Channel Representations in Protocol Verification (Preliminary Version)

Parosh Aziz Abdulla and Bengt Jonsson

Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden
{parosh,bengt}@docs.uu.se

Abstract. In automated verification of protocols, one source of complications is that channels may have unbounded capacity, in which case a naive model of the protocol is no longer finite state. Symbolic techniques have therefore been developed for representing the contents of unbounded channels. In this paper, we survey some of these techniques and apply them to a simple leader election protocol. We consider protocols with entities modeled as finite state machines which communicate by sending messages from a finite alphabet over unbounded channels; this is a framework for which many techniques have been developed. We also consider a more general model in which messages may belong to an unbounded domain of values which may be compared according to a total ordering relation: the motivation is to study protocols with timestamps or priorities. We show how techniques from the previous setting can be extended to this more general model, but also show that reachability quickly becomes undecidable if channels preserve the ordering between messages.

1 Introduction

Protocol verification has, since 25 years, been a driving application for the development of automated verification techniques. State space exploration techniques were developed in this context [36, 33], and it is one of the important application areas for current model checking tools such as SPIN [25] and UPPAAL [27].

Protocol verification involves the construction of a model of a protocol, which can be subject to analysis, e.g., by a model checking tool. The model should abstract from less relevant details of the protocol in order to facilitate the analysis. Typically, a protocol model consists of a number of processes, which communicate over channels of some kind. In many cases, the channels are a significant source of problems for the analysis. If communication channels are large or unbounded, naive model-checking cannot be performed exhaustively. When modeling a protocol, one must therefore be careful to model the channels in a way which suits subsequent analysis. Many model checkers do not support unbounded channels.

In general, unbounded channels have infinitely many states, and must therefore be represented symbolically in automated verification. Different representations have been proposed for different types of channels. In this paper, we will survey some of these symbolic techniques and illustrate them on a simple example.

First, we consider a model of protocols with entities modeled as finite state machines which communicate by sending messages from a finite alphabet over unbounded channels. If channels are unordered, this model can be represented by Petri Nets. FIFO ordered channels has been considered rather extensively in protocol verification [8, 23, 15, 24, 30, 31, 34]. Since this model can simulate Turing machines [12], we will devote most attention to a weaker model in which the FIFO channels can spontaneously lose messages at any time. We will illustrate two types of symbolic representations and their use in checking whether some states are reachable. The first type represents *upward closed* sets. It is useful in backward reachability analysis, since it gives a lower bound on which messages must be in each channel for a certain set of states to be reachable. This representation was used in [5] to decide the reachability problem, and in [5, 21] to decide the termination problem. The second type (called Simple Regular Expressions in [3]), dually represents *downward closed* sets. This representation is useful in forward reachability analysis, since it gives an upper bound on which messages *may* be in the channel (we can never be sure whether a message is in the channel, since it can be lost). Analogous symbolic representations of upward closed sets have also been used for unordered channels and related models like Petri nets and broadcast systems [20, 4, 22, 19, 17].

We also consider how the techniques illustrated in the first part may be extended to models with an infinite set of message values, on which a limited set operations can be performed. We focus on the case where the domain of message values is equipped with a total ordering, which may be used in guards. The motivation is to study protocols with timestamps or priorities. We present a negative result showing that with lossy FIFO channels one can simulate perfect FIFO channels already if tests for equality and inequality are allowed on messages. For unordered channels, however, the backward reachability analysis presented in the first part of the paper can be used as a decision procedure. Using ideas from our earlier work [6, 7] we can show that the techniques carry over to handle messages on which an ordering relation is defined.

This small survey is organized as follows. In the next section, we define protocols with a finite message alphabet and present a simple leader election protocol as a running example. Forward and backward reachability analysis with associated symbolic representations are presented in Section 3. In Section 4, we extend the model to an infinite set of messages, and present an undecidability result for lossy FIFO channels, and an extension to messages with an ordering relation. Section 6 contains discussion and conclusion.

2 Protocols with a Finite Set of Messages

In this section, we present our first protocol model: finite-state processes which communicate over unbounded channels using a finite message alphabet. In this model, a program has two parts: a control part and a channel part. The control part represents the combined behavior of the processes that perform local computations and communicate over the channels. It is represented by a finite automaton, which is typically the cross-product of control parts of the finite-state processes in the system. To each transition there may be associated the transmission or reception of a message to or from a channel. The channel part consists of a set of channels, each of which contains a potentially unbounded number of messages from a finite alphabet. Each channel can be ordered or unordered, lossy or non-lossy.

Notation. For a set $\mathcal{M}$ we use $\mathcal{N}^{\mathcal{M}}$ to denote the set of multisets (bags) of elements in $\mathcal{M}$, i.e., the set of mappings from $\mathcal{M}$ to the natural numbers $\mathcal{N}$. The empty bag is denoted by $\emptyset$. For $x, x' \in \mathcal{N}^{\mathcal{M}}$ we let $x \bullet x'$ denote the multiset union of x and x'. Define the partial order $\preceq$ on $\mathcal{N}^{\mathcal{M}}$ by $x \preceq x'$ if x is a subbag of x', i.e., $x(m) \leq x'(m)$ for all $m \in \mathcal{M}$.

For a set $\mathcal{M}$ we use $\mathcal{M}^*$ to denote the set of finite strings of elements in $\mathcal{M}$. The empty string is denoted by ε. For $x, x' \in \mathcal{M}^*$ we let $x \bullet x'$ denote the concatenation of x and x'. Define the partial order $\preceq$ on $\mathcal{M}^*$ $x \preceq x'$ if and only if x is a (not necessarily contiguous) substring of x'.

Note that we have overloaded the notations $\bullet$ and $\preceq$ for both bags and strings. We allow any element $m \in \mathcal{M}$ to be interpreted as the bag or string with the only element m, and thus $x \bullet m$ denotes the addition of element m to the bag or string x. Note also that the relation $\preceq$ closely corresponds to the notion of losing messages: $x \preceq x'$ if x can be obtained by deleting (losing) elements from the bag or string x'.

Programs. A *program* consists of

- A *channel part*, given by a finite set C of *channels*, and a finite set $\mathcal{M}$ of *messages*. Each channel is either ordered or unordered, either lossy or nonlossy.
- A *control part*, defined by a finite automaton $\langle S, s_0, \delta \rangle$, where S is a finite set of *control states*, $s_0 \in S$ is the *initial control state*, and δ is a finite set of *transitions*. Each transition is a triple of the form $\langle s, op, s' \rangle$, where $s, s' \in S$, and *op* is either the empty label ϵ or an operation of form $c!m$ or $c?m$, where $c \in C$ and $m \in \mathcal{M}$, which denotes the transmission of message m to channel c, or reception of message m from channel c, respectively.

Semantics. A *global state* γ of a program is a pair $\langle s, w \rangle$, where $s \in S$, and w is a mapping from C to bags and strings over $\mathcal{M}$ (depending on whether each particular channel is ordered or unordered). The *initial global state* γ_0 is $\langle s_0, w_0 \rangle$, where w_0 maps each channel to the empty bag or string. The global state can be changed by performing transitions in δ. More precisely, the global state of a nonlossy system can change from γ to γ', denoted $\gamma \longrightarrow \gamma'$, as follows.

- If $\langle s, \epsilon, s' \rangle \in \delta$ then $\langle s, w \rangle \longrightarrow \langle s', w \rangle$ for any w.
- If $\langle s, c!m, s' \rangle \in \delta$ then $\langle s, w \rangle \longrightarrow \langle s', w[c := w(c) \bullet m] \rangle$ for any w.
- If $\langle s, c?m, s' \rangle \in \delta$ then $\langle s, w[c := m \bullet w(c)] \rangle \longrightarrow \langle s', w \rangle$ for any w.

Here $w[c := w(c) \bullet m]$ denotes the mapping which maps c to $w(c) \bullet m$ and any $c' \neq c$ to $w(c')$. Extend $\preceq$ in the natural way to global states of form $\langle s, w \rangle$ by $\langle s, w \rangle \preceq \langle s', w' \rangle$ is $s = s'$ and $w(c) \preceq w'(c)$ for $c \in C$. The global state of a lossy system can change from γ to γ', denoted $\gamma \longrightarrow \gamma'$ if the corresponding nonlossy system can perform $\gamma'' \longrightarrow \gamma'''$ for γ'', γ''' with $\gamma'' \preceq \gamma$ and $\gamma' \preceq \gamma'''$.

A set Γ of global states is *upward closed* (UC) if $\gamma \in \Gamma$ and $\gamma \preceq \gamma'$ imply $\gamma' \in \Gamma$. We define the notion of *downward closed* (DC) analogously.

The behavior of a program depends on whether the channels are ordered or unordered and on whether they are lossy or nonlossy. In the following, we will assume that all channels of a program are of the same kind, and use the terms OL (ordered lossy), ON (ordered nonlossy), UL (unordered lossy), and UN (unordered nonlossy) for the four program models.

Reachability. A global state γ' is said to be *reachable* from a global state γ if $\gamma \xrightarrow{*} \gamma'$. A global state γ is said to be *reachable* if γ is reachable from the initial global state γ_0. For a program and a set Γ of global states, the *reachability problem* asks whether some state in Γ is reachable. Typically, the set Γ are undesirable states, which should not occur when the system executes. In the following, we will mostly consider the special problem of *control state reachability* for a set $T \subseteq S$ of control states, asking whether the set $\{\langle s, w \rangle : s \in T\}$ of global states is reachable.

The reachability problem is related to checking of *safety properties*. A safety property can be described by specifying a set of finite sequences of states or transitions that are allowed to occur when the system executes. If the set of allowed sequences is regular, then there is a standard procedure for transforming the problem of checking a safety property into the problem of checking for reachability [35].

Example As a running example, we use a simplification and variation of a leader election protocol, due to LeLann [28]. Assume a set of process connected into a ring by channels. Each process has a unique identity. In the algorithm, each process starts the algorithm by transmitting its identity over its outgoing channel to the next neighbor. Thereafter, the process waits to receive messages on its incoming channel. Each arriving identity is forwarded to the outgoing channel, except when the received identity is that of the process itself: in this case the process terminates the algorithm without forwarding its identity again. Each process elects as leader the process with the least identity among the ones received (including its own identity) during the algorithm. Thus, a process need not store all received identities during the algorithm, but only maintain the minimum identity among its own and the ones received so far. A correctness property of the algorithm is that all processes should elect the same leader.

In order to illustrate the algorithm for all four channel models with unbounded channels, we will vary the algorithm: each process will initially transmit its identity an unbounded number of times instead of just once, and channels may belong to any of the four channel models. With this modification, the processes should report the same leader if *all* processes terminate. Note that if channels are lossy, then some process may not terminate, in which case it is possible for two other terminating processes to disagree on the identity of the leader.

In this section, we specialize the protocol to two finite-state processes, with identities 1 and 2. They communicate over the channel c_1, which transfers messages from process 2 to process 1, and channel c_2, which transfers messages from process 1 to process 2. The set of messages $\mathcal{M}$ is simply the set of identities $\{1, 2\}$. Finite automata for the two processes are shown in Figure 1. In the figure, we have slightly extended the notation and let a transition labeled by $c?m/c!m$ denote two sequentially ordered transitions, the first of which is labeled $c?m$ and the second labeled $c!m$. The states of processes are labeled I (initial), W (waiting

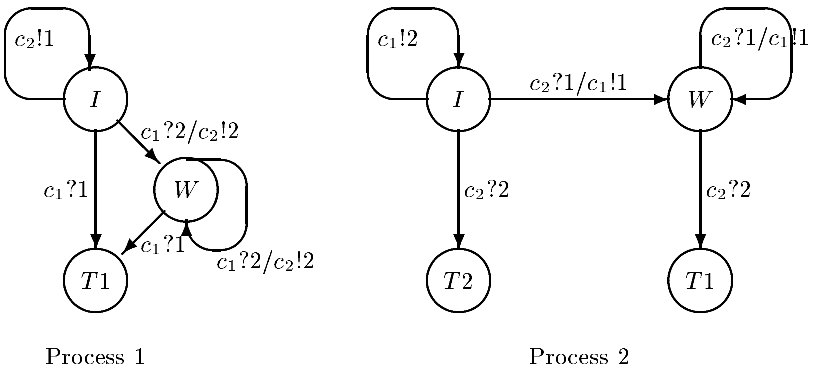

Fig. 1. Leader Election Protocol for Two Processes

for messages), and Ti for $i = 1, 2$ (terminated with i as elected leader). Initially, both processes are in state I, and the channels are empty. Process 1 initially transmits its identity an unbounded number of times. When receiving message 2, it compares that message with its current identity, and concludes that identity 1 is still the leading candidate for becoming a leader. Consequently, message 2 is simply forwarded to the outgoing channel. When receiving its own identity 1, the process terminates and announces that process 1 is the leader. Process 2 initially transmits its identity an unbounded number of times. When receiving message 1, it changes its internal state to reflect that 1 is the minimal identifier. Each received 1 is forwarded to the outgoing channel. When receiving message 2, the process terminates. In state $T2$ it announces 2 as the leader, and in state $T1$ it announces 1 as the leader.

According to our model, we represent control states as pairs of control states of each process. The above correctness property is expressed by requiring that the control state $\langle T1, T2 \rangle$ is not reachable. This state represents an incorrect termination of the algorithm in which each process elects itself as leader.

3 Reachability Analysis

In this section, we illustrate existing techniques for checking reachability, applied to the system in Figure 1. In Section 3.1, we illustrate backward reachability analysis for the OL model [5] from our earlier work, and thereafter for the UL and UN models [4, 22]. In Section 3.2, we illustrate forward reachability analysis for the OL model according to [3]; there are also other approaches [21]. We will also illustrate the approach of using standard finite automata (under the name QDDs) for the more general ON model [9, 10]. For the UN and UL models, forward reachability analysis has a large literature (e.g., [26, 20, 18, 38]).

3.1 Backward Symbolic Reachability Analysis

We first consider symbolic backward reachability analysis. Like standard symbolic model checking [32, 16, 13], it is based on symbolic calculation of pre-images, but with a specific representation of sets of states. The algorithm for checking reachability of a set Γ of global states consists in calculating the set of global states from which a state in Γ is reachable. For a set Γ of global states, let let $pre(\Gamma)$ denote the set $\{\gamma \;:\; \exists \gamma' \in \Gamma \,.\, \gamma \longrightarrow \gamma'\}$ of states from which a state in Γ can be reached by performing a transition. The naive version of the backward reachability algorithm for checking control state reachability of a set T of control states consists in generating a sequence $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$ of sets of global states, where $\Gamma_0 = \{\langle s, w\rangle \;:\; s \in T\}$ and $\Gamma_{i+1} = \Gamma_i \cup pre(\Gamma_i)$, which stops when $\Gamma_{i+1} = \Gamma_i$. Then T is reachable if and only if $\gamma_0 \in \Gamma_i$.

In the UL, UN, and OL models, it is useful to represent sets of states as finite unions of atomic constraints. An *atomic constraint* is given by a global state $\langle s, w\rangle$ and denotes the UC set $[\![\langle s, w\rangle]\!] = \{\langle s, w'\rangle \;:\; w \preceq w'\}$. Programs in the UL, UN, and OL models now satisfy the following two important properties:

- Any program is *monotonic* with respect to $\preceq$, i.e., whenever $\gamma \longrightarrow \gamma'$ and $\gamma \preceq \gamma_1$ there is a γ_1' such that $\gamma_1 \longrightarrow \gamma_1'$ and $\gamma' \preceq \gamma_1'$. In other words, $\preceq$ is a simulation relation on global states.
- $\preceq$ is a *well quasi-ordering* (wqo) on the set of global states, i.e., in each infinite sequence $\gamma_0\ \gamma_1\ \gamma_2\ \gamma_3 \cdots$ of global states, there are indices $i < j$ such that $\gamma_i \preceq \gamma_j$.

Monotonicity implies that $pre(\Gamma)$ is UC for any UC set Γ. Well quasi-orderedness of $\preceq$ implies that any UC set is a *finite* union of atomic constraints, and that any increasing sequence $\Gamma_0 \subseteq \Gamma_1 \subseteq \Gamma_2 \subseteq \cdots$ of UC sets eventually converges so that there is an i with $\Gamma_j = \Gamma_i$ whenever $i \leq j$. Together, this means that the iterative calculation of $pre^*(\Gamma_0)$ can be carried out as above, using finite sets of atomic constraints as symbolic representation, and that it is guaranteed to terminate after a finite number of iterations [4, 22].

Example Let us use this technique to check whether the erroneous terminal state is reachable in the OL model. We represent global states as 4-tuples

$\langle s_1, s_2, w_1, w_2 \rangle$ where s_i is the control state of process i, and w_i is the contents of channel c_i. We want to investigate whether the atomic constraint generated by $\langle T1, T2, \varepsilon, \varepsilon \rangle$ is reachable. Let $\Gamma_0 = [\![\langle T1, T2, \varepsilon, \varepsilon \rangle]\!]$. Then

$$\begin{aligned}
\Gamma_1 &= \Gamma_0 \cup [\![\langle I, T2, 1, \varepsilon \rangle]\!] \cup [\![\langle W, T2, 1, \varepsilon \rangle]\!] \cup [\![\langle T1, I, \varepsilon, 2 \rangle]\!] \\
\Gamma_2 &= \Gamma_1 \cup [\![\langle I, I, 1, 2 \rangle]\!] \cup [\![\langle I, T2, 21, \varepsilon \rangle]\!] \cup [\![\langle W, T2, 21, \varepsilon \rangle]\!] \cup [\![\langle W, I, 1, 2 \rangle]\!] \\
\Gamma_3 &= \Gamma_2 \cup [\![\langle W, I, 21, \varepsilon \rangle]\!] \cup [\![\langle I, I, 21, \varepsilon \rangle]\!]
\end{aligned}$$

Here the procedure converges, without containing the initial state $\langle I, I, \varepsilon, \varepsilon \rangle$. Note that in set Γ_2, the sets $[\![\langle I, T2, 21, \varepsilon \rangle]\!]$ and $[\![\langle W, T2, 21, \varepsilon \rangle]\!]$ are contained in the the already generated sets $[\![\langle I, T2, 1, \varepsilon \rangle]\!]$ and $[\![\langle W, T2, 1, \varepsilon \rangle]\!]$. They are therefore redundant, and are discarded; they will not be used further to calculate predecessors.

In the UN and UL models, the analysis is very similar. In atomic constraints of form $\langle s_1, s_2, w_1, w_2 \rangle$, the component w_i will be a bag rather than a sequence. The procedure looks very similar, except that constraints do not order messages in channels.

3.2 Forward Reachability Analysis

Backward reachability analysis, using calculation of preimages, is a suitable technique for checking that a certain invariant is satisfied. In fact, it generates the weakest inductive invariant which implies the invariant to be checked. In many cases, however, one would like to generate invariants which are as strong as possible. For instance, we could be interested in knowing the set of possible contents of a certain channel, or to generate a finite-state abstraction of the protocol, which will be useful in a subsequent analysis of a program with channels. To generate a representation of the set of reachable global states, forward reachability analysis is more appropriate. In this section, we will illustrate techniques for doing this in the OL and ON models.

Representing Sets in the OL Model. Consider what is a suitable symbolic representation of sets of global states in the OL model. Because of the possibility of losing messages, the set of reachable states will always be a downward closed (DC) set. Since any DC set is regular, the set of reachable global states of a program in the OL model be characterized as a regular set [5, 14]. On the other hand, it follows from undecidability results by Mayr [29] that this characterization cannot be effectively constructed. Various techniques for approximating the set of reachable states have been developed [9, 10, 3, 11], which are based on forward reachability analysis.

Let *post*(Γ) denote the set $\{\gamma \ : \ \exists \gamma' \in \Gamma \ . \ \gamma' \longrightarrow \gamma\}$. One immediately observes that the naive forward reachability analysis, based on successive calculations of *post*$^i(\Gamma_0)$, where Γ_0 is the set of initial states, will not converge whenever the program contains a loop with retransmissions. Therefore, this technique has been extended with *acceleration* techniques for calculating the effect of certain kinds of loops. Below we will illustrate the approach of [3] for the OL model.

A *Simple Regular Expressions (SRE)* is a sum of products of regular expressions of form $(m+\varepsilon)$ and $(m_1+\cdots+m_n)^*$. In [3], it is shown that *SREs* can express exactly the class of DC sets, and how to calculate the postcondition of an arbitrary number of iterations of a simple loop with respect to an SRE. We will illustrate this on the example.

Example Let us apply forward reachability analysis to the example in Figure 1. We will represent sets of states as unions of 4-tuples $\langle s_1, s_2, r_1, r_2\rangle$ where s_i is the control state of process i, and r_i is an *SRE* representing a set of contents of channel c_i. The set of initial states is the set $\langle I, I, \varepsilon, \varepsilon\rangle$. We first explore the effects of the simple self-loops in the states I, which is obviously to add an arbitrary sequence of messages to the corresponding channel. This means that the set of states $\langle I, I, 2^*, 1^*\rangle$ is reachable. The transitions from this set to control state W in each process add the sets represented by

$$\langle W, I, 2^*, 1^*(2+\varepsilon)\rangle \;\; \langle I, W, 2^*(1+\varepsilon), 1^*\rangle \;\; \langle W, W, 2^*(1+\varepsilon), 1^*(2+\varepsilon)\rangle \;\; .$$

The last set represents the effects of both transitions to W. Note that all these sets account for the possibility that the transmitted message is lost, by appending, e.g., $(1+\varepsilon)$ instead of just 1. Now comes the interesting step: to calculate the effects of the forwarding loops in states W. Consider the set $\langle W, W, 2^*(1+\varepsilon), 1^*(2+\varepsilon)\rangle$ and the self-loop of process 1 which receives message 2 and forwards it. Since there may be an unbounded number of 2's in the incoming channel, this loop may consume an arbitrary long initial string of 2's, and then transmit them to the outgoing channel. The effect is the set $\langle W, W, 2^*(1+\varepsilon), 1^*2^*\rangle$. We can now perform a similar argument using the forwarding loop of Process 2, and arrive at the set $\langle W, W, 2^*1^*, 1^*2^*\rangle$. By applying the same technique to the other sets in the previous display, we obtain the sets

$$\langle W, I, 2^*, 1^*2^*\rangle \;\; \langle I, W, 2^*1^*, 1^*\rangle \;\; \langle W, W, 2^*1^*, 1^*2^*\rangle \;\; .$$

Finally, we should consider the effect of transitions to final states. These transitions consume only one message, if it is present, and will to the already generated sets add the sets

$$\langle W, T2, 2^*, 2^*\rangle \;\; \langle T1, W, 1^*, 1^*2^*\rangle \;\; \langle W, T1, 2^*1^*, 2^*\rangle \;\; \langle T1, T1, 1^*, 2^*\rangle \;\; .$$

We see that the "dangerous" transition from control state $\langle W, T2\rangle$ to $\langle T1, T2\rangle$ is impossible since the incoming channel does not contain any message 1.

Abstract Transition Graph The result of the preceding analysis can be summarized in the symbolic abstract state transition graph, as in Figure 2, which describes the possible transitions between different control states. To each control state, we associate the possible channel contents. An abstract transition graph may be used in further analysis of the protocol, e.g., as in [1] for the Bounded Retransmission Protocol.

We note that backward reachability analysis, as in Section 3.1, is less suitable as a basis for generating an abstraction of the program. For this particular

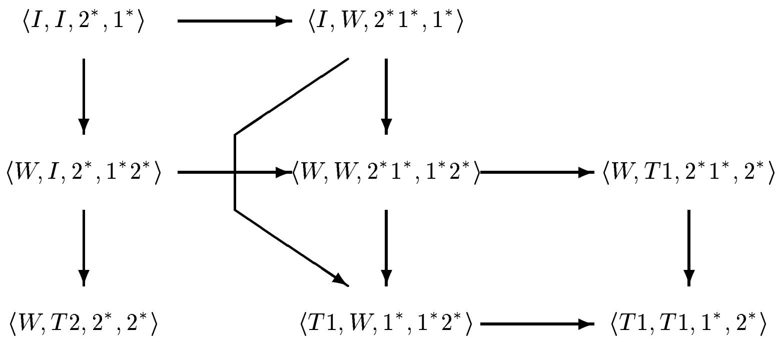

Fig. 2. Abstract Transition Diagram of Leader Election Protocol for Two Processes. All states except the bottom right have a self-loop.

example, the backward analysis gives no information about the channel contents when Process 2 is in control states W or $T1$, since the analysis investigates only the reachability of control state $\langle T1, T2 \rangle$.

Representing Contents by Finite Automata. Sets of channel states can also be represented as ordinary regular sets, using finite automata. This approach is the basis for the *Queue Decision Diagram (QDD)* representation developed by Boigelot and Godefroid [9, 10]. In this work, a global state is represented as a word, obtained by concatenation of the channel contents in a certain order. The approach is generally applicable in the ON model; for the OL model on can first modify the program (e.g., for each message transmission, a corresponding null transition is added). In [9] it is described how to calculate the effect of loops which first receive a (possibly empty) sequence from one channel and thereafter transmit a (possibly empty) sequence to another channel. The control loops of the processes in Figure 1 are of this form. In comparison to the techniques for *SRE*s described earlier, this approach is applicable to a more general model. On the other hand, using *SRE*s for the OL model, one can calculate the effect of a larger class of loops; a characterization of such loops is given in [2].

When applying the technique of [9] using QDDs to the example, we obtain essentially the same set of reachable states as that shown in Figure 2.

Forward reachability analysis in the UN and UL models can be performed analogously to the analysis for ordered channels. Techniques for calculating the effect of loops are well-developed (e.g., [26, 20, 18, 38]).

4 Channels with an Unbounded Message Alphabet

In the previous section, we assumed that messages in channels are taken from a finite set. In this section, we will consider a slightly more general model of programs that operate on a potentially infinite set of data, which can be stored in program variables, and transmitted as messages. We will consider program models in which very limited operations can be performed on data. In this section, we will consider a model where data values are equipped with a total order, which may be used in guards of transitions. We do not allow any other arithmetic on data values. One intended application is to consider protocols with time stamps, which may be compared with each other. In this section, we will also give a negative undecidability result, which holds in the more restricted model where we allow only tests for equality and inequality between values.

Programs. We extend the program model of the preceding section as follows. The finite set $\mathcal{M}$ of messages is replaced by an infinite set $\mathcal{D}$ of data values, which is equipped with a total order $<$. The control component is extended by a finite set $x_1, \ldots, x_m$ of *program variables*. Each transition in the control component may be labeled either by a receive statement of form

$$c?v; g(v, x_1, \ldots, x_m) \longrightarrow x_1, \ldots, x_m := p_1, \ldots, p_m$$

or a send statement of form

$$g(v, x_1, \ldots, x_m) \longrightarrow c!v; x_1, \ldots, x_m := p_1, \ldots, p_m$$

where v is a temporary variable and c is a channel in C; each parameter p_i is either v or among $x_1, \ldots, x_m$. The guard $g(v, x_1, \ldots, x_m)$ is a boolean combination of equalities ($=$) and comparisons (according to $<$) over v and $x_1, \ldots, x_m$. Intuitively, v is a free variable which may be bound to the received message in the receive statement, and which may be bound to any value in a send statement; in send statements it can represent the generation of new identifiers. We will sometimes use the abbreviation $\overline{x}$ for $x_1, \ldots, x_m$ and $\overline{p}$ for $p_1, \ldots, p_m$.

Semantics. A *global state* γ is a triple $\langle s, \sigma, w \rangle$, where $s \in S$, where σ is a mapping from $\{x_1, \ldots, x_m\}$ to $\mathcal{D}$, and where w maps each channel in C to a bag or string over $\mathcal{D}$. We use the notation $\sigma \models g$ to denote that the boolean expression g is satisfied by the mapping σ. The global state of a nonlossy system can change from γ to γ', denoted $\gamma \longrightarrow \gamma'$, as follows.

- For a transition from s to s' labeled by the receive statement $c?v; g(v, \overline{x}) \longrightarrow \overline{x} := \overline{p}$, we have $\langle s, \sigma, w[c := d \bullet w(c)] \rangle \longrightarrow \langle s', \sigma', w \rangle$ for any $d \in \mathcal{D}$ such that $\sigma \models g(d, \overline{x})$, where $\sigma'(x_i) = \sigma(p_i)$ if p_i is x_j for some j, otherwise (if p_i is v) $\sigma'(x_i)$ is d.
- For a transition from s to s' labeled by the send statement $g(v, \overline{x}) \longrightarrow c!v; \overline{x} := \overline{p}$, we have $\langle s, \sigma, w \rangle \longrightarrow \langle s', \sigma', w[c := w(c) \bullet d] \rangle$ for any $d \in \mathcal{D}$ such that $\sigma \models g(d, \overline{x})$, where $\sigma'(x_i)$ is $\sigma(p_i)$ if p_i is x_j for some j, otherwise (if p_i is v) $\sigma'(x_i)$ is d.

Definitions of transitions of lossy channels, reachable, are as in Section 2.

5 Reachability Analysis

We will in this section consider the control state reachability problem for programs defined in the model of Section 4. We will present two results. We first consider the OL model, where the control state reachability problem now becomes undecidable. In fact, it is undecidable also if only tests for equality and inequality between data values is allows. The undecidability result can be proven through a reduction from the control state reachability problem for perfect channel systems with a finite set of messages. Brand and Zafiropulo [12] showed that perfect channel systems can simulate Turing machines, and hence any nontrivial problem for perfect channel systems is undecidable. On the other hand, if guards may only test for equality (under an even number of negations) then the problem is decidable; the model is then similar to the model of data independent programs considered by Wolper [37].

5.1 Ordered Lossy Channels

In this section, we consider programs where guards are boolean combinations of equalities and inequalities. We then have the following negative result concerning decidability:

> The control state reachability problem is undecidable for OL programs where guards are boolean combinations of equalities and inequalities.

This can be shown by a reduction from the control state reachability problem for systems with perfect FIFO channels and a finite message alphabet, i.e., a program in the ON model of Section 2. Suppose we are given such a system $\mathcal{P}_{ON}$ and a control state s_f. We shall construct a program $\mathcal{P}_{OL}$ in the OL model over an infinite set $\mathcal{D}$ of values, which "simulates" $\mathcal{P}_{ON}$ in the sense that s_f is reachable in the lossy channel system if and only if s_f is reachable in $\mathcal{P}_{ON}$.

The main idea of the construction is to let $\mathcal{P}_{OL}$ be identical to $\mathcal{P}_{ON}$, but to add a protocol for transmission of messages, which makes it possible for the receiver at one channel to detect whether any message has been lost during transmission. If a loss is detected, the operation of the protocol is immediately aborted. Due to the availability of an infinite set of different messages, it is possible to assign "unique identifiers" to each message. This must be done in such a way that the receiver can detect whether some message has been lost. If the infinite set of different messages would have been the natural numbers, then if the $\mathcal{P}_{ON}$ transmits a sequence $m_1\ m_2\ m_3\ m_4 \ldots$ over a channel, then $\mathcal{P}_{OL}$ would transmit the sequence $0\ m_1\ 1\ m_2\ 2\ m_3\ 3\ m_4 \ldots$. It is easy to see that the receiver can detect whether any message has been lost.

In the current model, however, the set $\mathcal{D}$ is a set of "anonymous" values which can only be tested for equality and inequality. We therefore need a little trick to achieve the same effect. What is needed is to find for each message a "unique identifier" which is different from the previous identifiers, such that a receiver can detect when some identifier is missing. We can do this by using a predicate *fresh* to generate new sequence numbers, and to tag each new sequence number with a copy of the previous sequence number. Thus, in the above example, if the sequence of generated "fresh unique identifiers" is 0 1 2 3 . . ., then the sequence m_1 m_2 m_3 m_4 . . . will be equipped with sequence numbers as follows 0 m_1 1 0 m_2 2 1 m_3 3 2 m_4 . . .. Also here, a receiver can easily detect whether a message is missing, even if the sequence of identifiers is not known in advance.

The predicate *fresh* can be implemented by guessing an arbitrary identifier, and thereafter checking whether it already exists in the channel (this can be done by receiving all messages and then retransmitting them again).

The previous undecidability result illustrates a limitation of the approach of [4, 22] to deciding the reachability problem by using a preorder on global states which is monotonic and is a well quasi-ordering, as in Section 3.1. The limitation is that an unbounded data domain with two ordering relations usually cannot be equipped with a useful well quasi-ordering. In the current case, the data elements are ordered both in channels and by the equality/inequality relation. These two orderings create sufficiently much structure that the natural ordering between states is not a well quasi-ordering. Symbolic backward reachability analysis can still be performed, but it is not guaranteed to terminate.

5.2 Unordered Channels

Let us now consider the control state reachability problem for unordered (lossy or nonlossy) channels. We shall extend the symbolic backward reachability analysis of Section 3.1 to the program model in this section, using a symbolic representation introduced in [6] and [7].

We define an ordering $\preceq$ on global states, so that the conditions of monotonicity and well quasi-orderedness are preserved. The key property of $\preceq$ is that if $\langle s, \sigma, w\rangle \preceq \langle s', \sigma', w'\rangle$, then all guards which are satisfied in state $\langle s, \sigma, w\rangle$ are also satisfied in $\langle s', \sigma', w'\rangle$. Intuitively, this means that $\langle s, \sigma, w\rangle$ is "isomorphic" to a state obtained by deleting zero or more messages from the channels in $\langle s', \sigma', w'\rangle$. Formally, we say that $\langle s, \sigma, w\rangle \preceq \langle s', \sigma', w'\rangle$ if $s = s'$ and if there is an injection $h : \mathcal{D} \mapsto \mathcal{D}$ on the domain of data values which preserves the ordering $<$ such that $\sigma' = h \circ \sigma$ and such that $w(c)(d) \leq w'(c)(h(d))$ for each channel c and data element d (i.e., if in $w(c)$ we replace each occurrence of d by $h(d)$ then we obtain a subbag of $w'(c)$). One can prove that any program is monotonic with respect to $\preceq$ and that $\preceq$ is a well quasi-ordering. Hence for this class of programs, control state reachability can be decided using symbolic backward reachability analysis.

Example As an example of a protocol in this model, we consider again the leader election protocol. Let us model the algorithm for N processes, numbered in order from 0 to $N-1$. There are N channels $c_0, \ldots, c_{N-1}$, where channel c_i transmits messages from process $i-1$ to process i (arithmetic modulo N). Each process i as two local variables id_i and min_i, and will run the protocol in Figure 3. Correctness of the algorithm states that if all process reach the control state T, then min_i will have the same value in all processes. By instantiating

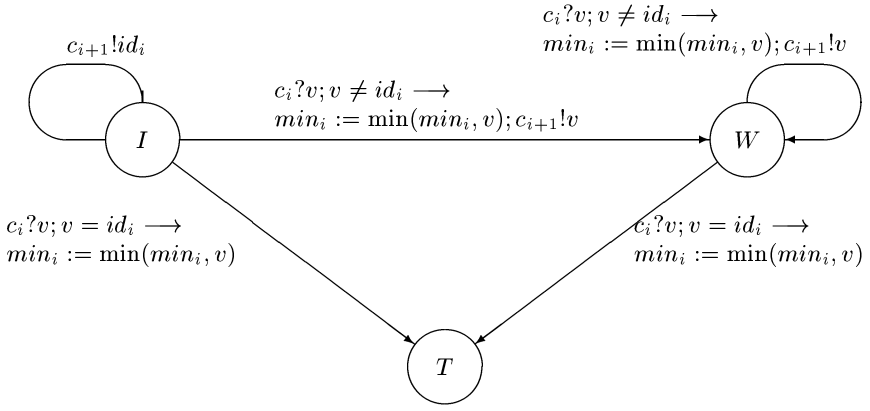

Fig. 3. Leader Election Protocol for Any Number of Processes

the above model to any particular number N of processes, we obtain a program in the model of this section. For each value of N, correctness can be expressed by requiring that the set of global states where all processes are in control state T and two processes disagree on the value of min_i is unreachable. For the case $N = 4$, the set can be represented symbolically as

$$\langle T, T, T, T, \phi \rangle$$

where ϕ is a formula stating that there are two processes that disagree on the value of min_i.

6 Conclusion

In this paper, we have surveyed some existing techniques for symbolic representation of unbounded channels in reachability analysis. We first considered the case where channels carry messages from a finite alphabet. We also briefly considered the case where messages are from an infinite set. It was shown that decidability may get lost in some models, but that in other frameworks it appears possible to incorporate this case into automated verification. More work seems necessary in order to understand whether and how messages from "interesting" data domains can be handled in general. Furthermore, it would be interesting to know whether protocols, such as the one in Section 5.2, can be verified parametrically for all values of N.

Acknowledgments. This work draws upon past and present collaboration, in particular with Aurore Annichini, Ahmed Bouajjani, Karlis Čerāns, Elena Fersman, Purushothaman Iyer, Mats Kindahl, Marcus Nilsson, Aletta Nylén, Yih-Kuen Tsay, and Wang Yi. Thanks to Pritha Mahata for valuable comments on drafts of the paper.

Acknowledgement

This work was supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082).

References

1. Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Algorithmic verification of lossy channel systems: An application to the bounded retransmission protocol. In *Proc. TACAS '99*, volume 1579 of *LNCS*, 1999.
2. Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages, 2001. To appear in Proc. ICALP '2001: 28^{th} International Colloquium on Automata, Lnaguages, and Programming.
3. Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc.* 10^{th} *Int. Conf. on Computer Aided Verification*, volume 1427 of *LNCS*, pages 305–318, 1998.
4. Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. LICS' 96* 11^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.
5. Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
6. Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes. In Bernhard Steffen, editor, *Proc. TACAS '98*, volume 1384 of *LNCS*, pages 298–312, 1998.
7. Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs, 2001. To appear in Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets.
8. T.P. Blumer and D.P. Sidhu. Mechanical verification of communication protocols. *IEEE Trans. on Software Engineering*, SE-12(8):827–843, Aug. 1986.
9. B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc.* 8^{th} *Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1996.
10. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, LNCS. Springer Verlag, 1997.
11. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, 1997.
12. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, April 1983.
13. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. LICS' 90,* 5^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1990.
14. Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 10 January 1996.
15. A. Choquet and A. Finkel. Simulation of linear FIFO nets having a structured set of terminal markings. In *Proc.* 8^{th} *European Workshop on Applications and Theory of Petri Nets*, 1987.

16. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.
17. G. Delzanno, J. Esparza, and A. Podelski. Constraint-based analysis of broadcast protocols. In *Proc. CSL'99*, 1999.
18. E.A. Emerson and K.S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. LICS' 98* 13^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 70–80, 1998.
19. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99* 14^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1999.
20. A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89:144–179, 1990.
21. A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3), 1994.
22. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere. Technical Report LSV-98-4, Ecole Normale Supérieure de Cachan, April 1998.
23. M. Gouda. Closed covers: to verify progress for communicating finite state machines. *IEEE Trans. on Software Engineering*, SE-10(6):846–855, Nov. 1984.
24. M.G. Gouda, E.M. Gurari, T.-H. Lai, and L.E. Rosier. On deadlock detection in systems of communicating finite state machines. *Computers and Artificial Intelligence*, 6(3):209–228, 1987.
25. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, SE-23(5):279–295, May 1997.
26. R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and Systems Sciences*, 3(2):147–195, May 1969.
27. K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1-2), 1997.
28. G. LeLann. Distributed systems – towards a formal approach. In B. Gilchrist, editor, *IFIP77*, pages 155–160. North-Holland, 1977.
29. R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.
30. J.K. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Protocol Specification, Testing, and Verification VII*, May 1987.
31. W. Peng and S. Purushothaman. Data flow analysis of communicating finite state machines. *ACM Trans. on Programming Languages and Systems*, 13(3):399–442, July 1991.
32. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *5th International Symposium on Programming, Turin*, volume 137 of *LNCS*, pages 337–352. Springer Verlag, 1982.
33. H. Rudin, C. West, and P. Zafiropulo. Automated protocol validation - one chain of development. In *Proc. Computer Network Protocols Symposium Liege*, 1978.
34. A.P. Sistla and L.D. Zuck. Automatic temporal verification of buffer systems. In Larsen and Skou, editors, *Proc. Workshop on Computer Aided Verification*, volume 575 of *LNCS*. Springer Verlag, 1991.
35. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.
36. C.H. West. Automated validation of a communications protocol: the ccitt x.21 recommendation. *IBM Journal on Research and Development*, 22(1), Jan. 1978.

37. Pierre 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.
38. Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *LNCS*, pages 88–97, Vancouver, July 1998. Springer Verlag.