General Decidability Theorems for Infinite-State Systems *

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@erika.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 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 well-ordered and well-founded preorder 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.

*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.

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 [3, 4, 18], hybrid automata [9], data-independent systems [15, 21], relational automata ([13, 19, 20]), Petri nets ([11, 12]), lossy channel systems [1, 2]. As the interest in this area increases, it will be important to extract common principles that underly 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 finding a finite-state system, which is in some simulation or bisimulation relation with the original system. This could

be achieved e.g. by partitioning the state space of the original system into equivalence classes under bisimulation [15, 3, 22, 9]. However, 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 *well-founded 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 well-founded 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; and (ii) the preorder on the data domain is a well-ordering, which means that each infinite ascending chain contains an element which is larger than or equivalent to an earlier element in the chain. We call the class of systems satisfying these properties *well-structured systems*.

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 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-founded and well-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-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 [5, 17, 7]. 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 do not give general methods for constructing abstractions, and are not concerned primarily with constructing decision procedures for verification as we do.

Comparing with finite partitioning methods, we observe that an equivalence is a preorder relation which in addition is symmetric. This means that, apart from systems whose state spaces can be finitely partitioned, e.g. timed automata [3, 18], various classes of hybrid automata [9], data-independent systems [15], and rational relational automata [20], our methods can be used to analyze systems which do not allow for finite partitioning, such as Petri nets [12], lossy channel systems [1], and integral relational automata [20].

Finkel [8] shows that, for well-structured systems, it is decidable whether a system has a finite reachability tree. 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 solved in [8] using a generalization of the Karp-Miller algorithm [16]. This algorithm depends on strict monotonicity, which does not hold in general for well-structured systems, and hence the Karp-Miller algorithm cannot be applied.

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.

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 $pre(\{s\})$ ($post(\{s\})$).

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. By $a \prec b$ we mean $a \preceq b$ and $b \not\preceq a$. 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 *well-founded* if there is no infinite sequence $a_1 \succ a_2 \succ a_3 \succ \cdots$. 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. It is easy to establish that if $\preceq$ is well-founded, then for each set $A \subseteq D$ there exists at least one (possibly infinite) minor set of A. We use *min* to denote a function which, given a set A, returns a minor set of A.

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.

3.2. Well-structured Systems

In our framework we require that the set D of data values is equipped with a decidable well-founded 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 well-founded 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 $s_1 \xrightarrow{\lambda} s_3$, then there exists s_4 such that $s_3 \preceq s_4$ and $s_2 \xrightarrow{\lambda} s_4$.

A preorder $\preceq$ is said to be a *well-ordering*, if there is no infinite sequence $s_0, s_1, s_2, \ldots$, in which $s_i \not\preceq s_j$ for all $i < j$.

Definition 3.1. A transition system $\mathcal{L} = \langle S, \delta \rangle$, assuming a decidable and well-founded 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-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 always a finite set if $\preceq$ is well-ordered. We define $minpre_\lambda(s)$ as notation for $min(pre_\lambda(\mathcal{C}(\{s\})))$. 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.

Lemma 3.2. *If $\langle S, \delta \rangle$ is a monotonic transition system, and I is an ideal in S, then $pre_\lambda(I)$ and $pre(I)$ are also ideals in S.*

Lemma 3.3. *If $\preceq$ is well-ordered, then each canonical set is finite.*

It follows that every minor set of a set is finite. Notice, however, that there may still be infinitely many such minor sets.

Lemma 3.4. *For a well-ordered transition system and an infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ of ideals there is a k such that $I_k = I_{k+1}$*

Proof. Suppose that no such k exists. It follows that there is a sequence $s_0, s_1, s_2, \ldots$ of states such that for all $k \geq 0$ we have $s_k \in I_k$ and $s_k \notin I_j$ for all $j < k$. This means $s_j \not\preceq s_k$ for $j < k$, otherwise $s_k \in I_j$, since I_j is an ideal. This is a contradiction since the sequence $s_0, s_1, s_2, \ldots$ will then violate the well-orderedness assumption. □

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.

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 3.2 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 3.4 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 3.3 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 $s' \preceq s$. □

Abstract Interpretation The above analysis algorithm can also be phrased in terms of abstract interpretation [6, 14]. 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 $C(M) \subseteq C(M')$. 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 $\gamma(M) = C(M)$ 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 $\mu X.\ min(I) \cup minpre(X)$ in the lattice $\langle \mathcal{M}, \sqsubseteq \rangle$. 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-orderedness of $\preceq$ implies that all ascending chains in $\langle \mathcal{M}, \sqsubseteq \rangle$ are finite, thus guaranteeing convergence of any minimal fixpoint computation.

5. Eventuality Properties

In this section we describe an algorithm for deciding whether each execution 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 an infinite path 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 $s \models$ EGI. The root node is labeled by $s_0 \models$ EGI. A node labeled by $s \models$ EGI is a leaf if either

- $s \notin I$. In this case, the node is considered *unsuccessful*, or
- the node has an ancestor labeled $s' \models$ EGI for some s' with $s' \preceq s$. In this case, the node is considered *successful*.

From a non-leaf node labeled $s \models$ EGI we create a child labeled $s' \models$ EGI or 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, 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. 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.*

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 $\langle s_1', s_2' \rangle \in \mathcal{R}$.

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 3.4 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 $s' \in M_k^q$. □

The above result can easily be extended to the case of weak simulation.

We next 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 a leaf if either

- there is a transition $q \xrightarrow{\lambda} q'$ such that no transition from s is labeled by λ, in which case the node is *unsuccessful*, or
- it has an ancestor labeled $q \sqsubseteq s'$ for some s' with $s \preceq s'$, in which case the node is *successful*.

Each non-leaf node labeled $q \sqsubseteq s$ is an and-node, which for each transition $q \xrightarrow{\lambda} q'$ has a descendant labeled $q' \ll_\lambda s$. Each node labeled $q' \ll_\lambda s$ is an or-node, which for each $s' \in maxpost_\lambda(s)$ has a descendant labeled $q' \sqsubseteq s'$.

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 unsuccessful leaves evaluate to *false* and successful leaves evaluate to *true*). We have thus proved the following theorem:

Theorem 6.3. *The problem whether $q_0 \sqsubseteq s_0$ for a state q_0 of a finite state system is decidable when s_0 is a state of a well-structured and essentially finite branching system.*

7. Example Models

In this section we give four examples of computation models namely: lossy channel systems, vector addition systems with states (a model equivalent to Petri Nets), relational automata, and timed automata. 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 three conditions in Section 3 for being well-structured.

7.1 Lossy Channel Systems

Lossy Channel Systems (LCSs) [1] are systems of finite-state processes that communicate messages (from a finite alphabet M) over unreliable unbounded FIFO channels. The channels are unreliable in the sense that they may lose messages at any time. 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.

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. The data domain D of an LCS is the set of mappings from channel names to finite sequences of messages. The preorder $\preceq$ on D (which is also a partial order) is given by $d_1 \preceq d_2$ iff for each channel c, the content of c in d_1 is a (not necessarily contiguous) substring of the content of c in d_2. The set δ of transitions is obtained from a finite set of commands. Each command is a tuple of the form $\langle q_1, \alpha, \lambda, q_2 \rangle$, where $q_1, q_2 \in Q$, $\lambda \in \Lambda$, and α is an operation of one of the following forms:

- $c!m$ (which sends a message m to channel c),
- $c?m$ (which receives a message m from channel c), or
- the empty operation e (which does not change the channel contents).

A command of form $\langle q, c!m, \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d' \rangle$, where d' is d except that the message m has been added to channel c. A command of form $\langle q, c?m, \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d' \rangle$, where the message m must be at the head of channel c in state d, and where d' is d except that the message m has been removed from channel c. A command of form $\langle q, e, \lambda, q' \rangle$ gives rise to the set of transitions of form $\langle q, d \rangle \xrightarrow{\lambda} \langle q', d \rangle$, where d is any channel state. Finally, 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 $\langle q, d_1 \rangle \xrightarrow{\lambda} \langle q', d_1' \rangle$ for any $d_1 \succeq d$ and any $d_1' \preceq d'$. 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.

In [1], we show that LCSs are well-structured systems. To see this, we first note that $\preceq$ is decidable and well-founded. Well-orderedness holds by a beautiful result which states that the substring relation among strings over a finite alphabet is a well-ordering [10]. The rules for computing $minpre_\lambda$ may be found in [1]. Intersection effectiveness and essentially finite branching are obvious.

The decidability of control state reachability and eventuality properties is shown in [1], while the decidability of simulation with finite transition systems in both directions is shown in [2].

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 define 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 $n_1 \preceq n_2$ if $n_1 \leq n_2$.

The set δ of transitions is obtained from a finite set of commands of form $\langle q_1, \lambda, n_1, n_2, q_2 \rangle$, where $q_1, q_2 \in Q$, $\lambda \in \Lambda$, and n_1, n_2 are k-dimensional vectors. The set δ then contains all transitions of form $\langle q, n_3 \rangle \xrightarrow{\lambda} \langle q', n_3 - n_1 + n_2 \rangle$ such that $\langle q_1, \lambda, n_1, n_2, q_2 \rangle$ is a command and $\overline{0} \leq n_3 - n_1$.

For a VASS, monotonicity, intersection effectiveness, and essentially finite branching are obvious. Well-orderedness and the rules for computing $minpre_\lambda$ can be seen as special cases of those for lossy channel systems.

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 [16]. 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 [12].

7.3 Real-Time Automata

The data part of a *real-time automaton* consists of the set of mappings from a finite set 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.

The set δ of transitions is obtained from a finite set of commands of form $\langle q, \alpha, \lambda, q' \rangle$, where 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 $\sim$ is a relation in $\geq$, $\leq$, $<$, $>$, and n is an integer where
- 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 [3]. 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-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 relational automaton (QRA) has a finite set X of *data variables* that assume values in the set Q of rational numbers. The set D of data values is the set $X \mapsto Q$ of possible combinations of values of the data variables. The set δ of transitions is obtained from a finite set of commands of form $\langle q, \alpha, \lambda, q' \rangle$, where α is a guarded command of form $g \rightarrow stmt$ 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 $c \in Q$.

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 variables in d satisfy g, and where the value of x in d' is equal to

- the value of x in d if $stmt$ contains $x := x$,
- the value of y in d if $stmt$ contains $x := y$,
- c if $stmt$ contains $x := c$,
- any element of Q if $stmt$ contains $x := \{?\}$.

Given a QRA, P, let $Cons(P)$ be the set of all constants in Q that occur syntactically in the commands of P (clearly, there is only a finite number of those). For $c \in Q$, we use the convention that $d(c) = c$ for any data value d. 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 Q 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 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. Let $Cons(P) = \{c_{min}, \ldots, c_{max}\}$, i.e. all integers between and including c_{min} and c_{max}. 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
- whenever $d(x) \leq d(y)$ then $d(y) - d(x) \geq d'(y) - d'(x)$.

For instance, if $Cons(P) = \{0, 1, 2\}$, then the data vector $\langle 2, 10, 12, 1995 \rangle$ is sparser than $\langle 2, 4, 6, 1000 \rangle$, but not sparser than $\langle 1, 10, 12, 1995 \rangle$ 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 $7 - 4 > 12 - 10$.

It can be verified that the relation $\preceq$ defined by $d \preceq d'$ iff d' is sparser than d meets the requirements for well-structured systems. However, it does not make IRAs essentially finite branching due to the presence of random assignment. More details about properties of $\preceq$ can be found in [20], where the decidability of the control state reachability and eventuality problems for IRAs is shown. The decidability of simulation of an IRA by a finite transition system has not been published before.

Acknowledgment

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

References

[1] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. 8^{th} IEEE Int. Symp. on Logic in Computer Science*, 1993. Extended Abstract.

[2] P. A. Abdulla and M. 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.

[3] 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.

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

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

[6] 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.

[7] 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.

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

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

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

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

[12] 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.

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

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] P. Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13^{th} ACM Symp. on Principles of Programming Languages*, pages 184–193, Jan. 1986.

[22] W. Yi. CCS + Time = an interleaving model for real time systems. In L. Albert, Monien, and R. Artalejo, editors, *Proc. ICALP '91*, volume 510 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.