Verifying Safety Properties of a Class of Infinite-State Distributed Algorithms *

Bengt Jonsson and Lars Kempe

Uppsala University, Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden

Abstract. We consider the problem of verifying correctness properties of a class of programs with states that are sets of ground atoms. Such programs can model specifications of telephone services, in which we are particularly interested. For this class of systems, we consider the problem of checking reachability properties. A large class of safety properties can also be reduced to the problem of checking reachability in a transformed system. The emphasis of our approach is on automated verification of such properties. Although the reachability problem is in general undecidable, we present a method for analyzing reachability properties, and show that it can be successfully applied to practical examples. The main idea of our method is the following. In order to check whether a certain set of "error" states can be reached from an initial state of the system, we first compute the set of "unsafe states" (i.e., states from which it is possible to reach an error state) as a fixpoint, and finally we prove that the initial state is not "unsafe". We present the application of our method to an example of a simple telephone service.

1 Introduction

Most established approaches to automated verification of distributed systems are based on finite-state methods. In these approaches systems are modeled as finite state programs by giving an explicit representation of the state space. Various tools for the analysis of state spaces exist. A major problem in these approaches is *the state explosion problem*, which is most prominent in systems that consist of many parallel processes. Another limitation of these approaches is the fact that only finite-state programs can be checked for correctness. Systems with infinitely many states, e.g. systems that operate on data from unbounded domains, fall beyond the capabilities of these methods. In general, verification of infinite-state systems requires a substantial manual effort, since most interesting verification problems are undecidable. However, algorithmic verification methods have recently been developed for some classes of infinite-state systems. Examples include certain types of real-time systems that operate on clocks [3, 18], data-independent systems [15, 19], systems with many identical processes [11, 13, 17],

* Supported in part by the Swedish Board for Industrial and Technical Development (NUTEK) as part of ESPRIT BRA project No. 6021 (REACT), and as part of grant No. 5321-93-3061 (on Feature Interaction).

context-free processes [8, 10, 9], Petri nets [14], and systems communicating over unbounded lossy channels [1, 2].

In this paper, we consider the problem of verifying systems that can be abstractly viewed as programs whose state is a set of relations over a possibly unbounded universe. A state change consists of changing how relations hold between objects in the universe. Such programs can model many distributed algorithms. We are particularly interested in the specification of telephone services, and the subscriber's view of a telephone system can conveniently be modeled as a dynamically changing set of relations, where the relations can be connections between users of the system. We are presently initiating this approach for application to specifications of telephone services in parallel with an effort to model telephone services [6]. Of particular interest is the so-called *feature interaction problem* [7], manifesting itself in that two services (features) of a telephone system may be in conflict with each other.

For this class of programs, we concentrate on verifying safety properties, and in particular, the reachability of sets of states. The problem of verifying safety properties can be transformed into a reachability problem.

For the class of programs that we consider in this paper, the reachability problem is undecidable in general (it is not difficult to show that they can simulate Turing machines). We present a method for proving reachability properties, and give some first results on proving some properties of basic telephone services. In order to check whether a certain set of "error" states can be reached from an initial state of the system, we first compute the set of "unsafe states" (i.e., states from which it is possible to reach an error state) as a fixpoint, and finally we prove that the initial state is not "unsafe". In the cases where the fixpoint computation terminates the method determines whether the set of "error" states is reachable. This approach to verification is, in principle, similar to the one used for lossy channel systems in [1], where the fixpoint computation always terminates, and for hybrid system in [4] where the fixpoint computation may not always terminate.

Sets of states (e.g., the "unsafe states") are represented by formulas restricted to a certain form. Some of the restrictions are motivated by simplicity reasons, whereas other restrictions are required by the analysis method given in Section 3. In spite of the restrictions, many useful properties can be checked. Intuitively, a formula describes a finite configuration by sets of atoms that should exists in a state. The formula is restricted to an existentially quantified conjunction of atoms, where an atom is a (possibly negated) relation symbol applied to variables. With this representation of sets of states, we can perform the basic operations in the verification method.

Alternative approaches for automated verification of telephone systems usually use finite-state techniques (for example [16]). These approaches are in most cases adequate for detecting interactions, but to prove the absence of interactions for an arbitrary number of subscribers these approaches need additional reasoning such as induction techniques and data-independence analysis that in many cases can not be automated.

The paper is structured as follows. In the next section, we present the basic syntax for the kind of programs we consider, together with the semantic model for our programs. In Section 3 we present the verification method, and in Section 4 we apply the method to show some properties of a basic telephone service.

2 Transition Systems

2.1 Syntax

In this section, we present the class of transition systems that we want to use. We assume some infinite set D of *objects*, ranged over by $d, d_1, d_2, \ldots$. We assume a finite set of *unary relations* ranged over by $a, a_1, a_2, \ldots$, and a finite set of *binary relations* ranged over by $b, b_1, b_2, \ldots$. Assume a (countably infinite) set of variables, typically ranged over by $x, y, z, \ldots$.

A *ground atom* is a formula of the form $a(d)$ or $b(d_1, d_2)$ where d, d_1, d_2 are objects in D. $a(d)$ is called a unary ground atom, and $b(d_1, d_2)$ is called a binary ground atom. A *state* s is a set of ground atoms. A state is *finite* if it contains a finite set of ground atoms. One can regard a state s as giving an interpretation to the unary and binary relations, so that a unary relation a is interpreted as the set of objects $\{ d \in D : a(d) \in s \}$, and a binary relation b is interpreted as the set of pairs $\{ (d_1, d_2) \in D \times D : b(d_1, d_2) \in s \}$. For simplicity reasons we restrict the relations to be unary or binary. The framework can easily be extended for n-ary relations.

Example 2.1 Configurations of a telephone system can be modeled by one unary and three binary relations:

active(d) indicates that subscriber d has his phone off the hook.
trying(d_1, d_2) indicates that subscriber d_1 tries to call subscriber d_2 who has not answered but is not busy.
busy(d_1, d_2) indicates that subscriber d_1 tries to call subscriber d_2 who is busy.
conn(d_1, d_2) indicates that there is a connection between subscribers d_1 and d_2. The connection is ordered, so d_1 is the caller and d_2 is the callee. □

We generalize ground atoms to *positive simple literals*, which are formulas of the form $a(u)$ or $b(u, v)$, where u, v ranges over objects or variables. A *negative simple literal* on the other hand is of the form $\neg a(u)$ or $\neg b(u, v)$ where u, v are objects or variables. A negative simple literal represents negative information, e.g., $\neg$*active*(d) means that the the relation *active* does not hold for object d. A *simple literal* is a positive or negative simple literal.

A *positive existential literal* is a formula of the form $\exists z : b(z, u)$ or $\exists z : b(u, z)$, denoted respectively $\overrightarrow{b}\ (u)$ or $\overleftarrow{b}\ (u)$ where u is an object or a variable. A positive existential literal says that an object is related to some other object by a certain relation, but we do not care to which object. We define *negative existential literals* and *existential literals* analogously. A *literal* is either a simple literal or an existential literal.

We can now define a *formula* over these literals as a (possibly existentially quantified) boolean combination of literals. Let φ, ψ, ρ range over formulas. A formula φ is *object free* if it contains no objects in the literals. As a notational convention we let $\psi \in \varphi$ denote that ψ is one of the conjuncts (disjuncts) if φ is a conjunction (disjunction) of formulas.

Example 2.2 The formula $\exists x, y : \mathit{active}(x) \wedge \mathit{trying}(x, y)$ says that there is an object d_1 in the relation *active* such that d_1 is related to some object d_2 by the relation *trying*. □

We define free and bound variables of formulas in the usual way. Let $FV(\varphi)$ denote the set of free variables of the formula φ. We sometimes write a formula as $\varphi(X)$ to denote that that $FV(\varphi) \subseteq X$. Formulas with at least one free variable are said to be *open*. Formulas without free variables are said to be *closed*.

Define a *replacement* as a mapping from strings of symbols to strings of symbols. Let a *substitution* be a mapping from variables to variables or objects. Let θ, σ range over substitutions and replacements. A *ground substitution* is a substitution from variables to objects. Let I, J, K range over ground substitutions. The application of a substitution θ to some argument a is denoted $\theta(a)$ or $a\theta$. When using postfix notation, the composition $\theta_2 \circ \theta_1$ is denoted $\theta_1\theta_2$.

Definition 2.1 A *Transition System* TS is a pair (I, Γ) where

I is a *finite* state intended to represent the initial state of the system.
Γ is a finite set of *transitions*. Each transition $\tau \in \Gamma$ is a pair of formulas, $\tau : (\varphi, \psi)$, where φ is an object free conjunction of literals, and ψ is an object free conjunction of simple literals and negative existential literals. The formula φ is called the *precondition* and the formula ψ is called the *postcondition* of τ. □

The postcondition ψ can be regarded as an assignment that adds (or deletes) objects and object pairs to the relations in the system.

The restrictions on the forms of the preconditions and postconditions are motivated by simplicity reasons and by pragmatic reasons originating in our special interest in telephone system specifications. In Section 4 we give an example of a system that models a basic telephone service.

2.2 Semantics of Transition Systems

In this section, we define the semantics for this special class of transition systems. Some definitions needed in this section are standard from first-order logic, and are therefore omitted unless needed for clarity (see [5] or some other introduction to first-order logic).

We define a satisfiability relation $s \models \varphi$ between states s and formulas φ as follows:

– $s \models \varphi$ iff $\varphi \in s$, when φ is a ground atom ($\varphi = a(d)$ or $\varphi = b(d_1, d_2)$)

- $s \models \varphi$ is defined in the standard way when φ is a non-atomic closed formula. E.g., $\exists x : \varphi(x)$ means that $\varphi(d)$ is true for some $d \in D$.

A closed formula φ is *satisfiable* if there exists a state s such that $s \models \varphi$. A closed formula φ is said to be *valid*, denoted $\models \varphi$, if $s \models \varphi$ for all states s.

Definition 2.2 Let TS be a graph system (I, Γ) and let $\tau : (\varphi, \psi) \in \Gamma$. We define a transition relation $\xrightarrow{\tau}$ on graphs by $(s, s') \in \xrightarrow{\tau}$ (denoted $s \xrightarrow{\tau} s'$) iff there exists a ground substitution K such that

1. $s \models \varphi K$
2. ψK is satisfiable
3. $s' = s \cup s_1 \setminus s_2$ where

$$\begin{aligned} s_1 &= \{\, l \mid l \in \psi K \text{ and } l \text{ is a positive simple literal} \,\}, \text{ and} \\ s_2 &= \quad \{\, l \mid \neg l \in \psi K \text{ and } \neg l \text{ is a negative simple literal} \,\} \\ &\quad \cup \{\, l \mid l = b(d_1, d_2) \text{ and } \neg \overleftarrow{b}\,(d_1) \in \psi K \,\} \\ &\quad \cup \{\, l \mid l = b(d_1, d_2) \text{ and } \neg \overrightarrow{b}\,(d_2) \in \psi K \,\} \end{aligned}$$ □

Condition 1 ensures that the precondition is satisfied. Condition 2 ensures that the postcondition imposes a consistent assignment to the relations. Condition 3 performs the assignment. The new state s' is obtained through a "minimal" change to the old state s such that ψ holds in s'.

A transition τ is said to be *enabled* on s if there is some s' such that $s \xrightarrow{\tau} s'$. Otherwise the transition is said to be *disabled* on s.

We define $\longrightarrow$ as $\bigcup_{\tau \in \Gamma} \xrightarrow{\tau}$. Let $\xrightarrow{*}$ be the reflexive, transitive closure of $\longrightarrow$. A graph s' is said to be *reachable* from a graph s if $s \xrightarrow{*} s'$ and is said to be *reachable* if s' is reachable from the initial graph I.

3 Reachability Analysis

In this section we look at the *reachability problem* for the transition systems we are using:

> Given a transition system TS and a class C of states, is any state in C reachable in TS?

We will specifically look at the case where the set C is defined by a certain type of formulas, called *patterns*, which we will define next. For a set Y of variables, let *disjoint*(Y) be the formula which states that all variables in Y are different. For instance, *disjoint*$(\{\, x, y, z \,\})$ is the formula $x \neq y \wedge x \neq z \wedge y \neq z$. A *pattern* is a closed object free formula of the form $\exists Y :$ *disjoint*$(Y) \wedge \varphi(Y)$ where $\varphi(Y)$ is a conjunction of literals. In the following, we will write a formula of form $\exists Y :$ *disjoint*$(Y) \wedge \varphi(Y)$ as $\exists_{disj} Y : \varphi(Y)$. We use p, q to range over patterns.
Note : In this paper, we restrict patterns to existentially quantified conjunctions of literals, but our results can be generalized to reachability analysis for sets C defined by arbitrary formulas by transforming φ into Disjunctive Normal

Form and check for reachability separately for each disjunct in the formula. The disjointness requirement can be removed in a similar way.

The Halting Problem can easily be encoded as a reachability problem for transition systems, and this shows that the reachability problem is undecidable in general. In spite of this, we present a method for analyzing reachability properties, which we believe to be useful in many telephone applications. To check whether a certain pattern p is reachable, we apply a classical method: compute the set of patterns from which p is reachable, and see whether the initial state is in this set. The set of states from which p is reachable is computed as a fixpoint. In this computation, we use disjunctions of patterns to represent sets of states. The fixpoint computation need not always terminate, but if it does our method will give an answer to the question whether p is reachable. This approach to verification is, in principle, similar to e.g., the one used for lossy channel systems in [1], where however the fixpoint computation always terminates.

The sets of states that we want to check for reachability are, in the setting of telephone systems, usually finite erroneous configurations. These configurations are local phenomena, and the choice of a backward computation therefore seems reasonable. A forward search would start from a global state and would more or less build the entire state space.

In the following subsections, we will first in Section 3.1 present the general structure of our verification method. This method is built from three nontrivial operations, discussed in Sections 3.2 and 3.3.

3.1 Outline of the Reachability Analysis Procedure

As already stated, the idea of our reachability analysis is to compute the set of states from which a certain pattern p can be reached. This can be reformulated as computing the set of states that can be reached by performing transitions "backwards" from some state in p. Before presenting the analysis procedure, we introduce some notation for representing such sets of states.

Definition 3.1 For a transition τ, pattern p and state s, define a "possibility" predicate $\langle\tau\rangle p$ on states as

$$s \models \langle\tau\rangle p \quad \text{iff} \quad \exists s' : s \xrightarrow{\tau} s' \wedge s' \models p \qquad \square$$

The notation is inspired by the modal possibility operator in Dynamic Logic (see e.g., [12]).

For a set Γ of transitions define $\langle\Gamma\rangle p$ as $\bigvee_{\tau\in\Gamma} \langle\tau\rangle p$ and define $\langle\Gamma\rangle^i p$ for $i \geq 0$ as $\langle\Gamma\rangle^0 p = p$ and $\langle\Gamma\rangle^{i+1} p = \langle\Gamma\rangle(\langle\Gamma\rangle^i p)$ for $i \geq 0$. Finally, define $\langle\Gamma\rangle^* p$ as $\bigvee_{i\geq 0} \langle\Gamma\rangle^i p$. We see that p is reachable in a transition system (I, Γ) if and only if $I \models \langle\Gamma\rangle^* p$.

In Section 3.2 we will show how to compute $\langle\tau\rangle p$. Before that, and assuming that we can compute $\langle\tau\rangle p$, we present our procedure to analyze whether $I \models \langle\Gamma\rangle^* p$ holds for an initial state I and a pattern p.

```
**Procedure**
**Input:** A transition system $TS = (I, \Gamma)$ and a pattern $p$.
**Output:** does $I \models \langle\Gamma\rangle^* p$ hold in $TS$?
var $A, W$ : sets of patterns

$A := \emptyset$
$W = \{ p \}$
while $W \neq \emptyset$ do
    choose $q \in W$
    if $(I \models q)$ then return $true$ and exit
    $W := W \setminus \{ q \}$
    $A := A \setminus \{ q' \in A : q' \succeq q \} \cup \{ q \}$
    $W := W \cup \{ q' \in \langle\Gamma\rangle q : (\neg\exists q'' \in (A \cup W) : q' \succeq q'') \}$
end while
return $false$
```

Fig. 1. Procedure for analyzing reachability

The idea of our method is to calculate $\langle\Gamma\rangle^* p$, represented as the disjunction of a set of patterns, by backward reachability analysis. At each iteration the procedure checks whether I satisfies some pattern in the current set. The method uses as data structures two sets of patterns, A and W, such that $A \cup W$ represents the set of states that have so far been found to be "backward reachable" from the pattern p. The set A contains the patterns from which backward transitions have already been generated, and W contains the pattern from which backward transitions must still be generated and analyzed. If, during the computation, we have computed a pattern p_1 and can find a pattern $p_2 \in (A \cup W)$ that is entailed by p_1 (denoted $p_1 \succeq p_2$), then p_1 is "redundant", and can safely be discarded. Entailment here is required to be a relation as strong as or stronger than implication: $\models p_1 \Longrightarrow p_2$. A pseudocode description of the analysis method is given in Figure 1.

The procedure relies on the ability to perform three nontrivial operations on patterns p and q: calculating $\langle\Gamma\rangle p$, deciding $p_1 \succeq p_2$, and deciding whether $I \models q$ for a state I.

3.2 Computing Backward Transitions

In this section, we describe how to compute $\langle\Gamma\rangle p$. Let $\psi(X)$ be a conjunction of simple literals and negative existential literals. We can easily see that the formula $\exists X : \psi(X)$ is satisfiable if and only if

1. no positive literal occurs as a negative literal, and
2. if $b(x, y) \in \psi$ then no literal of the form $\neg \overrightarrow{b}\ (y)$ or of the form $\neg \overleftarrow{b}\ (x)$ occurs in ψ.

Definition 3.2 For a conjunction $\psi(X)$ of simple literals and negative existential literals such that $\exists X : \psi(X)$ is satisfiable, define the textual replacement θ_ψ on patterns by defining its effect on a pattern p as follows.

1. if $a(x)$ (or $b(x,y)$) is a positive literal in ψ, then replace all occurrences of $a(x)$ (of $b(x,y)$) in p by *true*,
2. if $\neg a(x)$ (or $\neg b(x,y)$) is a negative literal in ψ, then replace all occurrences of $a(x)$ (of $b(x,y)$) by *false*,
3. if $\neg \overrightarrow{b}\,(x)$ (or $\neg \overleftarrow{b}\,(x)$) is a negative existential literal in ψ, then for all variables y, replace all occurrences of $b(y,x)$ and $\overrightarrow{b}\,(x)$ (of $b(x,y)$ and $\overleftarrow{b}\,(x)$) by *false*,
4. if $b(x,y)$ is a positive literal in ψ, then replace all occurrences of $b(x,y)$, $\overrightarrow{b}\,(y)$ and $\overleftarrow{b}\,(x)$ by *true*. □

We compute $\langle\Gamma\rangle p$ as the formula $\bigvee_{\tau\in\Gamma}\langle\tau\rangle p$. The following theorem states how to compute $\langle\tau\rangle p$ for each transition $\tau \in \Gamma$.

Theorem 3.1 For any state s, transition $\tau : (\varphi(X), \psi(X))$ and pattern $p = \exists_{disj} Z : \rho(Z)$, where X and Z are assumed disjoint, we have that $s \models \langle\tau\rangle p$ if and only if there is a substitution σ which maps the variables in X to variables in $Z \cup X$ such that

1. $\psi\sigma$ is satisfiable, and
2. $s \models \exists_{disj}(Z \cup X\sigma) : \varphi\sigma \wedge \rho\theta_{\psi\sigma}$ □

Since we only have a finite number of possible σ, we can then calculate the pattern $\langle\tau\rangle p$ as $\bigvee_\sigma \exists_{disj}(Z \cup X\sigma) : \varphi\sigma \wedge \rho(Z)\theta_{\psi\sigma}$ for all substitutions σ restricted as before.

3.3 Checking Entailment and Model Checking of States

The computational complexity of deciding whether $\models p_1 \Longrightarrow p_2$ turns out to be NP-complete, and to speed up the computation one can use a stronger entailment relation $p_1 \succeq p_2$. However, the risk of nontermination then becomes greater.

In the current implementation we use implication as the entailment relation, where implication is computed as follows: $\models p_1 \Longrightarrow p_2$ holds for patterns $p_1 = \exists_{disj} X : \rho_1(X)$ and $p_2 = \exists_{disj} Y : \rho_2(Y)$ iff p_1 is satisfiable and there exists an injective ground substitution K_2 with domain X such that

$$\rho_2 K_2 \subseteq \rho_1 K_1 \cup \{\ \overleftarrow{b}\,(d_1), \overrightarrow{b}\,(d_2) : b(d_1,d_2) \in \rho_1 K_1\ \}$$

where K_1 is some injective ground substitution with domain Y.

In the procedure we need to compute $s \models p$ for a finite state s and a pattern $p = \exists_{disj} X : \rho(X)$. This is done as follows: $I \models p$ iff there exists an injective ground substitution K from variables in X to objects in I such that all positive literals in ρK are in I' and no negative literal in ρK occurs in positive form in I', where $I' = I \cup \{\ \overleftarrow{b}\,(d_1), \overrightarrow{b}\,(d_2) : b(d_1,d_2) \in I\ \}$.

4 Application: A Basic Telephone Service

In this section we describe a basic telephone service (sometimes called Plain Old Telephone Service, POTS) as a transition system, and give some examples of properties that have been automatically verified for this system.

Each subscriber is modeled by an object. Internal state information for each subscriber is modeled by the unary relation *active* and call information is modeled by the binary relations $trying(d_1, d_2)$, $busy(d_1, d_2)$, and $conn(d_1, d_2)$. These relations were described earlier in Example 2.1.

4.1 The transitions for POTS

The following transitions model basic calling between subscribers. Each transition models the system response to an action from a subscriber. For convenience, we define the macro

$$idle(x) \triangleq \neg\overleftarrow{trying}(x) \land \neg\overleftarrow{busy}(x) \land \neg\overrightarrow{conn}(x) \land \neg\overleftarrow{conn}(x) \ .$$

The transitions are:

$\tau = (\varphi, \psi)$	φ	ψ
$OnHook(x)$	$active(x)$	$\neg active(x)$ $\land\ idle(x)$
$Answer(x, y)$	$trying(x, y)$ $\land\ \neg active(y)$	$conn(x, y)$ $\land\ \neg trying(x, y)$ $\land\ active(y)$
$OffHook(x)$	$\neg active(x)$ $\land\ \neg\overrightarrow{trying}(x)$	$active(x)$
$Dialling(x, y)$	$active(x)$ $\land\ \neg active(y)$ $\land\ idle(x)$ $\land\ \neg\overrightarrow{trying}(y)$	$trying(x, y)$
$DialBusy_1(x, y)$	$active(x)$ $\land\ \neg active(y)$ $\land\ idle(x)$ $\land\ \overrightarrow{trying}(y)$	$busy(x, y)$
$DialBusy_2(x, y)$	$active(x)$ $\land\ active(y)$ $\land\ idle(x)$	$busy(x, y)$

The transition *OnHook*(x) reflects that subscriber x puts down the receiver. Any connections to/from x are released and x is no longer trying to call someone. The transition *Answer*(x, y) reflects that subscriber x is calling y, and y answers the call by lifting his receiver. Subscriber y must have is phone on the hook for this to happen. The transition *OffHook*(x) reflects that subscriber x lifts his receiver, and there is no one trying to call him. The transition *Dialling*(x, y)

reflects that subscriber x dials the number to y, and no one else is calling y. The transitions $DialBusy_1(x,y)$ and $DialBusy_2(x,y)$ both reflect that subscriber x dials the number to y and that y is busy, either because someone is trying to call y or because y has the phone off the hook.

4.2 Verifying Properties of the Telephone Service

In this section, we give examples of properties that have been verified by an implementation of the reachability procedure written in Prolog. At present, the implementation exists in an early stage only doing very simple optimizations. We show some properties for the model of POTS given earlier.

Initially we shall look at the transition system $TS = (\emptyset, POTS)$, with an empty initial state and the transitions are the transitions stated for POTS in section 4.1.

We want to check the property that there can be at most one connection to a subscriber. This requires three patterns, and one of them is

$$p = \exists_{disj} x, y, z : conn(x,y) \wedge conn(z,y)$$

which says that there are subscribers x and z such that both subscribers are connected to the subscriber y. The property holds iff $\emptyset \not\models \langle POTS\rangle^* p$.

From the computation of $\langle POTS\rangle^* p$ we obtain the pattern $q = \exists_{disj} x, y, z : \rho_1 \vee \rho_2 \vee \rho_3 \vee \rho_4 \vee \rho_5$, where

$$\begin{aligned}
\rho_1 &= conn(x,y) \wedge conn(z,y)\\
\rho_2 &= trying(x,y) \wedge \neg active(y) \wedge conn(z,y)\\
\rho_3 &= active(x) \wedge idle(x) \wedge \neg \overrightarrow{trying}\,(y) \wedge conn(z,y) \wedge \neg active(y)\\
\rho_4 &= active(x) \wedge \neg \overrightarrow{trying}\,(x) \wedge \neg active(y) \wedge conn(z,y) \wedge \neg \overrightarrow{trying}\,(y)\\
\rho_5 &= \neg active(x) \wedge \neg \overrightarrow{trying}\,(x) \wedge \neg active(y) \wedge conn(z,y) \wedge \neg \overrightarrow{trying}\,(y)
\end{aligned}$$

Now, $\emptyset \not\models q$ so we are done. This computation was run on a SPARC ELC work station, and took approximately 1 minute to compute running the Prolog program in interpreted mode. In total, 102 intermediate patterns were generated, but $\rho_1, \ldots, \rho_5$ were the only patterns not discarded because of entailment.

Other properties that have been successfully checked are:

	Pattern	Result	Time	# interm. patterns
1	$conn(x,y) \wedge conn(z,y)$	Not reachable	1 min	102
2	$conn(x,y) \wedge conn(y,z)$	Not reachable	2 1/2 min	278
3	$conn(y,x) \wedge conn(y,z)$	Not reachable	1 1/2 min	157
4	$conn(x,x)$	Not reachable	5 sec	10
5	$busy(x,y) \wedge \neg active(y)$	Reachable	45 sec	81

Pattern 1–3 say that a subscriber can have two (or more) connections. None of these should be reachable. Pattern 4 says that subscriber x can connect to himself, which should not be reachable. Pattern 5 says that subscriber x can be calling subscriber y which has not picked up the receiver and still hear a busy tone in the phone. This state should be reachable.

4.3 Example: Mutual Exclusion in a Graph

The following is a mutual exclusion algorithm for processes communicating through an arbitrary connection graph. Each object d represents a process. The atom $cs(d)$ represent that process d executes a critical section, and the absence of the atom $cs(d)$ represents the execution of a non-critical section. Mutual exclusion is preserved by passing around a unique token. A ground atom $token(d)$ indicates that process d has the token. The transitions are:

$\tau = (\varphi, \psi)$	φ	ψ
$enter(x)$	$token(x)$	$cs(x)$
$leave(x)$	$cs(x)$	$\neg cs(x)$
$pass(x, y)$	$\neg cs(x) \wedge token(x)$	$\neg token(x) \wedge token(y)$

The property $\exists_{disj} x, y : cs(x) \wedge cs(y)$ was found not to be reachable in 6 seconds, with a total of 16 intermediate patterns computed.

5 Conclusion

We have considered the problem of verifying reachability properties of a class of infinite-state distributed algorithms that is well suited for the specification of telephone services. We have presented a method for analyzing reachability properties, and show that it can be successfully applied to practical examples.

Future work in general includes trying to characterize classes of systems for which the method would terminate, studying how the choice of entailment relation changes termination speed, and applying the method to larger examples to see how well it scales up.

For telephone systems specifications, larger case studies of detecting interactions should be made to find out how useful the procedure is for our purposes.

References

1. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc.* 8th *IEEE Int. Symp. on Logic in Computer Science*, 1993. Accepted for Publication in Information and Computation.
2. P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. In Abiteboul and Shamir, editors, *Proc. ICALP '94*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327. Springer Verlag, 1994.
3. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc.* 5th *IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, Philadelphia, 1990.
4. R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proc.* 14th *IEEE Real-Time Systems Symposium*, pages 2–11, 1993.
5. M. Ben-Ari. *Mathematical Logic for Computer Science*. Prentice Hall, 1993.

6. J. Blom, B. Jonsson, and L. Kempe. Using temporal logic for modular specification of telephone services. In *Feature Interactions in Telecommunications Systems*, Amsterdam, Holland, May 1994.
7. T. Bowen, F. Dworack, C. Chow, N. Griffeth, G. Herman, and Y.-J. Lin. The feature interaction problem in telecommunications system. *SETS*, 1989.
8. O. Burkart and B. Steffen. Model checking for context-free processes. In Cleaveland, editor, *Proc. CONCUR '92, Theories of Concurrency: Unification and Extension*, number 630 in Lecture Notes in Computer Science, pages 123–137. Springer Verlag, 1992.
9. 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.
10. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In W. R. Cleaveland, editor, *Proc. CONCUR '92, Theories of Concurrency: Unification and Extension*, pages 138–147, 1992.
11. E. M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing, Vancouver, Canada*, pages 294–303, 1987.
12. M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.
13. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
14. P. Jančar. Decidability of a temporal logic problem for petri nets. *Theoretical Computer Science*, 74:71–93, 1990.
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. F. J. Lin and Y.-J. Lin. A building block approach to detecting and resolving feature interactions. In L. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommuniactions Systems*, pages 86–119. IOS Press, 1994.
17. Z. Shtadler and O. Grumberg. Network grammars, communication behaviours and automatic verification. In Sifakis, editor, *Proc. Workshop on Computer Aided Verification*, volume 407 of *Lecture Notes in Computer Science*, pages 151–165. Springer Verlag, 1990.
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. P. Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–193, Jan. 1986.