Algorithmic Analysis of Programs with Well Quasi-Ordered Domains *

Parosh Aziz Abdulla
Dept. Computer Systems
Uppsala University
P.O.Box 325
751 05 Uppsala, Sweden
parosh@docs.uu.se

Kārlis Čerāns
Institute of Mathematics
and Computer Science
University of Latvia
karlis@cclu.lv

Bengt Jonsson
Dept. Computer Systems
Uppsala University
P.O.Box 325,
751 05 Uppsala, Sweden
bengt@docs.uu.se

Yih-Kuen Tsay
Department of Information Management,
National Taiwan University, Taipei, Taiwan,
tsay@im.ntu.edu.tw

Abstract

Over the last few years there has been an increasing research effort directed towards the automatic verification of infinite state systems. This paper is concerned with identifying general mathematical

*Supported in part by the Swedish Board for Industrial and Technical Development (NUTEK) and by the Swedish Research Council for Engineering Sciences (TFR). The work of the second author has been partially supported by Grant No. 93-596 from Latvian Council of Science. The work of the fourth author has been partially supported by the National Science Council, TAIWAN (R.O.C).

structures which can serve as sufficient conditions for achieving decidability. We present decidability results for a class of systems (called *well-structured systems*), which consist of a finite control part operating on an infinite data domain. The results assume that the data domain is equipped with a preorder which is a well quasi-ordering, such that the transition relation is "monotonic" (is a simulation) with respect to the preorder. We show that the following properties are decidable for well-structured systems:

- *Reachability:* whether a certain set of control states is reachable. Other safety properties can be reduced to the reachability problem.
- *Eventuality:* whether all executions eventually reach a given set of control states (represented as AFp in CTL).
- *Simulation:* whether there exists a simulation between a finite automaton and a well-structured system. The simulation problem will be shown to be decidable in both directions.

We also describe how these general principles subsume several decidability results from the literature about timed automata, relational automata, Petri nets, and lossy channel systems.

1 Introduction

Over the last few years there has been an increasing research effort directed towards the automatic verification of infinite state systems. This has resulted in numerous highly nontrivial algorithms for the verification of different classes of such systems. Examples include timed automata [ACD90, AH89, Č92a], hybrid automata [Hen95], relational automata ([BBK77, Č92b, Č94]), Petri nets ([Jan90, JM95]), systems with many identical processes [CG87, PP92], and lossy channel systems [AJ93, AK95]. As the interest in this area increases, it will be important to extract common principles that underlie these and related results.

Our goal is to develop general mathematical structures which could serve as sufficient conditions for achieving decidability. Our objective is twofold. We aim on the one hand to give a unified explanation of existing decidability results including those mentioned above, and on the other hand to provide guidelines for discovering similar decidability results for other classes of systems.

Existing work on general principles for deciding properties of infinite-state systems is fairly limited. Many existing methods are based on *finite partitioning*, where the state space of the original system is partitioned into a finite number of equivalence classes under bisimulation [ACD90, Hen95, Č94]. Two states belonging to the same partition are equivalent in the sense that transitions from them lead to equivalent states. The requirement of having an appropriate finite partitioning of the state space is rather restrictive since it implies that the system under consideration is "essentially finite state".

In this paper we present substantially more general conditions for decidability of several verification problems. We work with a *preorder* on states instead of an *equivalence*. We consider systems which consist of a finite control part operating on an infinite data domain. The main requirement is that the data domain is equipped with a preorder such that the following properties (which are generalizations of those required by finite partitioning methods) hold: (i) the transition system is "monotonic" with respect to the preorder, i.e. transitions from larger states lead to larger states (this means that smaller states are *simulated* by larger states); and (ii) the preorder on the data domain is a well quasi-ordering, which means that each infinite sequence contains an element which is larger than or equivalent to an earlier element in the sequence. We call the class of systems satisfying these properties *well-structured systems.*

Our method generalizes finite partitioning in the following sense:

- We employ a preorder instead of an equivalence relation. It is clear that having an equivalence relation is a special case of having a preorder (an equivalence relation is a preorder, where the relation is also symmetric).
- We work with states which are related through simulation, instead of bisimulation in the case of finite partitioning. Observe that by taking the preorder to be an equivalence, the definitions of simulation and bisimulation coincide.
- we require the preorder to be a well quasi-ordering, instead of requiring the number of equivalence classes to be finite. In case the preorder is taken to be an equivalence relation, our requirement implies the number of equivalence classes to be finite.

This means that, apart from systems whose state spaces can be finitely partitioned, e.g. timed automata [ACD90, Č92a], various classes of hybrid automata [Hen95], and rational relational automata [Č94], our methods can be used to analyze systems which do not allow for finite partitioning, such as Petri nets [JM95], lossy channel systems [AJ93], and integral relational automata [Č94].

In this paper, we show that the following properties are decidable for well-structured systems:

- *Reachability:* whether a certain set of control states is reachable. Several properties can be reduced to the reachability problem, notably invariant properties and safety properties represented by the prefix-closed set of traces of a finite automaton.
- *Eventuality:* whether all executions eventually reach a given set of control states (represented as AFp in CTL).
- *Simulation:* whether there is a simulation between a finite automaton and a well-structured system. The simulation problem is shown to be decidable in both directions.

The reachability problem is solved by a backward reachability analysis. Starting from a set I of states, the reachability of which is to be decided, we generate the set of states from which I can be reached by a sequence of transitions of length less than or equal to j for successively larger j. The sets that are successively generated in this way are upwards closed with respect to the preorder and form an ascending chain (under set-inclusion). Since the preorder is well quasi-ordered, each set can be represented by a finite set of minimal states, and the chain converges after a finite number of iterations. The problem of whether a well-structured system is simulated by a finite automaton is solved using similar principles. Eventuality properties and the problem of whether a finite automaton is simulated by a well-structured system system are checked by a standard tableau method. Again the tableau construction terminates by the well quasi-ordering property.

The iteration method in this paper can also be viewed as an abstract interpretation of the infinite state space. Instead of working with sets of states of the transition system, we work in an abstract domain consisting of finite sets of minimal states. One contribution is that we show, for well-structured systems, that we can work in this abstract domain, without losing precision in our analysis of reachability, and with the additional benefit that fixpoint iterations of this kind always converge.

Related Work The idea of verifying a system by analyzing a property for an abstraction or a simpler approximation of the system has been considered by several authors [CGL92, LGS⁺95, DGG94]. These papers present conditions such that if the property is satisfied by the abstract program then it will be satisfied by the original program. Sufficient conditions are given for an abstraction to preserve e.g. the branching time logic CTL^* or fragments thereof. However, these works are not concerned primarily with constructing decision procedures for verification as we do.

Finkel [Fin90] shows that, for well-structured systems, it is decidable whether a system has a finite reachability tree. In this paper we use a variant of his algorithm for checking eventuality properties. He also considers a restricted class of well-structured systems, namely those with *strict monotonicity*. This means that transitions from *strictly* larger states lead to *strictly* larger states. For this class it is shown that the coverability problem and the problem whether the set of reachable states is finite, are both decidable. The coverability problem is equivalent to the control state reachability problem, and is solved in [Fin90] using a generalization of the Karp-Miller algorithm [KM69]. This algorithm depends on strict monotonicity, which does not hold in general for well-structured systems (e.g. for lossy channel systems), and hence the Karp-Miller algorithm cannot be applied to our class of systems.

Outline The remainder of the paper is structured as follows. In the following section, we define infinite-state systems as systems with a finite-state control part which operates on a possibly infinite domain of data values. In Section 3 we define well-structured systems. Section 4 presents the method for deciding reachability, Section 5 treats eventuality properties, and Section 6 shows how to check the existence of simulations. In Section 7 we give examples of several classes of well-structured systems. In Section 8 we give some conclusions and directions for future research.

2 Infinite-State Systems

In this section we give the basic definitions for infinite-state systems. As a general model of such systems, we adopt labeled transition systems. We assume a finite set Λ of *labels*. Each label $\lambda \in \Lambda$ represents an observable interaction with the environment.

Definition 2.1 A (labeled) transition system $\mathcal{L}$ is a pair $\langle S, \delta \rangle$, where

- S is a set of states, formed by the cartesian product of a finite set Q of *control states* and a possibly infinite set D of *data values*, and
- $\delta \subseteq S \times \Lambda \times S$ is the set of allowed transitions.

We use $\langle q, d \rangle$ to denote the state whose control part is q and whose data part is d, and $s \xrightarrow{\lambda} s'$ to denote that $\langle s, \lambda, s' \rangle \in \delta$. Intuitively, $s \xrightarrow{\lambda} s'$ means that the system can move from state s to state s' while performing the observable action λ. We let $s \longrightarrow s'$ denote that there is a λ such that $s \xrightarrow{\lambda} s'$, and let $\xrightarrow{*}$ denote the reflexive transitive closure of $\longrightarrow$.

For $q \in Q$ and $A \subseteq D$, we use $\langle q, A \rangle$ to denote the set $\{\langle q, d \rangle \mid d \in A\}$. For $s \in S$ and $T \subseteq S$ we say that T *is reachable from* s (written $s \xrightarrow{*} T$) if there exists a state $s' \in T$ such that $s \xrightarrow{*} s'$.

For $T \subseteq S$ and $\lambda \in \Lambda$, we define $pre_\lambda(T)$ to be the set $\left\{ s' \mid \exists s \in T.\ s' \xrightarrow{\lambda} s \right\}$. Analogously, we define $post_\lambda(T)$ as $\left\{ s' \mid \exists s \in T.\ s \xrightarrow{\lambda} s' \right\}$. By $pre(T)$ ($post(T)$) we mean $\cup_{\lambda \in \Lambda} pre_\lambda(T)$ ($\cup_{\lambda \in \Lambda} post_\lambda(T)$). Sometimes we write $pre(s)$ ($post(s)$) instead of $pre(\{s\})$ ($post(\{s\})$).

A *computation from a state* s is a sequence of the form $s_0 s_1 \cdots s_n$, where $s_0 = s$, $s_i \longrightarrow s_{i+1}$, and either $n = \infty$ (i.e. the sequence is infinite) or there is no state s' such that $s_n \longrightarrow s'$.

3 Well-structured Systems

In this section, we define the class of transition systems which we call *well-structured systems*, for which we will present our decidability results. First, we recall the notion of *preorders*.

3.1 Preliminaries

A *preorder* $\preceq$ is a reflexive and transitive (binary) relation on a set D. We say that $\preceq$ is *decidable* if there is a procedure which, given $a, b \in D$, decides whether $a \preceq b$. We say that $\preceq$ is a *well quasi-ordering*, if there is no infinite sequence $a_0, a_1, a_2, \ldots$, such that $a_i \not\preceq a_j$ for all $i < j$. A set M is said to be *canonical* if $a, b \in M$ implies $a \not\preceq b$. We say that $M \subseteq A$ is a *minor set* of A, if (i) for all $a \in A$ there exists $b \in M$ such that $b \preceq a$, and (ii) M is canonical.

A set $I \subseteq D$ is an *ideal (in D)* if $a \in I$, $b \in D$, and $a \preceq b$ imply $b \in I$. We define the *(upward) closure* of a set $A \subseteq D$, denoted $\mathcal{C}(A)$, as the ideal $\{b \in D \mid \exists a \in A.\ a \preceq b\}$ which is generated by A.

For sets A and B, we say that $A \equiv B$ if $\mathcal{C}(A) = \mathcal{C}(B)$. Observe that $A \equiv B$ if and only if for all $a \in A$ there is a $b \in B$ such that $b \preceq a$, and vice versa.

Lemma 3.1 If a preorder $\preceq$ is a well quasi-ordering, then for each set A there exists at least one finite minor set of A.

Proof. Suppose that no finite minor set of A exists. We show that $\preceq$ is not a well quasi-ordering. We define the infinite sequence $a_0, a_1, a_2, \ldots$ of elements in A as follows. Let a_0 be any arbitrary element in A. We choose a_{i+1} such that $a_j \not\preceq a_{i+1}$ for each $1 \leq j \leq i$. The element a_{i+1} exists, since otherwise we could easily construct a minor set of the finite set $\{a_0, a_1, \ldots, a_i\}$, which would also be a minor set of A, contradicting the assumption that no such sets exist. It is clear that the sequence $a_0, a_1, a_2, \ldots$ violates the well quasi-orderedness property. $\square$

Notice that although Lemma 3.1 implies that each minor set is finite, there may still be infinitely many such minor sets. Also, we observe that if $\preceq$ is a partial order then there exists a unique minor set of A. We use *min* to denote a function which, given a set A, returns a minor set of A.

From Lemma 3.1, and the fact that $\mathcal{C}(min(I)) = I$ for each ideal I, it follows that we can use $min(I)$ as a finite representation of I.

Lemma 3.2 For a preorder $\preceq$ on a set A, $\preceq$ is a well quasi-ordering iff for each infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ of ideals in A there is a k such that $I_k = I_{k+1}$

Proof. *(only if)* - Suppose that we have an infinite sequence $I_0 \subset I_1 \subset I_2 \subset \cdots$ It follows that there is a sequence $a_0, a_1, a_2, \ldots$ of elements in A such that for all $k \geq 0$ we have $a_k \in I_k$ and $a_k \notin I_j$ for each $j < k$. This means $a_j \not\preceq a_k$ for $j < k$, otherwise $a_k \in I_j$, since I_j is an ideal. This is a contradiction since the sequence $a_0, a_1, a_2, \ldots$ will then violate the well quasi-ordering assumption.

(if) - Suppose that we have an infinite sequence of elements $a_0, a_1, a_2, \ldots$ in A where $a_j \not\preceq a_k$ if $j < k$. We define an infinite sequence $I_0, I_1, I_2, \ldots$ of ideals where $I_j = \mathcal{C}(\{a_0, a_1, \ldots, a_k\})$. It is clear that $I_0 \subset I_1 \subset I_2 \subset \cdots$. $\square$

In fact, for any sequence $I_0, I_1, I_2, \ldots$ we can show that there are j and k such that $j < k$ and $I_j = I_k$. We use the *only if*-direction of Lemma 3.2 to prove termination of some of our verification algorithms.

3.2 Well-structured Systems

In our framework we require that the set D of data values is equipped with a decidable preorder $\preceq$, and assume that we are given a minor set of D which we henceforth call D_{min}. We extend the preorder $\preceq$ on D to a decidable preorder $\preceq$ on the set S of states defined by $\langle q, d\rangle \preceq \langle q', d'\rangle$ if and only if $q = q'$ and $d \preceq d'$.

A transition system $\langle S, \delta\rangle$ is *monotonic* (with respect to $\preceq$) if for each $s_1, s_2, s_3 \in S$ and $\lambda \in \Lambda$, if $s_1 \preceq s_2$ and $s_1 \stackrel{\lambda}{\longrightarrow} s_3$, then there exists s_4 such that $s_3 \preceq s_4$ and $s_2 \stackrel{\lambda}{\longrightarrow} s_4$.

Lemma 3.3 A transition system $\langle S, \delta\rangle$ is monotonic iff the set of ideals in S is closed under the applications of both pre_λ and pre.

Proof. First we show the claim for pre_λ *(only if)* - Suppose that $\langle S, \delta\rangle$ is monotonic. Take any ideal I in S. Suppose that $s_1 \in pre_\lambda(I)$ and $s_1 \preceq s_2$. We show that $s_2 \in pre_\lambda(I)$. We know that there is $s_3 \in I$ such that $s_1 \stackrel{\lambda}{\longrightarrow} s_3$. By monotonicity it follows that there is s_4 such that $s_3 \preceq s_4$ and $s_2 \stackrel{\lambda}{\longrightarrow} s_4$. Since I is an ideal, we have $s_4 \in I$, and hence $s_2 \in pre_\lambda(I)$.

(if) - Suppose that $\langle S, \delta\rangle$ is not monotonic. It follows that there are states s_1, s_2, and s_3, and $\lambda \in \Lambda$ such that $s_1 \preceq s_2$, $s_1 \stackrel{\lambda}{\longrightarrow} s_3$, but there is no s_4 where $s_3 \preceq s_4$ and $s_2 \stackrel{\lambda}{\longrightarrow} s_4$. Define the ideal $I = \mathcal{C}(\{s_3\})$. It is clear that $s_1 \in pre_\lambda(I)$ but $s_2 \notin pre_\lambda(I)$. This means that $pre_\lambda(I)$ is not an ideal. The claim for pre follows from the fact that $pre = \cup_{\lambda \in \Lambda} pre_\lambda$. $\square$

Definition 3.4 A transition system $\mathcal{L} = \langle S, \delta \rangle$, assuming a decidable preorder $\preceq$ on the set D of data values, is said to be *well-structured* if

1. it is monotonic;
2. $\preceq$ is a well quasi-ordering; and
3. for each state $s \in S$ and $\lambda \in \Lambda$, the set $min(pre_\lambda(\mathcal{C}(\{s\})))$ is computable.

Note that $min(pre_\lambda(\mathcal{C}(\{s\})))$ is finite if $\preceq$ is a well quasi-ordering. We define $minpre_\lambda(s)$ as notation for $min(pre_\lambda(\mathcal{C}(\{s\})))$. For a set T of states we use $minpre_\lambda(T)$ to denote $\cup_{s \in T} minpre_\lambda(s)$. On the concrete models where we shall apply our theory (Section 7) the computability of $minpre_\lambda(s)$ will be rather obvious given the explicit syntactic representations of the transition relations.

Comment on the Representation of Ideals In this paper, we will represent ideals by minor sets. An alternative (and more general) representation is in terms of *constraints*. We then assume a set Φ of constraints over the domain D of data values. Each constraint ϕ denotes a subset $[\![\phi]\!]$ of D. Given a set Φ of constraints, we can define a preorder $\preceq_\Phi$ on D by $d \preceq_\Phi d'$ iff for all constraints ϕ in Φ we have that $d \in [\![\phi]\!]$ implies $d' \in [\![\phi]\!]$. We observe that the constraint representation is at least as general as the approach we use here where we start by a preorder: an arbitrary preorder $\preceq$ can be obtained as the preorder $\preceq_\Phi$ by letting Φ be the set which for each $d_0 \in D$ contains a constraint that denotes the set $\{d \in D \mid d_0 \preceq d\}$.

Instead of using finite minor sets to represent ideals, we can use finite sets of constraints. A set of constraints denotes the union of the denotations of its elements (recall that each constraint denotes a set). In some cases (e.g., for real-time automata) such a representation is more convenient, since a constraint sometimes represents a large minor set.

3.3 Composition

For labeled transition systems $\mathcal{L}_1 = \langle S_1, \delta_1 \rangle$ and $\mathcal{L}_2 = \langle S_2, \delta_2 \rangle$, we define the *composition* $\mathcal{L}_1 \| \mathcal{L}_2$ of $\mathcal{L}_1$ and $\mathcal{L}_2$ to be the transition system $\langle S, \delta \rangle$ where

- $S = S_1 \times S_2$.
- $\langle s_1, s_2 \rangle \xrightarrow{\lambda} \langle s_1', s_2' \rangle$ iff $s_1 \xrightarrow{\lambda} s_1'$ and $s_2 \xrightarrow{\lambda} s_2'$.

Our definition of composition is the standard one taken from process algebras such as CSP, or LOTOS. It can also be used to describe the behaviour of a transition system operating on different data domains (see e.g. Section 7.5).

Theorem 3.5 For well-structured systems $\mathcal{L}_1$ and $\mathcal{L}_2$, the composition $\mathcal{L}_1 \| \mathcal{L}_2$ is well-structured. Moreover, if both $\mathcal{L}_1$ and $\mathcal{L}_2$ are essentially finite branching, resp. intersection effective, then also $\mathcal{L}_1 \| \mathcal{L}_2$ is.

Proof. Let $\mathcal{L}_1 = \langle S_1, \delta_1 \rangle$ and $\mathcal{L}_2 = \langle S_2, \delta_2 \rangle$. Let $\preceq_1$ and $\preceq_2$ be the respective preorders defined on S_1 and S_2. Define the preorder $\preceq$ for L by $\langle s_1, s_2 \rangle \preceq \langle s_1', s_2' \rangle$ whenever both $s_1 \preceq_1 s_1'$ and $s_2 \preceq_1 s_2'$. Monotonicity, computability of *minpre*, essentially finite branching, and intersection effectiveness follow directly from their definitions.

To prove the well quasi-ordering property of $\preceq$, consider an infinite sequence of state pairs $\langle s_0, s_0' \rangle, \langle s_1, s_1' \rangle, \langle s_2, s_2' \rangle, \ldots$ where $s_i \in S_1$ and $s_i' \in S_2$. It follows from the well quasi-ordering of $\preceq_1$ that there is an infinite increasing sequence $i_0, i_1, i_2, \ldots$ such that $s_{i_j} \preceq_1 s_{i_k}$ whenever $i_j \leq i_k$. Since $\preceq_2$ is also a well quasi-ordering we conclude that there are j and k where $j < k$ and $s_{i_j}' \preceq_2 s_{i_j}'$. This means that $\langle s_{i_j}, s_{i_j}' \rangle \preceq \langle s_{i_k}, s_{i_k}' \rangle$. $\square$

4 Control State Reachability

In this section we describe an algorithm to solve the control state reachability problem for well-structured transition systems. More precisely, given a state s and a control state q, we want to check whether $\langle q, D \rangle$ is reachable from s. Our algorithm actually solves the more general problem of deciding whether an ideal I is reachable from a given state s. Since $\langle q, D \rangle$ is an ideal, the control state reachability problem is a special case of the reachability problem for ideals.

To check the reachability of an ideal I, we perform a reachability analysis backwards. Starting from I we define the sequence $I_0, I_1, I_2, \ldots$ of sets by $I_0 = I$ and $I_{j+1} = I \cup pre(I_j)$. Intuitively, I_j denotes the set of states from which I is reachable in j or less steps. Thus, if we define $pre^*(I)$ to be $\cup_{j\geq 0} I_j$, then I is reachable from s if and only if $s \in pre^*(I)$. Notice that $pre^*(I)$ is the least fixpoint $\mu X.\ I \cup pre(X)$. By Lemma 3.3 each I_j is an ideal in S. We know that $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$, and hence from Lemma 3.2 it follows that there is a k such that $I_k = I_{k+1}$. It can easily be seen that $I_\ell = I_k$ for all $\ell \geq k$ implying that $pre^*(I) = I_k$.

Our method for deciding whether I is reachable is based on generating the above sequence $I_0, I_1, I_2, \ldots$ of ideals, and checking for convergence. This cannot be carried out directly since I_j is an infinite set. Instead, we represent each I_j by a canonical set $M_j = min(I_j)$. By Lemma 3.1 each minor set M_j is finite. It is straightforward to show that $M_{j+1} \equiv min(min(I) \cup minpre(M_j))$, which is computable as

$$M_{j+1} = min\left(min(I) \cup \bigcup_{s \in M_j} min(pre(\mathcal{C}(\{s\})))\right)$$

since, by the definition of well-structured transition systems, each set $min(pre(\mathcal{C}(\{s\})))$ is computable, and the union is taken over a finite set of sets.

From the above discussion we conclude that if we define $minpre^*(M_0)$ to be $\cup_{j\geq 0} M_j$, then there is a k such that $M_{k+1} \equiv M_k$, and $minpre^*(M_0) \equiv M_k$. This implies that $minpre^*(M)$ is computable for any minor set M of I and in fact $\mathcal{C}(minpre^*(M)) = pre^*(I)$.

Theorem 4.1 The control state reachability problem is decidable for well-structured systems.

Proof. Given a state s and a control state q we compute $minpre^*(\langle q, D_{min}\rangle)$. We then check whether there is an $s' \in minpre^*(\langle q, D_{min}\rangle)$ such that $s' \preceq s$. $\square$

Abstract Interpretation The above analysis algorithm can also be phrased in terms of abstract interpretation [CC77, JN94]. We intend to compute the fixpoint $\mu X.\ I \cup pre(X)$ for a set $I \subseteq S$ by iteration. Instead of computing this fixpoint in the lattice $\langle 2^S, \subseteq\rangle$ of sets of states, we move to the abstract lattice $\langle \mathcal{M}, \sqsubseteq\rangle$, where $\mathcal{M}$ is the set of canonical subsets of S, and where $M \sqsubseteq M'$ if $\mathcal{C}(M) \subseteq \mathcal{C}(M')$. The correspondence between the concrete lattice $\langle 2^S, \subseteq\rangle$ and the abstract lattice $\langle \mathcal{M}, \sqsubseteq\rangle$ is expressed by a pair $\langle \alpha, \gamma\rangle$ of functions as follows.

- $\alpha : 2^S \mapsto \mathcal{M}$, defined by $\alpha(T) = min(T)$ maps each set of states in the concrete lattice to its abstract representation.
- $\gamma : \mathcal{M} \mapsto 2^S$, defined by $\gamma(M) = \mathcal{C}(M)$ recovers the concrete meaning of an element in the abstract lattice.

The pair $\langle \alpha, \gamma \rangle$ forms a *Galois insertion* of $\langle \mathcal{M}, \sqsubseteq \rangle$ into $\langle 2^S, \subseteq \rangle$.

Our algorithm for deciding reachability can be seen as computing the fixpoint $\mu X.\ min(I) \sqcup minpre(X)$ in the lattice $\langle \mathcal{M}, \sqsubseteq \rangle$, where $M_1 \sqcup M_2 = min(M_1 \cup M_2)$. The monotonicity of the transition relation ensures that this computation corresponds exactly to the computation $\mu X.\ I \cup pre(X)$ in $\langle 2^S, \subseteq \rangle$ if I is an ideal in S. Exactness follows from the identity

$$pre(\gamma(M)) = \gamma(minpre(M))$$

for all $M \in \mathcal{M}$, and ensures that if the fixpoint computation converges to M_k, then $\gamma(M_k)$ is the least fixpoint of $\mu X.\ I \cup pre(X)$ in $\langle 2^S, \subseteq \rangle$. Finally, well quasi-orderedness of $\preceq$ implies that all ascending chains in $\langle \mathcal{M}, \sqsubseteq \rangle$ are finite, thus guaranteeing convergence of any least fixpoint computation.

5 Eventuality Properties

In this section we describe an algorithm for deciding whether each computation starting from an initial state eventually reaches a certain control state satisfying a predicate p over control states. In CTL, these properties are of the form AFp. We present an algorithm for the dual property EGp from which an algorithm for AFp can easily be derived using the correspondence AF$p \equiv \neg$EG$\neg p$. The property EGp is true in a state s_0 iff there is a computation from s_0 in which all states have a control part that satisfies p. Our algorithm will actually solve the more general problem of whether s_0 satisfies a property of the form EGI for an ideal I. We write this property as $s_0 \models$ EGI.

The algorithm essentially builds a tree of reachable states, starting from the initial state and successively exploring the successors of each state in the tree. We must then consider the possibility that $post(s)$ is infinite for some states s (i.e., the transition relation is not finite branching). To overcome this difficulty, we say that a transition system is *essentially finite branching* if for each state s we can effectively compute a finite subset of $post(s)$, denoted $maxpost(s)$, such that for each state $s' \in post(s)$ there is a state $s'' \in maxpost(s)$ with $s' \preceq s''$. If $post(s)$ is finite, then $maxpost(s)$ can be taken as $post(s)$. However, in the cases where $post(s)$ is infinite (as can be the case, e.g., for real-time automata), the subset $maxpost(s)$ can fully represent the set $post(s)$ for the purposes of this algorithm.

In the algorithm, we build a tree labeled by properties of the form $s \models \mathrm{EG}I$. The root node is labeled by $s_0 \models \mathrm{EG}I$. A node labeled by $s \models \mathrm{EG}I$ is a leaf if either

1. $s \notin I$. In this case, the node is considered *unsuccessful*, or
2. the node has an ancestor labeled $s' \models \mathrm{EG}I$ for some s' with $s' \preceq s$. In this case, the node is considered *successful*.
3. $s \in I$ and $post(s)$ is empty. In this case, the node is considered *successful*.

From a non-leaf node labeled $s \models \mathrm{EG}I$ we create a child labeled $s' \models \mathrm{EG}I$ for each state $s' \in maxpost(s)$. The algorithm answers "yes" if a successful node is encountered, otherwise it answers "no".

The correctness of the algorithm follows from the fact that when a successful node is encountered according to criterion 2, we can, by monotonicity, construct an infinite path where all states are in I by continuing from the ancestor node. Completeness follows by the observation that the possibly unexplored successors of a state (i.e., those in $maxpost(s)$ but not in $post(s)$ for some s) can be satisfactorily represented by "larger" states (with respect to $\preceq$) in $maxpost(s)$. The construction of the tree terminates by König's lemma, since the tree is finite branching and all branches are finite (this follows from well quasi-orderedness). We have thus proved the following theorem:

Theorem 5.1 The eventuality problem for control states is decidable for well-structured and essentially finite branching systems.

In [Fin90] an algorithm is presented to check whether the reachability tree of a well-structured system is finite. The algorithm can be seen as a variant of our algorithm to check eventuality properties as follows. We take I to be the set S of all states. A node labeled by $s \models \mathrm{EG}S$ is a leaf if either

- the node has an ancestor labeled $s' \models \mathrm{EG}S$ for some s' with $s' \preceq s$. In this case, the node is considered *successful*, or
- $post(s)$ is empty. In this case, the node is considered *unsuccessful*.

The reachability tree is finite iff no successful nodes are encountered.

6 Simulations between Infinite Systems and Finite Systems

In this section we consider the problem of whether a well-structured system is simulated by a finite transition system. A transition system is said to be *finite* if it has a finite set of states. In our algorithms we assume that a finite transition system is described by finite sets representing states and transitions.

Definition 6.1 Given two transition systems $\mathcal{L}_1 = \langle S_1, \delta_1 \rangle$ and $\mathcal{L}_2 = \langle S_2, \delta_2 \rangle$, we say that a relation $\mathcal{R} \subseteq S_1 \times S_2$ is a *simulation* (of $\mathcal{L}_1$ by $\mathcal{L}_2$) if for each $\langle s_1, s_2 \rangle \in \mathcal{R}$, $s_1' \in S_1$, and $\lambda \in \Lambda$, if $s_1 \xrightarrow{\lambda} s_1'$ then there exists $s_2' \in S_2$ such that $s_2 \xrightarrow{\lambda} s_2'$ and $\langle s_1', s_2' \rangle \in \mathcal{R}$.

Simulating an Infinite System by a Finite System For $s_1 \in S_1$ and $s_2 \in S_2$, we say that s_1 is *simulated* by s_2, denoted $s_1 \sqsubseteq s_2$, if there is a simulation $\mathcal{R}$ of $\mathcal{L}_1$ by $\mathcal{L}_2$ such that $\langle s_1, s_2 \rangle \in \mathcal{R}$.

A transition system is said to be *intersection effective* if $min(\mathcal{C}(s_1) \cap \mathcal{C}(s_2))$ is computable for any states s_1 and s_2.

Theorem 6.2 For a state s in an intersection effective well-structured transition system and a state q in a finite transition system, it is decidable whether $s \sqsubseteq q$.

Proof. The idea is to calculate the set of pairs $\langle s, q \rangle$ of states such that $s \not\sqsubseteq q$. We observe that for each q, the set $\{s \mid s \not\sqsubseteq q\}$ is an ideal. This allows us to compute the set by a fixpoint iteration analogous to that used for the reachability problem. For each state q of the finite transition system, we define a sequence $I_0^q, I_1^q, I_2^q, \ldots$, where $I_0^q = \emptyset$, and $s \in I_{j+1}^q$ if and only if either

- $s \in I_j^q$; or
- there are λ and s' such that $s \xrightarrow{\lambda} s'$ and for all q' if $q \xrightarrow{\lambda} q'$ then $s' \in I_j^{q'}$.

It is clear that I_j^q is an ideal and that $I_0^q \subseteq I_1^q \subseteq I_2^q \subseteq \cdots$. By Lemma 3.2 it follows that there is a k such that $I_{k+1}^q = I_k^q$ for all q, and $s \not\sqsubseteq q$ iff $s \in I_k^q$.

We represent I_j^q by the canonical set $M_j^q = min(I_j^q)$, where $M_0^q = \emptyset$, and

$$M_{j+1}^q = \bigcup_{\lambda} minpre_{\lambda}\left(\bigcap_{q' \in post_{\lambda}(q)} M_j^{q'}\right)$$

Note that M_{j+1}^q can be computed from M_j^q for intersection effective well-structured transition systems. We iterate until we reach a k such that $M_{k+1}^q \equiv M_k^q$. To decide whether $s \not\sqsubseteq q$ we check if $\exists s' \preceq s$ such that $s' \in M_k^q$. $\Box$

Weak Simulation The result of Theorem 6.2 can be generalized to the case of weak simulation as follows. We assume that the set of labels is extended by the silent event τ. Let $\langle S, \delta \rangle$ be a transition system. For $s_1, s_2 \in S$ and $\lambda \neq \tau$, we let $s_1 \stackrel{\lambda}{\Longrightarrow} s_2$ denote that s_2 is reachable from s_1 through a finite number of τ-transitions, followed by a λ-transition, followed by a finite number of τ-transitions. For $T \subseteq S$ and $\lambda \neq \tau$, we define $pre_{\lambda}^*(T)$ to be the set $\left\{s' \mid \exists s \in T.\ s' \stackrel{\lambda}{\Longrightarrow} s\right\}$. Analogously, we define $post^*{}_{\lambda}(T)$ as $\left\{s' \mid \exists s \in T.\ s \stackrel{\lambda}{\Longrightarrow} s'\right\}$. We let $minpre_{\lambda}^*(T)$ denote $min(pre_{\lambda}^*(\mathcal{C}(T)))$. From the discussion in Section 4, we conclude that $minpre_{\lambda}^*(\{s\})$ is computable for each $s \in S$.

The definition of simulation can be generalized to *weak simulation* by replacing the relation $\longrightarrow$ by $\Longrightarrow$.

Theorem 6.3 For a state s in an intersection effective well-structured transition system and a state q in a finite transition system, it is decidable whether s is weakly simulated by q.

Proof. We modify the algorithm in the proof of Theorem 6.2 and define

$$M^q_{j+1} = \bigcup_{\lambda} minpre^*_{\lambda} \left(\bigcap_{q' \in post^*{}_{\lambda}(q)} M^{q'}_j \right)$$

□

Simulating a Finite System by an Infinite System We consider the problem of whether a finite transition system is simulated by a well-structured system. We present an algorithm which assumes that the well-structured system is essentially finite branching. For a state s and label λ, define the set $maxpost_{\lambda}(s)$ analogously to the definition of $maxpost(s)$. To determine whether $q_0 \sqsubseteq s_0$, we construct an and-or tree as follows. The root is labeled by $q_0 \sqsubseteq s_0$. A node labeled by $q \sqsubseteq s$ is an and-node. Such a node is a leaf if either

- $post(q)$ is empty, or
- it has an ancestor labeled $q \sqsubseteq s'$ for some s' with $s \preceq s'$.

Every non-leaf and-node, with a label $q \sqsubseteq s$, has a descendant labeled $q' \ll_{\lambda} s$, for each $q' \in post_{\lambda}(q)$. A node labeled by $q' \ll_{\lambda} s$ is an or-node. Such a node is a leaf if $post_{\lambda}(s)$ is empty. Every non-leaf or-node, with a label $q' \ll_{\lambda} s$, has a descendant labeled $q' \sqsubseteq s'$, for each $s' \in maxpost_{\lambda}(s)$.

By arguments similar to those used in the eventuality algorithm, the and-or tree is always finite, and that $q_0 \sqsubseteq s_0$ if and only if the root in the and-or tree generated from $q_0 \sqsubseteq s_0$ evaluates to *true* (where or-nodes which are leaves evaluate to *false*, and and-nodes which are leaves evaluate to *true*). We have thus proved the following theorem:

Theorem 6.4 The problem whether $q_0 \sqsubseteq s_0$, for a state q_0 of a finite state system, and a state s_0 of a well-structured and essentially finite branching system, is decidable.

7 Example Models

In this section we give five examples of five computation models namely: lossy channel systems, vector addition systems with states (a model equivalent to

Petri Nets), relational automata, timed automata, and a class of timed Petri nets. For each model, we describe how it can be viewed as an infinite-state transition system, and show that it is equipped with a preorder such that it satisfies the conditions in Section 3 for being well-structured. The set S of states of each model is formed by the cartesian product of a finite set Q of control states and an infinite set D of *data values*. The set δ of transitions is derived from a finite set Cmd of *commands*, where a command corresponds to an atomic event involving a change of state while performing an observable interaction (represented by an element in the set Λ of labels) with the environment.

7.1 Lossy Channel Systems

Lossy Channel Systems (LCSs) [AJ93] are systems of finite-state processes that communicate messages from a finite alphabet M over a finite set C of unreliable unbounded FIFO channels. The channels are unreliable in the sense that they may lose messages at any time. The set Q of control states of an LCS is typically the cartesian product of the control states of the finite-state processes in the system. LCSs have been used to model and verify data transfer protocols (e.g., sliding window protocols) that are designed to tolerate message losses in channels [BZ83, Boc78].

A command in Cmd is a quadruple of the form $\langle q, op, \lambda, q' \rangle$, where $q, q' \in Q$, $\lambda \in \Lambda$, and op is an operation of one of the forms

- $c!m$: sends $m \in M$ to $c \in C$ (appends m to the end of c).
- $c?m$: receives $m \in M$ from $c \in C$ (removes c from the head of c). This operation may be performed only if m is in the head of c.
- $skip$: is the empty operation (which does not change the channel contents).

The data domain D of an LCS is the set of mappings $C \mapsto M^*$. For strings $x, y \in M^*$, we use $x \bullet y$ to denote the concatination of x and y. For $d \in D$, we use $d[c := x]$ to denote d' where $d'(c) = x$, and $d'(c') = d(c')$, for $c' \neq c$.

Let d be any data value. A command of form $\langle q, c!m, \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d[c := d(c) \bullet m] \rangle$. A command of form $\langle q, c?m, \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d[c := m \bullet c] \rangle \xrightarrow{\lambda}$

$\langle q', d\rangle$. A command of form $\langle q, skip, \lambda, q'\rangle$ gives rise to the set of transitions of form $\langle q, d\rangle \xrightarrow{\lambda} \langle q', d\rangle$. Also, if $\langle q, d\rangle \xrightarrow{\lambda} \langle q', d'\rangle$ is a (non-lossy) transition obtained from a command in one of the ways just described, then a *lossy transition* of the form $\langle q, d_1\rangle \xrightarrow{\lambda} \langle q', d'_1\rangle$ may be performed for any $d_1 \succeq d$ and any $d'_1 \preceq d'$. Intuitively, the messages in d_1 but not in d are unobservably lost immediately before the non-lossy transition and those in d' but not in d'_1 are lost after the non-lossy transition.

The preorder $\preceq$ on D (which is also a partial order) is given by $d_1 \preceq d_2$ iff for each channel c, $d_1(c)$ is a (not necessarily contiguous) substring of $d_2(c)$. Now we show that LCSs are well-structured systems.

- *monotonicity:* follows from the existence of lossy transitions, implying that a state may lose any number of messages transforming to a smaller state and then perform all the transitions of the smaller state.
- *well quasi-orderedness:* holds by a beautiful result which states that the substring relation among strings over a finite alphabet is a well quasi-ordering [Hig52].
- *computability of $minpre_\lambda$:* The set $minpre_\lambda(\langle q, d\rangle)$ is defined to be $min(T)$, where T is the smallest set containing the following states:
  - $\langle q', d'\rangle$, if $\langle q', c!m, \lambda, q\rangle \in Cmd$ and $d = d'[c := d'(c) \bullet m]$.
  - $\langle q', d'\rangle$, if $\langle q', c!m, \lambda, q\rangle \in Cmd$, $d = d'$ and the last element of $d(c)$ is not equal to m.
  - $\langle q', d'\rangle$, if $\langle q', c!m, \lambda, q\rangle \in Cmd$, $d = d'$ and $d(c)$ is equal to the empty string.
  - $\langle q', d'\rangle$, if $\langle q', c?m, \lambda, q\rangle \in Cmd$ and $d' = d[c := m \bullet d(c)]$.
  - $\langle q', d'\rangle$, if $\langle q', skip, \lambda, q\rangle \in Cmd$ and $d = d'$.

Intersection effectiveness and essentially finite branching are obvious.

The decidability of control state reachability and eventuality properties is shown in [AJ93], while the decidability of simulation with finite transition systems in both directions is shown in [AK95].

7.2 Vector Addition Systems with States (Petri Nets)

A Vector Addition System with States (abbreviated VASS) models a finite-state machine operating on a finite number of variables which range over the natural numbers.

In a VASS, the data domain D is the set of k-dimensional vectors $n = \langle n_1, n_2, \ldots, n_k \rangle$, where each n_i is a natural number. We use $n[i]$ to denote n_i. We apply addition, subtraction, and relational operators pointwise on k-dimensional vectors. We let $\overline{0}$ denote the vector which is constantly 0. The preorder (in fact partial order) $\preceq$ on D is defined as $n_1 \preceq n_2$ iff $n_1 \leq n_2$.

A command in Cmd is of the form $\langle q, n_1, n_2, \lambda, q' \rangle$, where $q, q' \in Q$, $\lambda \in \Lambda$, and n_1, n_2 are k-dimensional vectors. The set δ then contains all transitions of form $\langle q, n_3 \rangle \stackrel{\lambda}{\longrightarrow} \langle q', n_3 - n_1 + n_2 \rangle$ such that $\langle q, n_1, n_2, \lambda, q' \rangle$ is a command and $\overline{0} \leq n_3 - n_1$.

For a VASS, monotonicity, intersection effectiveness, and essentially finite branching are obvious. Well quasi-orderedness is a special case of that for lossy channel systems. The set $minpre_\lambda(\langle q, d \rangle)$ is defined to be $min(T)$, where T is the set of elements of the form $\langle q', n' \rangle$ such that $\langle q', n_1, n_2, \lambda, q \rangle \in Cmd$ and $n'[i] = \max(0, n[i] + n_2[i] - n_1[i])$ for each $1 \leq i \leq k$.

Control state reachability (often called coverability in the Petri net literature) and eventuality properties for VASSs can also be decided by the Karp-Miller algorithm [KM69]. The control state reachability algorithm we present in this paper performs backward reachability analysis, and can be considered as an alternative to the Karp-Miller algorithm which uses forward reachability analysis. The decidability of simulation with finite transition systems in both directions is shown in [JM95].

Basic Parallel Processes (BPPs) A *Basic Parallel Process (BPP)* [CHM93] is defined by a finite set of equations of the form $X_i = E_i$, $i \in \{1, \ldots, n\}$, where X_i is a *process variable* and E_i is a *process expression*. The set of process expressions contain the inactive process $\mathbf{0}$, process variables, and is closed under action prefixing, non-deterministic choice, and parallel composition. Basic parallel process form a special class of VASSs [Mol96] and are therefore well-structured systems.

7.3 Real-Time Automata

The data part of a *real-time automaton* consists of the set of mappings from a finite set X of *clocks* to the set of nonnegative real numbers. Real-time automata have in recent years become important for modelling and analysis of time-dependent systems. The exact definitions of real-time automata vary slightly. Here we give a typical presentation. A command in Cmd is of the form $\langle q, \alpha, \lambda, q' \rangle$, where $q, q' \in Q$, $\lambda \in \Lambda$, and α is a guarded command of form $g \rightarrow stmt$ in which

- the guard g is a boolean combination of inequalities of form $x \sim n$, where $x \in X$, n is an integer and and $\sim$ is one of the relations $\geq$, $\leq$, $<$, or $>$.
- the body $stmt$ for each $x \in X$ contains an assignment of one of the forms $x := x$ or $x := 0$.

Each command $\langle q, (g \rightarrow stmt), \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d' \rangle$, where the values of clocks in d satisfy g, and where the value of x in d' is obtained by performing the assignments in $stmt$. The advance of time is represented by delay transitions of form $\langle q, d \rangle \xrightarrow{\tau} \langle q, d' \rangle$ in which all clocks advance by the same (real-valued) amount, subject to the condition that not all guards of transitions from the control state q are made false. In some variations of real-time automata, delay transitions are instead constrained by an invariant conditions over the clocks in each control state, such that this invariant must not be violated in a delay transition.

As the preorder $\preceq$ we take the equivalence relation on clock states, introduced in [ACD90]. This is the largest congruence induced by predicates of the form $x \geq n$, $x \leq n$, and $x_1 - x_2 \geq n$ for clocks x and any nonnegative integer n which is at most equal to the largest constant that occur syntactically in commands of the automaton.

It can be shown that $\preceq$ is an equivalence, which is also a bisimulation, and hence the transition system is monotonic. The system is well quasi-ordered since there are finitely many equivalence classes. The computation rules for $minpre_\lambda$, intersection effectiveness, and the essentially finite branching property are also rather straightforward.

7.4 Relational Automata

A Relational Automaton (RA) is a computing device which besides its finite control structure possesses a finite number of data variables each one taking its values from some (possibly infinite) ordered domain. The operations that the automaton can perform on the data variables are comparison and assignment. The ordering of the data values stored into the variables during the runtime may influence the control flow of the automaton.

A (rational) relational automaton (QRA) has a finite set X of *data variables* that assume values in the set Q of rational numbers. A command in Cmd is a quadruple of the form $\langle q, \alpha, \lambda, q' \rangle$ where $q, q' \in Q$, $\lambda \in \Lambda$, and α is a guarded command of form $g \to stmt$ in which

- the guard g is a boolean combination of inequalities of form $x < y$, $x < c$ and $c < y$ for $x, y \in X$ and $c \in Q$, and where
- the body $stmt$ contains, for each $x \in X$, an assignment of one of the forms $x := y$, $x := c$, or $x := \{?\}$ for $y \in X$ and $c \in Q$.

Given a QRA P, we let $Cons(P)$ be the set of all constants that occur syntactically in the commands of P (clearly, there is only a finite number of those). The set D of data values is the set $X \mapsto Q$ of possible combinations of values of the data variables. For $c \in Q$, we use the convention that $d(c) = c$ for any data value d. Each command $\langle q, (g \to stmt), \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d' \rangle$, where the values of variables in d satisfy g, and where the value of x in d' is equal to

- $d(y)$ if $stmt$ contains $x := y$, for $y \in X \cup Cons(P)$.
- any element of Q if $stmt$ contains $x := \{?\}$.

We say that two data values $d, d' \in (X \mapsto Q)$ are *equivalent* if for all x, y in $(X \cup Cons(P))$ we have $d(x) \leq d(y)$ iff $d'(x) \leq d'(y)$. In other words, two data values are equivalent if the relative ordering between the data variables and constants of the program is the same. It turns out that by taking $\preceq$ to be this equivalence on D, each QRA becomes a well-structured system. The equivalence $\preceq$ has a finite number of equivalence classes.

We will also consider the variation of relational automata, called Integral Relational Automata (IRA), obtained by replacing the data domain by the set of integers. In the case of IRAs, the situation is slightly more complicated, due to the fact that the ordering on the set of integers is not dense. For IRAs there is no equivalence relation which can be taken as $\preceq$ (see [Č94]).

For an IRA P let c_{min} and c_{max} be, respectively, the smallest and the largest constants that occur syntactically in the commands of P.

We say that a data value $d \in (X \mapsto I)$ is *sparser than* a data value $d' \in (X \mapsto I)$ if for all x, y in $(X \cup Cons(P))$ we have

- $d(x) \leq d(y)$ iff $d'(x) \leq d'(y)$, and
- $|d(y) - d(x)| \geq |d'(y) - d'(x)|$ for all $x, y \in X \cup Cons(P)$.

For instance, if $Cons(P) = \{0, 1, 2\}$, then the data vector $\langle 2, 10, 12, 1997\rangle$ is sparser than $\langle 2, 4, 6, 1000\rangle$, but not sparser than $\langle 1, 10, 12, 1997\rangle$ since the value of the first variable is no longer equal to the constant 2, and not sparser than $\langle 2, 4, 7, 17\rangle$ since $7 - 4 > 12 - 10$. If $Cons(P) \neq \emptyset$, then the *sparser than* relation is a partial order. If $Cons(P) = \emptyset$ then the relation is a preorder, but neither an equivalence, nor a partial order.

We define the relation $\preceq$ by $d \preceq d'$ iff d' is sparser than d and show that $\preceq$ meets the requirements for well structured systems.

- *monotonicity:* Suppose that $\langle q, d_1\rangle \xrightarrow{\lambda} \langle q', d_1'\rangle$ and $d_1 \preceq d_2$. We show that there exists $d_2' : d_1' \preceq d_2'$ such that $\langle q, d_2\rangle \xrightarrow{\lambda} \langle q', d_2'\rangle$. We know that there is a tuple $\langle q, g \rightarrow stmt, \lambda, q'\rangle \in Cmd$, where d_1 satisfies g and d_1' is derived from d_1 according to $stmt$. By $d_1 \preceq d_2$ it is clear that d_2 satisfies g. We define d_2' as follows. Since $d_1 \preceq d_2$ we can define a mapping $\rho : I \mapsto I$ such that

  - $\rho(d_1(x)) = d_2(x)$ for all $x \in (X \cup Cons(P))$, and
  - $|\rho(a) - \rho(b)| \geq |a - b|$ for all $a, b \in I$.

  For any $x \in X$ we let $d_2'(x) = \rho(d_1'(x))$. It follows that $\langle q, d_2\rangle \xrightarrow{\lambda} \langle q', d_2'\rangle$ and $d_1' \preceq d_2'$.

- *well quasi-orderedness:* Consider an infinite sequence $d_0, d_1, d_2, \ldots$ of data values. We observe that there is an infinite increasing sequence $i_0, i_1, i_2, \ldots$ such that the variables and constants have the same relative ordering in each d_{i_j}. In other words, $d_{i_j}(x) \leq d_{i_j}(y)$ iff $d_{i_k}(x) \leq d_{i_k}(y)$

for each $j, k \geq 0$ and $x, y \in X \cup Cons(P)$. This implies that $d_{i_j} \preceq d_{i_k}$ iff $|d_{i_j}(x) - d_{i_j}(y)| \leq |d_{i_j}(x) - d_{i_j}(y)|$ for all $x, y \in X \cup Cons(P)$. Since $|d_{i_j}(x) - d_{i_j}(y)|$ is a natural number, the result follows as a special case of that for the substring relation.

- *computability of* $minpre_\lambda$*:* We define $minpre_\lambda(\{\langle q, d\rangle\})$ to be $min(T)$, where T is the smallest set with elements satisfying the following properties. Let $\langle q', g \to stmt, \lambda, q\rangle \in Cmd$. Let $\{X_1, X_2\}$ be a partitioning of X such that $x \in X_1$ iff x appears in the right hand side of an assignment operation in $stmt$. Let k be the number of elements in X_2. We say that a data element d_0 is *k-close* to d iff $d \preceq d_0$ and $|d_0(x) - d_0(y)| - |d(x) - d(y)| \leq k$ and $|d_0(x) - d(x)| \leq k$ for all $x, y \in X \cup Cons(P)$. It is clear that for any d the set of k-close d_0 is finite.

  The set T contains each element $\langle q', d'\rangle$ such that d' satisfies g and for some d_0 which is k-close to d:

  - if $y := x$ occurs in $stmt$ then $d'(x) = d_0(y)$, and
  - if $x \in X_2$ then $d'(x) \in \{d_{min} - k, \ldots, d_{max} + k\}$, where d_{min} and d_{max} are the minimal and maximal values of $d'(y)$ for $y \in X_1 \cup Cons(P)$.

Intersection effectiveness of IRA is straightforward, however IRA are not necessarily essentially finite branching due to the presence of random assignment.

The decidability of control state reachability for IRA is shown in [BBK74, BBK77]. A detailed study of checking properties of IRA, which includes the decidability of eventuality properties can be found in [Č94]. The decidability of simulation of an IRA by a finite transition system has not been published before.

7.5 Timed Petri Nets

By taking a composition (Section 3.3) of the Petri net model (Section 7.2) and real-time automata, one can obtain a natural semantic variant of Timed Petri Nets [MF76, GMMP91, BD91]. In our model, a *Timed Petri Net (TPN)* operates on a finite set of real-valued *clocks* and a finite number of *variables* which range over the natural numbers. The data part of a TPN consists of the set of pairs of mappings of the form $\langle d_1, d_2 \rangle$ where d_1 maps each clock to a real number, and d_2 maps each variable to a natural number. The preorder $\preceq$ among data values is defined such that $\langle d_1, d_2 \rangle \preceq \langle d_1', d_2' \rangle$ if d_1 is equivalent to d_1', according to the equivalence relation defined in Section 7.3, and d_2 is less than or equal to d_2', according to the partial order defined in Section 7.2. Each transition is either a delay transition which increases the values of the clocks by the same real number, or a discrete transition which changes the values of the clocks and variables according to the rules given in Section 7.3 and Section 7.2 respectively. In a similar manner to the works in [MF76, GMMP91, BD91] we can model the class of time Petri nets where every transition t has an earliest firing time t_1 and a latest firing time t_2. In order for t to be fired it needs to be continuously enabled for some time $t \in [t_1, t_2]$. However, in contrast to the models in [MF76, GMMP91, BD91], we do not allow overlapping waiting intervals for multiple instances of the same transition. It is precisely this restriction which makes reachability analysis possible in our case but impossible in the above approaches.

8 Conclusions and Future Research

We provide a unified approach for algorithmic verification of several classes of infinite-state systems. Our method generalizes verification methods based on finite partitioning in the sense that (i) we consider a well quasi-ordered state space instead of a finitely partitioned one, and (ii) we consider states which are related through simulation instead of bisimulation. We are able to explain and derive several, seemingly diverse, algorithms for verification of different classes of infinite-state systems in a uniform manner. We aim to extend the applicability of our results to derive novel algorithms for verification of new classes of infinite state states such as parametrized networks of processes, programs with multi-sorted domains, etc.

Acknowledgment

We are grateful to S. Purushothaman-Iyer for many interesting comments and discussions.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5^{th} IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, Philadelphia, 1990.

[AH89] R. Alur and T. Henzinger. A really temporal logic. In *Proc. 30^{th} Annual Symp. Foundations of Computer Science*, pages 164–169, 1989.

[AJ93] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proc. 8^{th} IEEE Int. Symp. on Logic in Computer Science*, 1993. Extended Abstract.

[AK95] Parosh Aziz Abdulla and Mats Kindahl. Decidability of simulation and bisimulation between lossy channel systems and finite state systems. In Lee and Smolka, editors, *Proc. CONCUR '95, 6^{th} Int. Conf. on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 333 – 347. Springer Verlag, 1995. Extended Abstract.

[BBK74] J. M. Barzdin, J. J. Bicevskis, and A. A. Kalninsh. Construction of complete sample systems for program testing. *Latv. Gosudarst. Univ. Uch. Zapiski*, 210, 1974. in Russian.

[BBK77] J. M. Barzdin, J. J. Bicevskis, and A. A. Kalninsh. Automatic construction of complete sample systems for program testing. In *IFIP Congress, 1977*, 1977.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.

[Boc78] G. V. Bochman. Finite state description of communicating protocols. *Computer Networks*, 2:361–371, 1978.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, April 1983.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4^{th} ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[CG87] E. M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking algorithms. In *Proc. 6^{th} ACM Symp. on Principles of Distributed Computing, Vancouver, Canada*, pages 294–303, 1987.

[CGL92] E. M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *Proc. 19^{th} ACM Symp. on Principles of Programming Languages*, 1992.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In *Proc. CONCUR '93, Theories of Concurrency: Unification and Extension*, pages 143–157, 1993.

[Č92a] K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In *Proc. Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 302–315, 1992.

[Č92b] K. Čerāns. Feasibility of finite and infinite paths in data dependent programs. In *LFCS'92*, volume 620 of *Lecture Notes in Computer Science*, pages 69–80, 1992.

[Č94] K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *Proc. ICALP '94*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer Verlag, 1994.

[DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In *Proc. IFIP working conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, 1994.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, (89):144–179, 1990.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè. A unified high-level petri net formalism for time-critical systems. *IEEE Trans. on Software Engineering*, 17(2):160–172, 1991.

[Hen95] T.A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. ICALP '95*, 1995.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.

[Jan90] P. Jančar. Decidability of a temporal logic problem for petri nets. *Theoretical Computer Science*, 74:71–93, 1990.

[JM95] P. Jančar and F. Moller. Checking regular properties of petri nets. In *Proc. CONCUR '95, 6^{th} Int. Conf. on Concurrency Theory*, pages 348–362, 1995.

[JN94] Neil D. Jones and Flemming Nielson. Abstract interpretation: a semantics-based tool for program analysis. In *Handbook of Logic in Computer Science*. Oxford University Press, 1994.

[JP93] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 107(2):272–302, Dec. 1993.

[KM69] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and Systems Sciences*, 3(2):147–195, May 1969.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Boujjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, (6):11–44, 1995.

[MF76] P. Merlin and D.J. Farber. Redoverability of communication protocols - implications of a theoretical study. *IEEE Trans. on Computers*, COM-24:1036–1043, Sept. 1976.

[Mol96] F. Moller. Decidability results and algorithms for infinite state automata, 1996. Lecture Notes for the 1996 European School on Logic, Language, and Information.

[PP92] W. Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Informatica*, 29(6/7):499–522, June 1992.