi_1 \sim \pi_2$, $\forall a \in Act$ and $\langle l',{\pi_1}'\rangle \in \mathcal{S}_\mathcal{T}$, if $\langle l,\pi_1\rangle \xrightarrow{a}_\mathcal{T} \langle l',{\pi_1}'\rangle$, then $\exists\langle l',{\pi_2}'\rangle \in \mathcal{S}_\mathcal{T}$ such that $\langle l,\pi_2\rangle \xrightarrow{a}_\mathcal{T} \langle l',{\pi_2}'\rangle$ and ${\pi_1}' \sim {\pi_2}'$.

In the definition, two states of $\mathcal{T}$ having the same location and variable valuations in the same region have successors in the same target regions. That is, the partition induced by the equivalence relation $\sim$ is preserved by the transitions of the SPPA; hence the name *partition preserving*.

Let $\mathcal{A}$ be an SPPA, $\mathcal{T}$ its induced LTS with states $\mathcal{S}_\mathcal{T}$, and $\mathcal{G}$ the region graph of $\mathcal{T}$ with states $\mathcal{S}_\mathcal{G}$. To show the decidability of SPPA, we define the relation $\mathcal{B} \in \mathcal{S}_\mathcal{T} \times \mathcal{S}_\mathcal{G}$ and show that $\mathcal{B}$ is a bisimulation:

$$\mathcal{B} = \{(\langle l,\pi\rangle,\langle l,R\rangle) \mid \langle l,\pi\rangle \in \mathcal{S}_\mathcal{T}, \langle l,R\rangle \in \mathcal{S}_\mathcal{G}, \pi \in R\}$$

Theorem 3.2 *The relation $\mathcal{B}$ is a bisimulation.*

$\mathcal{B}$ is a bisimulation due to the following key observation: the definition of SPPA ensures that $\rightarrow_\mathcal{T}$ can simulate $\rightarrow_\mathcal{G}$, i.e., $\forall(\langle l,\pi\rangle,\langle l,R\rangle) \in \mathcal{B}$ and $a \in Act$, if $\exists\langle l',R'\rangle$ such that $\langle l,R\rangle \xrightarrow{a}_\mathcal{G} \langle l',R'\rangle$, then $\exists\pi' \in R'$, such that $\langle l,\pi\rangle \xrightarrow{a}_\mathcal{T} \langle l',\pi'\rangle$.

Theorem 3.2 implies that the initial state $\langle l_0,\pi_0\rangle$ of $\mathcal{T}$ and the initial state $\langle l_0,R_0\rangle$ of $\mathcal{G}$ are bisimilar. Since $\mathcal{G}$ is a finite LTS, the modal mu-calculus model-checking problem for $\mathcal{G}$ is decidable. On the other hand, bisimilar states satisfy exactly the same set of modal mu-calculus formulas [18]. This yields the following result:

Corollary 3.3 *The model-checking problem for SPPA is decidable.*

4 The Multi-Process Bakery Algorithm is in SPPA

In this section we prove that, for any fixed n, the n-process Bakery algorithm is in SPPA and thus decidable. The Bakery algorithm [12,3] is an n-process mutual-exclusion algorithm. Associated with each process is an integer variable representing the ticket held by that process. When a process wants to enter the critical section, it gets a ticket numbered higher than all the ticket numbers held by other processes. The process with the lowest ticket number is granted access to the critical section. The Bakery algorithm, even for $n = 2$, is an infinite-state system because the values of its variables may increase without bound.

The n-process Bakery algorithm is given in Figure 1. n discrete variables $y_1, y_2, ..., y_n$ are used to control the access to the critical section. The initial value of these variables is 0. In Figure 1, the operator *max* returns the maximum value of its arguments, while the operator *non0_min* returns the smallest non-zero value of its arguments. The conditional test $y_i = non0\_min\ (y_1, ..., y_n)$ can be regarded as a sequence of comparisons of the form $y_j > 0$ or $y_i \leq y_j$.

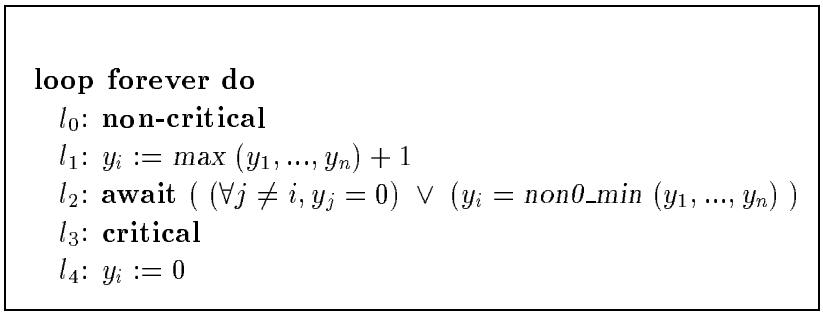


Fig. 1. Process i in the n-process Bakery algorithm. *max* returns the maximum value of its arguments. *non0_min* returns the smallest non-zero value of its arguments.

The assignment $y_i := max\ (y_1, \ldots y_n) + 1$ is executed atomically. The *max* operator can be implemented through the use of assignments of boolean constants to turn variables [3], comparisons involving two variables, and assignments of the form $y_i := y_j + 1$. It can thus be seen as an EA-implementable operation "in disguise".

Theorem 4.1 *The transitions of the n-process Bakery algorithm preserves the partition of its state space induced by $\sim$.*

Proof. For each $n > 0$, the n-process Bakery algorithm—which we shall refer to as Bakery(n)—can be represented by an extended automaton with variables ranging over the domain of non-negative integers. In Bakery(n), $c_{max} = 1$ and $b_{max} = 0$. By the definition of $\sim$, if g is a guard and $\pi \sim \pi'$, then π satisfies g if and only if π' satisfies g. Hence transitions that leave the values of variables unchanged are partition-preserving.

Bakery(n) contains only assignments of form (1) and (2) as defined in Section 2.1. We now show that these assignments, performed respectively at locations l_1 and l_4 in Figure 1, preserve the partition of the state space induced by $\sim$.

Let $\pi_1 \sim \pi_2$, and let $\pi_1{}'$ and $\pi_2{}'$ be the respective variable valuations that arise from subjecting π_1 and π_2 to an assignment. We use a case analysis on the possible forms of the assignment to show that the partition is preserved, i.e. $\pi_1{}' \sim \pi_2{}'$. Since the system is over the integers, we need only consider Conditions (1) and (4) of the definition of $\sim$.

- $l_4 :\ y_i := 0$
  Consider Condition (1) of the definition of $\sim$ for $\pi_1{}'$ and $\pi_2{}'$. $\forall j$, if $j = i$, then $\pi_1{}'(y_j) = \pi_2{}'(y_j) = 0$. If $j \neq i$, then $\pi_1{}'(y_j) = \pi_1(y_j)$, $\pi_2{}'(y_j) = \pi_2(y_j)$. Since $\pi_1(y_j)$ and $\pi_2(y_j)$ satisfy Condition (1) of $\sim$, so do $\pi_1{}'(y_j)$ and $\pi_2{}'(y_j)$. Thus Condition (1) of $\sim$ is true for $\pi_1{}'$ and $\pi_2{}'$.

  Consider Condition (4) of $\sim$. Let $j \neq i$. $\pi_1{}'(y_i) - \pi_1{}'(y_j) = -\pi_1(y_j)$, and $\pi_2{}'(y_i) - \pi_2{}'(y_j) = -\pi_2(y_j)$. Since $\pi_1 \sim \pi_2$, $\pi_1(y_j) = \pi_2(y_j)$, or both $\pi_1(y_j)$

and $\pi_2(y_j)$ are greater than $c_{max} + b_{max} = 1$. Thus $\pi_1'(y_i) - \pi_1'(y_j)$ is either equal to $\pi_2'(y_i) - \pi_2'(y_j)$, or both are less than $-b_{max}$. Thus Condition (4) of $\sim$ holds for $\pi_1'(y_i) - \pi_1'(y_j)$ and $\pi_2'(y_i) - \pi_2'(y_j)$. A similar argument shows that Condition (4) also holds for $\pi_1'(y_j) - \pi_1'(y_i)$ and $\pi_2'(y_j) - \pi_2'(y_i)$.

Now let $j \neq i$ and $k \neq i$. Then $\pi_1'(y_j) - \pi_1'(y_k) = \pi_1(y_j) - \pi_1(y_k)$, $\pi_2'(y_j) - \pi_2'(y_k) = \pi_2(y_j) - \pi_2(y_k)$. Thus Condition (4) also holds in this case. Since both Condition (1) and (4) of $\sim$ still hold after the assignment $y_i := 0$, this assignment preserves the partition.

- $l_1 : \ y_i := max\ (y_1, ..., y_n) + 1$
  Since $\pi_1 \sim \pi_2$, they agree on the ordering of the variables. Let y_k be the variable with the largest value in both π_1 and π_2. Consider Condition (1) of $\sim$ for π_1' and π_2'. $\forall j$, if $j = i$, then $\pi_1'(y_j) = \pi_1(y_k)+1$, $\pi_2'(y_i) = \pi_2(y_k)+1$. Since $\pi_1 \sim \pi_2$, either $\pi_1(y_k) = \pi_2(y_k)$, or both $\pi_1(y_k)$ and $\pi_2(y_k)$ are greater than $c_{max}+b_{max}$. Thus $\pi_1'(y_j) = \pi_2'(y_j)$ or both are greater than $c_{max}+b_{max}$. If $j \neq i$, then since the value of y_j is unchanged, $\pi_1'(y_j) = \pi_2'(y_j)$ or both are greater than $c_{max} + b_{max}$. Thus Condition (1) of $\sim$ holds.

  Consider Condition (4) of $\sim$. There are four cases based on the variables involved:
  - $\pi_1'(y_i) - \pi_1'(y_k) = \pi_2'(y_i) - \pi_2'(y_k) = 1$. Condition (4) is true in this case. It is also true for $\pi_1'(y_k) - \pi_1'(y_i)$ and $\pi_2'(y_k) - \pi_2'(y_i)$.
  - For $j \neq k$ and $j \neq i$, $\pi_1'(y_i) - \pi_1'(y_j) = \pi_1(y_k)+1-\pi_1(y_j)$, $\pi_2'(y_i) - \pi_2'(y_j) = \pi_2(y_k) + 1 - \pi_2(y_j)$. Since $\pi_1 \sim \pi_2$, and y_k is the variable with the largest value in both π_1 and π_2, $\pi_1(y_k) - \pi_1(y_j) = \pi_2(y_k) - \pi_2(y_j)$ or both are greater than b_{max}. Thus $\pi_1'(y_i) - \pi_1'(y_j)$ and $\pi_2'(y_i) - \pi_2'(y_j)$ are either equal or are both greater than b_{max}. Thus Condition (4) holds in this case.
  - For $j \neq k$ and $j \neq i$, $\pi_1'(y_j) - \pi_1'(y_i) = \pi_1(y_j) - \pi_1(y_k) - 1$, $\pi_2'(y_j) - \pi_2'(y_k) = \pi_2(y_j) - \pi_2(y_k) - 1$. Since $\pi_1 \sim \pi_2$, and y_k is the variable with the largest value in both π_1 and π_2, $\pi_1(y_j) - \pi_1(y_k) = \pi_2(y_j) - \pi_2(y_k)$ or both are less than $-b_{max}$. Thus $\pi_1'(y_j) - \pi_1'(y_i)$ and $\pi_2'(y_j) - \pi_2'(y_i)$ are either equal or are both less than $-b_{max}$. Thus Condition (4) holds in this case.
  - For $j \neq i$ and $m \neq i$, $\pi_1'(y_j) - \pi_1'(y_m) = \pi_1(y_j) - \pi_1(y_m)$, $\pi_2'(y_j) - \pi_2'(y_m) = \pi_2(y_j) - \pi_2(y_m)$. Since $\pi_1 \sim \pi_2$, Condition (4) of $\sim$ holds in this case.

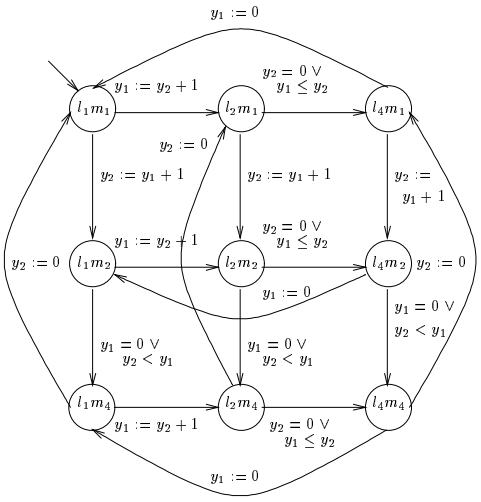
  Thus $\pi_1' \sim \pi_2'$. Therefore, Theorem 4.1 is true. □

To illustrate the proof, consider Bakery(2), the two-process instance of Bakery, which has been widely used as a benchmark for the verification of infinite-state systems [5,7,16,17]. Its pseudo-code, which is given in Figure 2(a), has a straightforward translation into the single-label extended automaton depicted in Figure 2(b). For succinctness, locations l_0, l_3, m_0, m_3 and the label are omitted from this figure. The finite partition of the set of

non-negative integer:
$y_1 = 0, y_2 = 0$

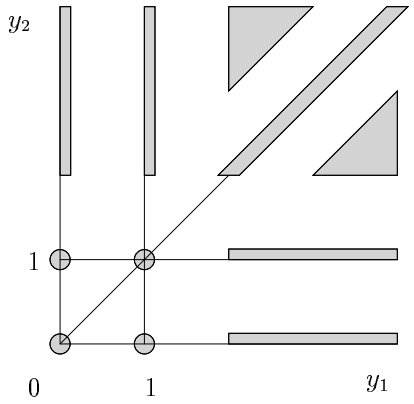
loop forever do	**loop forever do**
l_0: **non-critical**	m_0: **non-critical**
l_1: $y_1 := y_2 + 1$	m_1: $y_2 := y_1 + 1$
l_2: **await** $(y_2 = 0$	m_2: **await** $(y_1 = 0$
$\vee y_1 \leq y_2)$	$\vee y_2 < y_1)$
l_3: **critical**	m_3: **critical**
l_4: $y_1 := 0$	m_4: $y_2 := 0$

(a)



(b)

Fig. 2. The two-process Bakery algorithm.



4 Corner points: e.g. [$(0, 1)$]
5 line segments: e.g. [$1 < x = y$]
2 regions: e.g. [$x > y$, $y > 1$]

Fig. 3. The finite partition for the two-process Bakery algorithm.

variable valuations of Bakery(2) induced by $\sim$ is illustrated in Figure 3 and has a total of 11 regions.

To see that this partition is preserved by the transitions of Bakery(2), consider the two possible forms of assignments: $y_i := 0$ and $y_i := y_j + 1$, $1 \leq i \neq j \leq 2$. For any of the four regions containing single points, these assignments obviously will result in target variable valuations within the same region. It is also easy to see that the other regions are not split. For example, consider the region $\{1 < y_1 < y_2\}$. The effect of the two types of assignments is as follows:

- $y_i := 0$. Then the region in question becomes the region $\{y_i = 0, y_j > 1\}$.
- Let $y_i := y_j + 1$. Then the region in question becomes the region $\{1 < y_j, y_i = y_j + 1\}$. This is a subset of the region $\{1 < y_j < y_i\}$.

In both cases the region is not split and the partition is preserved by the transitions of the system.

Theorem 4.1 leads to the following corollary:

Corollary 4.2 *The n-process Bakery algorithm is in SPPA.*

General techniques for establishing membership in SPPA

In examining the proof of Theorem 4.1, several general techniques for establishing membership in SPPA emerge. First, transitions that do not change values of variables are partition-preserving, and hence need not be considered. Secondly, we have the following lemmas on various forms of assignments:

Lemma 4.3 *Given an EA $\mathcal{A}$, an assignment of the form $x := c$ appearing in $\mathcal{A}$ preserves the partition induced by $\sim$.*

Lemma 4.3 can be proved using arguments similar to those used in the proof of Theorem 4.1 for assignments of the form $y_i := 0$.

Lemma 4.4 *Given an EA $\mathcal{A}$, a region R of $\mathcal{A}$, a positive constant c, and an assignment of the form $x := y + c$ in $\mathcal{A}$, R cannot be split by the assignment if the following invariant holds for R: $\forall \pi \in R$, $\forall v \in V$ such that $v \neq x$ and $v \neq y$, $\pi(y) - \pi(v) \geq -b_{max}$.*

Lemma 4.4 can be proved using arguments similar to those used in the proof of Theorem 4.1 for assignments of the form $y_i := max\ (y_1, ..., y_n) + 1$. In the case of Bakery($n$), the desired invariant is true as the operator max always chooses the y_i with the maximal value as the variable y in assignments of the form $x := y + c$.

Lemma 4.5 *Given an EA $\mathcal{A}$, an assignment of the form $x := y$ appearing in $\mathcal{A}$ preserves the partition induced by $\sim$.*

Lemma 4.5 is true because of the following: $\forall \pi_1, \pi_2$ such that $\pi_1 \sim \pi_2$, let ${\pi_1}' = \pi_1[x := y]$ and ${\pi_2}' = \pi_2[x := y]$. Then the conditions imposed on ${\pi_1}'(x)$ and ${\pi_2}'(x)$ by $\sim$ can be expressed in terms of the conditions imposed on $\pi_1(y)$ and $\pi_2(y)$ by $\sim$. Since $\pi_1 \sim \pi_2$, it immediately follows that ${\pi_1}' \sim {\pi_2}'$.

5 Related Work

The decidability of the class SPPA can be attributed to the existence of an equivalence relation that finitely partitions the infinite state space of an extended automaton. A related approach is taken in [9,1] where the class of Well-Structured Transition Systems (WSTS) is defined. An infinite-state system is in WSTS if there exists a *simulation* relation $\prec$ (a "well ordering")

that induces a finite partition of the state space. WSTS is a very general class of infinite-state systems; timed automata, vector addition systems with states (VASS) [10], normed BPA [4], and rational relational automata [6] are all contained in WSTS. SPPA is also a subclass of WSTS. The main difference between SPPA and WSTS is that SPPA is based on an equivalence relation $\sim$ that defines a *bisimulation* over the set of states. Moreover, $\sim$ is predetermined according to the constants that appear in the system, while there are no general rules about how to find the desired well-ordering $\prec$. SPPA also enjoys decidability for model checking of the full modal mu-calculus, while WSTS is decidable for a limited class of properties, such as reachability and eventuality.

In other related work, various semi-decision procedures and approximation techniques for infinite-state systems have been proposed, and applied to the two-process Bakery algorithm [17,7,8,5]. Our decidability results target the multi-process Bakery algorithm, of which the two-process version is a particular instance. In [17], logic formulas are used to represent (possibly infinite) sets of states, and the satisfiability of the input temporal logic formula is computed via a tableau of the input formula and system; the tableau is successively refined based on the transition relation. This approach requires user intervention and gives no termination guarantee.

[7,8] encode discrete, infinite-state systems as CLP programs, and verify CTL properties by computing least and greatest fixed points of the logical consequence operator. While these computations are intrinsically global, Magic Set transformations [15] are used to make them more goal-directed. The widening operator can also be used as an approximation technique when the fixed-point computation does not terminate.

In [5], Presburger formulas are used to encode sets of states as well as the transition relations of systems under investigation. The Omega Library, which is a fast Presburger solver, allows efficient manipulation of these formulas, including the computation of the set of predecessors of a set of states. CTL model checking can be performed by computing fixed points of the predecessor relation over these formulas.

In [14], an approach to verifying infinite-state systems using program transformations is proposed. Given a program P, they essentially attempt to compute a finite partition of P's state space in a top-down manner, starting with the initial set of constraints in P (i.e. conditional tests) and iteratively generating new constraints. This approach is quite general and can be applied to any system having a finite simulation- or bisimulation-equivalent abstract program. For example, they show that in the case of the two-process bakery algorithm, only finitely many new constraints are generated and their technique thereby correctly computes the (finite) bisimulation-induced partition.

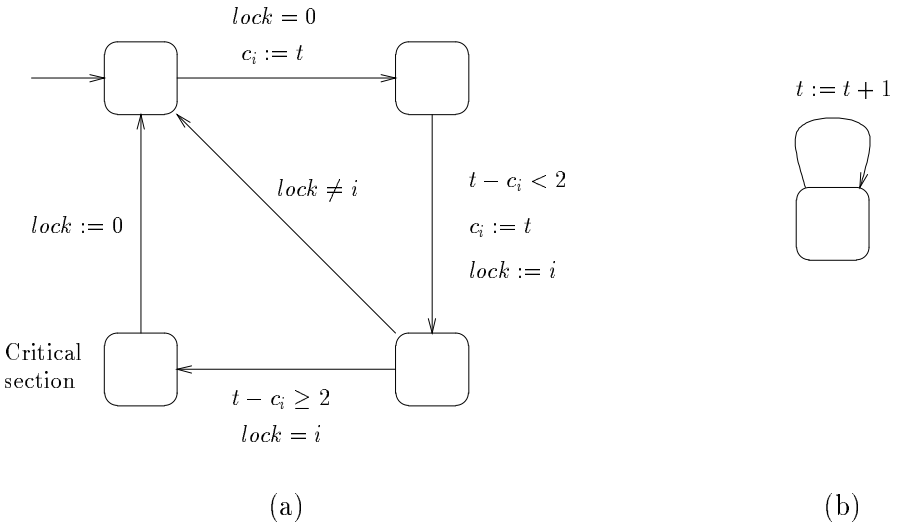


Fig. 4. The n-process Fischer's protocol as described in [13]. t is a global timer. $lock$ is a shared variable, and c_i's are local variables.

For some systems, however, infinitely many new constraints may be generated unless an appropriate bound is placed on the number of iterations. An example of such a system is the version of the n-process Fischer's mutual exclusion protocol described in [13] and reproduced here as Figure 4. On the other hand, this protocol is in SPPA, for any fixed n. It has three forms of assignments: (1) $lock := i$, where i is a constant; (2) $c_i := t$; and (3) $t := t+1$. Techniques for determining membership in SPPA as discussed in Section 4 can be directly used to show that assignments of the forms (1) and (2) preserve the partition induced by $\sim$. Assignments of form (3) also preserve the partition, as the invariant $t - c_i \geq 0$ holds for the system.

6 Conclusions

We have shown how region graphs, originally intended for timed automata, can be fruitfully deployed in the verification of infinite-state untimed systems. Our results were framed in terms of SPPA, a decidable class of extended automata, which we showed contains the n-process Bakery algorithm, for any fixed n.

We have also shown that SPPA contains Čerāns's rational relational automata [6,1], modulo constant conversion; and we have identified a generalization of SPPA called WPPA (Weak Partition Preserving Automata) in which *sequences* of transitions (rather than individual transitions, as in SPPA) are partition-preserving. These additional results will appear in the full version of the paper.

So how is membership in SPPA decided? Let $\mathcal{A}$ be an extended automaton. Since $\mathcal{A}$ has a finite number of transitions and the partition induced by $\sim$ has a finite number of regions, all regions can be inspected against all transitions to determine whether the partition is preserved. [2] Such a brute-force approach clearly requires time exponential in the number of variables in $\mathcal{A}$. It would thus be desirable to obtain a syntactic characterization of an interesting subclass of SPPA that could be checked efficiently. Research in this direction is ongoing.

In other future work, the Ticket algorithm [3], like the Bakery algorithm, is a discrete, infinite-state system widely used as a benchmark for the verification of infinite-state systems. Its extended automaton, however, is not in SPPA nor WPPA. This is because the Ticket algorithm contains assignments of the form $s := s+1$, for which the desired invariants relating the variable s to other variables in the algorithm cannot be proved. Nevertheless, a finite partition of its state space can be constructed by hand, and this partition is preserved by the transitions of its extended automaton. It would therefore be interesting to see how the notion of partition-preservation can be extended to accommodate such problems.

Acknowledgments

The authors are grateful to the anonymous MTCS 2000 referees, especially the one who provided us with an extremely helpful and detailed report on our submission. This research was supported in part by NSF grants CCR-9505562 and CCR-9705998.

References

[1] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of LICS'96*, pages 313–321. IEEE Computer Society Press, 1996.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] G. R. Andrews. *Concurrent Programming: Principles and Practice.* Addison-Wesley, Menlo Park, CA, 1991.

[4] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proceedings of POPL'95*, pages 95–106. ACM Press, 1995.

[2] Typically, however, such an exhaustive check is not needed; rather, a case analysis on the types of assignments that appear in $\mathcal{A}$ can be used instead. Indeed, this is the approach we followed in the case of the n-process Bakery algorithm.

[5] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite-state systems using Presburger Arithmetics. In *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV '97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 400–411, Haifa, Israel, July 1997. Springer-Verlag.

[6] K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *Proceedings of ICALP'94*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer-Verlag, 1994.

[7] G. Delzanno and A. Podelski. Model checking in CLP. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99), Lecture Notes in Computer Science*, volume 1579, pages 223–239, Amsterdam, March 1999.

[8] G. Delzanno and A. Podelski. Constraint-based deductive model checking in CLP. *International Journal on Software Tools for Technology Transfer*, 2000. To appear.

[9] A. Finkel and P. Schnoebelen. Fundamental structures in well-structured infinite transition systems. In *Proceedings of the 3rd Latin American Theoretical Informatics Symposium (LATIN '98)*, volume 1380 of *Lecture Notes in Computer Science*, pages 102–118, Campinas, Brazil, April 1998.

[10] J. Hopcroft and J. J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.

[11] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[12] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, August 1974.

[13] D. Lesens and H. Sadi. Automatic verification of parameterized networks of processes by abstraction. In *Proceedings of the 2nd International Workshop on Verification of Infinite State Systems (INFINITY'97)*, Bologna, July 1997.

[14] K. S. Namjoshi and R. P. Kurshan. Syntactic program transformations for automatic abstraction. In *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV '00)*. Springer-Verlag, 2000. To appear.

[15] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Efficient bottom-up evaluation of logic programs. In P. De Wilde and J. Vandewalle, editors, *Computer Systems and Software Engineering: State-of-the-Art*. Kluwer Academic, 1992.

[16] V. Rusu and E. Singerman. On proving safety properties by integrating static analysis, theorem proving and abstraction. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 178–192. Springer-Verlag, March 1999.

[17] H. B. Sipma, T. E. Uribe, and Z. Manna. Deductive model checking. *Formal Methods in System Design*, 15(1):49–74, July 1999.

[18] C. Stirling. The joys of bisimulation. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS '98*, volume 1450 of *Lecture Notes in Computer Science*, pages 142–151. Springer-Verlag, 1998.