An Improved Search Strategy for Lossy Channel Systems

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

Mats Kindahl
Dept. of Computer Systems
P.O. Box 325
S-751 05 Uppsala, Sweden
matkin@docs.uu.se

Doron Peled
Bell Laboratories
700 Mountain Avenue
Murray Hill, NJ 07974
doron@research.bell-labs.com

Key Words: Protocol validation, lossy channels, state space reduction, safety properties.

Abstract

In [1] we considered *lossy channel systems* which are a particular class of infinite state systems consisting of finite state processes communicating through channels that are unbounded and unreliable. We presented a backward reachability algorithm which, starting from a set of "bad" states, checks whether there is a backward path to the initial state of the system. Using standard techniques, the reachability algorithm can be used to check safety properties for lossy channel systems.

In this paper we adopt partial order techniques to improve the algorithm in [1]. We define a preorder, which we call the *better than relation*, among the set of transitions of the system. Intuitively a transition is better than another if choosing the first transition instead of the second preserves the reachability of the initial state during the analysis. This relation is weaker than the *independence relation*, which is an equivalence relation, used in traditional partial order methods, in the sense that two transitions are independent if and only if each of them is better than the other. Consequently, our method gives a better reduction in the number of states considered during the analysis. We demonstrate the efficiency of the approach by a number of experimental results.

1 Introduction

In the last few years, there has been a considerable interest in algorithmic verification of distributed and parallel systems. The research has led to the discovery of numerous efficient methods for the verification of *finite-state* systems ([9], [14], [21], [31], etc.). An obvious limitation of these methods is that systems with infinitely many states fall beyond their capabilities. Recently, algorithmic verification methods have been developed for some classes of infinite-state systems, such as certain types of real-time systems that operate on clocks [5, 32, 25], data-independent systems [24, 34], systems with many identical processes [13, 17, 29], context-free processes [10, 12, 11], relational automata [33], and Petri nets [23].

A particular class of infinite-state systems which has been important in the analysis of e.g. communication protocols consists of finite-state processes that communicate via unbounded FIFO channels [8, 7]. Such systems are infinite-state due to the unboundedness of the channels, and it is well-known that these systems can simulate Turing machines, and hence most interesting verification problems are undecidable for them [8]. In an earlier work [1], we considered a variant of this class, called *lossy channel systems*, where the FIFO channels are unreliable, in the sense that they may nondeterministically lose messages. In spite of this restriction, lossy channel systems can be used to model and analyze many interesting systems, e.g. link protocols such as the Alternating Bit Protocol [6] and HDLC [22], which are designed to operate correctly even in the case that the FIFO channels are faulty and may lose messages.

In [1, 2] we present a reachability algorithm for lossy channel systems which, for any state, checks whether the state is reachable form the initial state of the system. The idea of the algorithm is to define a partial order $\preceq$ on the set of states, where one state is smaller than another if the two states differ only in that the content of each channel in the first state is a (not necessarily contiguous) substring of the content of the same channel in the second state. An important property of lossy channel systems is that their behaviour is monotone with respect to $\preceq$, i.e. larger states have transitions to larger states. The algorithm operates on *ideals*, where the *ideal generated by a state* is the set of states larger than or equal to that state. The reachability of a state γ is obviously equivalent to the reachability of the ideal generated by γ, since if there is a path from the initial state to any state γ', which is larger than γ, then we can continue from γ' losing messages until we obtain γ. To check the reachability of an ideal I, we perform a reachability analysis starting from I and going backwards. At each step we pick an ideal I' which has already been generated during the analysis, and compute the set $pre(I)$ of states from which I' is reachable through the application of a single step of the transition relation. It is shown in [1, 2] that the monotonicity of lossy channel systems implies that $pre(I)$ is also an ideal and that it is in fact computable.

Partial order reductions [27, 28, 15, 18, 19, 30, 4] are a family of techniques which can be used to perform more efficient verification of systems consisting of asynchronously communicating concurrent processes, e.g. lossy channel systems. These methods are based on the observation that concurrent actions of the processes are often independent and hence their interleavings are equivalent in that they all lead to the same states. Most existing partial order methods work with a forward search of the state space, starting from the initial state and trying to find a path forwards to a final state. Action sequences are grouped into equivalence classes, for each of which only one representative may be analyzed. The algorithm then searches a reduced state-space, where at each step only a subset (an *ample set* [27]) of the enabled transitions is considered.

The requirement is that an ample set should contain at last one representative of each equivalence class.

In this paper, we employ ideas based on partial order reductions to derive a more efficient version of the algorithm in [1, 2]. Instead of working with an equivalence relation (the *independence* relation), we define a *preorder* (the *"better than"* relation) among actions. An event α is *better than* an event β in a state γ if α followed by β leads to a smaller state (with respect to $\preceq$) from γ than β followed by α. We extend the preorder to sequences of actions, where a sequence is "better than" another if the first sequence can be obtained from the second by permuting adjacent actions so that "better" actions occur earlier in the first sequence than in the second sequence. We show that it is sufficient, for each set of sequences, to consider only the "best" elements of the set. The correctness of our approach can be shown again to follow from monotonicity of lossy channel systems.

Our method differs from traditional partial order techniques in the following aspects.

- We work with a preorder (the "better than" relation) which is weaker than the independence relation, since two sequences are equivalent if and only each of them is better than the other. Instead of choosing representatives of each equivalent class, it is sufficient to choose the best elements of a certain set of sequences. This leads to a more efficient search algorithm, since two sequences may not be equivalent, but one of them may still be discarded from the analysis, if it is "worse" than the other.

- We apply our methods in the context of backward reachability analysis. This is necessary since the reachability algorithm for lossy channel systems operates backwards. In fact it is shown in [16] that forward reachability analysis is infeasible for lossy channel systems.

- The reachability algorithm works on ideals, so our method can be considered to operate on sets of states (ideals) rather than individual states. Since ideals are infinite sets, each ideal is represented symbolically by its set of generators. This set of generators can be shown always to be finite. In [4] partial order methods are also combined with symbolic model checking, where sets of states are represented by Binary Decision Diagrams (BDDs).

Using standard techniques [31, 19], we can check safety properties for lossy channel systems through a reduction to the reachability problem.

In fact our method is quite general. In [16, 3] general theories are presented for the verification of systems with monotone behaviour. Examples of such systems include besides lossy channel systems, Petri nets, real-time automata, relational automata [33], Basic Parallel Processes (BPPs) [11], and certain classes of parametrized systems [17]. The general reachability algorithm presented in [3] exploits the monotonicity of the system in the same manner as the algorithm for lossy channel systems. Consequently our techniques are applicable even for these classes of systems.

outline In the next section we introduce lossy channel systems and some properties of these. In section 3 we introduce the reachability algorithm. In section 4 we introduce the better-than relation and some properties of action sequences. In section 5 we show the modifications needed to improve the reachability algorithm. In section 6 we give an algorithm to compute ample sets. In section 7 we show some experimental results. In section 8 we give some conclusions and directions for future research.

2 Preliminaries

A *lossy channel system* consists of a *control part* and a *channel part*. The control part is modelled as a number of finite-state processes communicating via the channels, while the channel part consists of a finite set of channels. Each channel behaves as a FIFO buffer which is unbounded and unreliable in the sense that it can lose messages. A channel is used to perform asynchronous communication between a pair of processes, so for each channel there is unique process sending messages to the channel, and a unique process receiving messages from the channel.

For a tuple $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$, we use $\vec{x}(i)$ to denote x_i and $\vec{x}[i \mapsto e]$ to denote the tuple $\langle x_1, x_2, \ldots, x_{i-1}, e, x_{i+1}, \ldots, x_n \rangle$. The *domain* $\mathrm{dom}(\vec{x})$ of $\vec{x}$ is the set $\{1, 2, \ldots, n\}$ of indices, while the *range* $\mathrm{rng}(\vec{x})$ of $\vec{x}$ is the set $\{x_1, x_2, \ldots, x_n\}$ of values. For strings x and y, we use $x \cdot y$ to denote the concatination of x and y.

Formally lossy channel systems are defined as follows.

Definition 1 A *lossy channel system* is a tuple $\mathcal{L} = \left\langle \vec{S}, \vec{s}_0, A, C, M, \delta \right\rangle$, where

$\vec{S}$ $= S_1 \times \cdots \times S_n$ is a tuple of finite sets of *control states*, where n is the number of processes in the control part, and S_i denotes the set of control states of the i^{th} process.

$\vec{s}_0$ $\in \vec{S}$ is a tuple of *initial control states*,

A is a finite set of *actions*,

C is a finite set of *channels*,

M is a finite set of *messages*,

δ is a finite set of *transitions*, each of which is a triple of the form $\langle s_1^i, l, s_2^i \rangle$, where $s_1^i, s_2^i \in \vec{S}(i)$ for some process i, and l is of the form

- $c!m$ where $c \in C$ and $m \in M$, representing sending a message m to channel c (m is appended to the end of c);
- $c?m$ where $c \in C$ and $m \in M$, representing receiving a message m from channel c (m is removed from the head of c); or
- a where $a \in A$, representing an observable interaction with the environment without changing the contents of the channels. □

A *global state* (or just *state*) $(\vec{s}, \vec{w})$ consists of a tuple $\vec{s} \in \vec{S}$ of control states and a tuple $\vec{w}$ of strings over M. The string $\vec{w}$ is indexed by the elements of C, with $\vec{w}(c)$ representing the content of c. We use γ to denote a global state and use Γ to denote the set of all global states.

We formalise the intuitive behaviour of lossy channel systems by introducing a labelled transition system on global states.

Definition 2 Given a lossy channel system $\mathcal{L} = \left\langle \vec{S}, \vec{s}_0, A, C, M, \delta \right\rangle$ we construct the labelled transition system $\mathcal{G} = \langle \Gamma, \longrightarrow, A \cup \{\tau\} \rangle$ where Γ is the set of global states, $A \cup \{\tau\}$ is a set of labels, and $\longrightarrow \subseteq \Gamma \times (A \cup \{\tau\}) \times \Gamma$ is a transition relation among global states. We will use $\gamma_1 \xrightarrow{a} \gamma_2$ to denote $(\gamma_1, a, \gamma_2) \in \longrightarrow$. The set $\longrightarrow$ is the smallest set such that

- if $\langle s_1^i, c!m, s_2^i \rangle \in \delta$ and $(\vec{s}, \vec{w})$ is a global state such that $\vec{s}(i) = s_1^i$, then $(\vec{s}, \vec{w}) \xrightarrow{\tau} (\vec{s}[i \mapsto s_2^i], \vec{w}[c \mapsto \vec{w}(c) \cdot m])$. This corresponds to sending message m to channel c.
- If $\langle s_1^i, c?m, s_2^i \rangle \in \delta$ and $(\vec{s}, \vec{w})$ is a global state such that $\vec{s}(i) = s_1^i$ and $\vec{w}(c) = m \cdot v$, then $(\vec{s}, \vec{w}) \xrightarrow{\tau} (\vec{s}[i \mapsto s_2^i], \vec{w}[c \mapsto v])$. This corresponds to receiving message m from channel c. Notice that this transition may be performed only if the element in the head of channel c is equal to m.
- If $\langle s_1^i, a, s_2^i \rangle \in \delta$ and $(\vec{s}, \vec{w})$ is a global state such that $\vec{s}(i) = s_1^i$, then $(\vec{s}, \vec{w}) \xrightarrow{a} (\vec{s}[i \mapsto s_2^i], \vec{w})$. This corresponds to performing an observable interaction a with the environment.
- If $(\vec{s}, \vec{w})$ is a global state such that $\vec{w}(c) = x \cdot m \cdot y$, then $(\vec{s}, \vec{w}) \xrightarrow{\tau} (\vec{s}, \vec{w}[c \mapsto x \cdot y])$. This corresponds to losing message m from channel c. □

We use $(\vec{s}_1, \vec{w}_1) \longrightarrow (\vec{s}_2, \vec{w}_2)$ to denote that $(\vec{s}_1, \vec{w}_1) \xrightarrow{a} (\vec{s}_2, \vec{w}_2)$ for some $a \in A \cup \{\tau\}$, and use $\xrightarrow{*}$ to denote the transitive closure of $\longrightarrow$.

We define a partial order $\preceq$ on elements of M^* by letting $w_1 \preceq w_2$ iff w_1 is a (not necessarily contiguous) substring of w_2, e.g. given $M = \{a, b, c, x, y, z\}$, $ab \preceq abc \preceq axbyc \not\preceq axyc$. We use ε to denote the empty string. We extend the partial order $\preceq$ to states by letting $(\vec{s}_1, \vec{w}_1) \preceq (\vec{s}_2, \vec{w}_2)$ iff $\vec{s}_1 = \vec{s}_2$ and $\vec{w}_1(c) \preceq \vec{w}_2(c)$ for each $c \in C$. An interesting property of this partial order is that it is a well quasi-order, i.e. there is no infinite set of strings such that all strings are pairwise incomparable.

Lemma 1 (Higman's Lemma) Let M be a finite set. If $S \subseteq M^*$ is a subset such that all strings in S are pairwise incomparable with respect to $\preceq$, then S is finite.

Proof. This proof can be found in [26], where it is attributed to Higman [20]. □

The well quasi-orderedness of our partial order on global states follows directly from the lemma.

A set I of states is said to be an *ideal* if it is the case that $\gamma \in I$ and $\gamma \preceq \gamma'$ implies $\gamma' \in I$. For a state γ, the *ideal generated by* γ is the set $\{\gamma';\ \gamma \preceq \gamma'\}$.

3 A Reachability Algorithm

In this section we describe a reachability algorithm [1, 2] for lossy channel systems.

The *initial state* of a lossy channel system (denoted ι) is given by $(\vec{s}_0, \vec{w}_0)$, where $\vec{s}_0$ is the initial control state and $\vec{w}_0(c) = \varepsilon$ for each $c \in C$. We say that a state γ is *reachable* if $\iota \xrightarrow{*} \gamma$. We define the reachability problem formally as follows.

Definition 3 [**Reachability Problem**]

Instance: A lossy channel system $\mathcal{L} = \left\langle \vec{S}, \vec{s}_0, A, C, M, \delta \right\rangle$ and a set $F \subseteq \Gamma$ of *final states.*

Question: Is there a state $\gamma \in F$ such that $(\vec{s}_0, \vec{w}_0) \xrightarrow{*} \gamma$. □

The set F is usually used to represent a set of "bad" states which we do not want to occur during the execution of the system. It can be shown [1, 2] that, using standard techniques [31, 19], all safety properties, formulated as regular sets of allowed finite traces, can be reduced to the reachability problem.

To decide reachability, we perform a backward reachability analysis, starting with the states in F and try to find a path back to ι. Since the state space is infinite, the analysis is not *a priori* bounded. It turns out that it is inconvenient to choose the inverse of the forward transition relation (i.e. $\longrightarrow^{-1}$) as our basic backward transition step, since this would add messages to the channels in an uncontrolled manner. Instead we define a "backward" semantics for lossy channel systems as follows.

Let an *operation* be a partial function from states to states. An operation α is *enabled* at a state γ if $\alpha(\gamma)$ is defined for γ. The set of all enabled operations at state γ is denoted *enabled*(γ).

Definition 4 Let $(\vec{s}, \vec{w})$ be a state of a lossy channel system. We define three different operations, each of which corresponds to going backwards in the transition realtion. Notice that we execute the operations in reverse, e.g. receiving corresponds to adding a message to the beginning of a channel, etc.

rcv c, m A backwards *receive* operation. Enabled from $(\vec{s}, \vec{w})$ if $\langle s^i_2, c?m, s^i_1 \rangle \in \delta$ and $\vec{s}(i) = s^i_1$. When executed, the state $(\vec{s}[i \mapsto s^i_2], \vec{w}[c \mapsto m \cdot \vec{w}(c)])$ is obtained.

snd c, m A backwards *send* operation. Enabled from $(\vec{s}, \vec{w})$ if $\langle s^i_2, c!m, s^i_1 \rangle \in \delta$ and $\vec{s}(i) = s^i_1$. When executed, the state $(\vec{s}[i \mapsto s^i_2], \vec{w}[c \mapsto v])$ is obtained if $\vec{w}(c) = v \cdot m$ for some $v \in M^*$, otherwise $(\vec{s}[i \mapsto s^i_2], \vec{w})$ is obtained, corresponding to a send with a subsequent message loss.

lcl A backwards *local* operation. Enabled from $(\vec{s}, \vec{w})$ if $\langle s^i_2, a, s^i_1 \rangle \in \delta$ and $\vec{s}(i) = s^i_1$. When executed, the state $(\vec{s}[i \mapsto s^i_2], \vec{w})$ is obtained. □

Algorithm 1 *(Reachability Algorithm)*
Input: A lossy channel system $\mathcal{L}$ and a set F of states.
Output: *true* if any state in F is reachable, *false* otherwise.
Local: A set V containing the already visited states and a set W containing states to be investigated.

1) Add all states in F to the set W
2) If W is empty, exit with the result *false*
3) Select and remove a state $\gamma \in W$
4) If $\gamma \preceq \iota$, exit with the result *true*
5) If there is $\gamma' \in V$ such that $\gamma' \preceq \gamma$, goto 2
6) For each operation $\alpha \in enabled(\gamma)$
7)   Add $\alpha(\gamma)$ to W
8) Add γ to V
9) Goto 2

Figure 1: Reachability Algorithm

Intuitively, for a state γ and an operation α, if I and I' are the ideals generated by γ and $\alpha(\gamma)$ respectively, then I' is the set of states from which a state in I is reachable through a single application of α.

We are now ready to introduce the algorithm to decide reachability for lossy channel systems [2]. The algorithm uses a set V of "visited" states, and a "working" set W of states which are yet to be investigated. The algorithm proceeds by selecting and removing a state γ from the working set W. From γ the set of all enabled operations $enabled(\gamma)$ is computed. Each operation is used to compute a predecessor state $\alpha(\gamma)$, which is in turn added to the working set W. The state γ is then added to the set V. If we in our search find a state larger than some state in V, we know that the state is redundant since it gives no new information, hence we can throw it away. A detailed description of the algorithm can be seen in Figure 1.

Theorem 2 The reachability algorithm is correct in the sense that it always terminates, and it returns the value *true* if and only if a state in F is reachable.

Proof. The proof can be found in [2], where termination of the algorithm is shown to follow from Lemma 1. □

4 Dependency among Operations

In this section we define a dependency relation among operations.

For a state γ and a sequence of operations $\rho = \alpha_1\alpha_2\cdots\alpha_n$, we say that ρ is *enabled* from γ if, for each $1 \leq i < n$, α_i is enabled from $\alpha_{i-1}(\alpha_{i-2}(\ldots\alpha_2(\alpha_1(\gamma))))$. If ρ is enabled from γ then we use $\rho(\gamma)$ to denote the state $\gamma' = \alpha_n(\cdots\alpha_2(\alpha_1(\gamma)))$.

An operation α is *monotone* iff $\gamma \preceq \gamma'$ implies $\alpha(\gamma) \preceq \alpha(\gamma')$. Observe that this means that α is enabled in both γ and γ'.

Proposition 3 The operations $\mathsf{snd}\ c, m$, $\mathsf{rcv}\ c, m$ and lcl are monotone.

Proof. Follows directly from the definitions. $\square$

For a state γ and operations α and β, we say that α *is better than* β *in* γ when executing α before β results in a smaller state with respect to $\preceq$. It is defined formally as follows.

Definition 5 An operation α is better than an operation β in a state γ (denoted $\beta \sqsubseteq_b \alpha$) if and only if:

- $\alpha, \beta \in \mathit{enabled}(\gamma)$,
- $\alpha \in \mathit{enabled}(\beta(\gamma))$,
- $\beta \in \mathit{enabled}(\alpha(\gamma))$, and
- $\alpha\beta(\gamma) \preceq \beta\alpha(\gamma)$. $\square$

If $\beta \sqsubseteq_b \alpha$ and $\alpha \sqsubseteq_b \beta$ then α and β are *independent* and if $\beta \not\sqsubseteq_b \alpha$ and $\alpha \not\sqsubseteq_b \beta$ the operations are *dependent*.

Since $\alpha\beta(\gamma) \preceq \beta\alpha(\gamma)$, it is "better", from the point of view of the reachability algorithm, to perform α followed by β than the opposite. The reason is that we know that smaller states generate larger ideals. Consequently, it is safe to discard the sequence $\beta\alpha$ since the ideal generated by $\beta\alpha(\gamma)$ is a subset of the ideal generated by $\alpha(\beta(\gamma))$.

Proposition 4 Given lossy channel system $\mathcal{L}$ with operations $\mathsf{snd}\ c, m$, $\mathsf{rcv}\ c, m$, and lcl enabled in a state $(\vec{s}, \vec{w})$.

1. Operations of the same process are always dependent.
2. $\mathsf{rcv}\ c, n$ is better than $\mathsf{snd}\ d, m$ iff $c = d$, $\vec{w}(c) = \varepsilon$, and $m = n$; otherwise they are independent.
3. all other combinations are independent.

Proof. Follows directly from the definitions. $\square$

Notice that the operation $\mathsf{snd}\ c, n$ has two possible behaviours, depending on the state from which it is taken. The fact that $\mathsf{rcv}\ c, n$ is better than $\mathsf{snd}\ c, n$ makes use of this, replacing one type of execution by another.

Definition 6 Let γ be a state, ρ be a finite sequence such that $\rho \in \mathit{enabled}(\gamma)$ and α an operation. An operation α is *contributory* to ρ from state γ iff for any partition of ρ into $\sigma\beta\sigma'$, where β is an operation, $\beta \sqsubseteq_b \alpha$ holds in $\sigma(\gamma)$. $\square$

Intuitively, a contributory operation has the property that when it is moved forward in a sequence, we reach "smaller" states (according to $\preceq$).

Lemma 5 Let γ be a state, ρ be a finite sequence of monotone operations such that $\rho \in$ *enabled*(γ), and α a single operation. If α is contributory to ρ from γ then $\alpha\rho(\gamma) \preceq \rho\alpha(\gamma)$.

Proof. Proof by induction on the length of ρ. For the base case $|\rho| = 0$ the lemma holds trivially. For the induction case we let ρ be a sequence of operations such that $|\rho| > 0$ and α is contributory to ρ from γ. Let $\rho = \rho'\beta$ for some sequence of operations ρ' and some operation β. The operation α is better than every operation in ρ and in particular the operation β. Hence $\alpha\beta(\rho'(\gamma)) \preceq \beta\alpha(\rho'(\gamma))$, i.e.

$$\rho'\alpha\beta(\gamma) \preceq \rho'\beta\alpha(\gamma). \tag{1}$$

From the induction hypothesis we know that $\alpha\rho'(\gamma) \preceq \rho'\alpha(\gamma)$. Since β is monotone and enabled at $\alpha\rho'(\gamma)$ it follows that

$$\alpha\rho'\beta(\gamma) \preceq \rho'\alpha\beta(\gamma). \tag{2}$$

By combining (1) and (2) we see that $\alpha\rho'\beta(\gamma) \preceq \rho'\beta\alpha(\gamma)$. $\square$

A note on Lemma 5. We define a preorder $\sqsubseteq_B$ on sequences of operations, such that $\rho_1 \sqsubseteq_B \rho_2$ whenever $\rho_1 = \rho'\beta\alpha\rho'$ and $\rho_2 = \rho'\alpha\beta\rho'$, with $\beta \sqsubseteq_b \alpha$ in $\rho'(\gamma)$. We define the "better than" relation among sequences to be the reflexive transitive closure $\sqsubseteq_B^*$ of $\sqsubseteq_B$. Intuitively a sequence ρ_2 is better than a sequence ρ_1, if ρ_2 can be obtained from ρ_1 by permuting adjacent actions so that "better" actions occur earlier in ρ_2 than in ρ_1. It follows from Lemma 5 that $\rho_2(\gamma) \preceq \rho_1(\gamma)$.

This approach is more general than that used in traditional partial reduction methods, where $\sqsubseteq_b$ is taken to be an equivalence relation and hence $\sqsubseteq_B^*$ becomes an equivalence (rather than a preorder) on operation sequences.

5 Improving the Reachability Algorithm

In this section we introduce two strategies to improve the search conducted in Algorithm 1. The general idea is to consider only a subset of the enabled operations for each state. We replace *enabled*(γ) on line 6 in Algorithm 1 with *ample*(γ) [27], where *ample*$(\gamma) \subseteq$ *enabled*(γ).

We now proceed by investigating how to select a subset *ample*(γ) of *enabled*(γ). For Algorithm 1 to be correct when replacing *enabled*(γ) with *ample*(γ), the selection of a subset of *enabled*(γ) is guided by the following rules.

B0 The set *ample*(γ) is empty if and only if *enabled*(γ) is empty.

B1 Each of the operations in *ample*(γ) is contributory to every finite sequence of operations ρ enabled from γ that does not contain an operation from *ample*(γ).

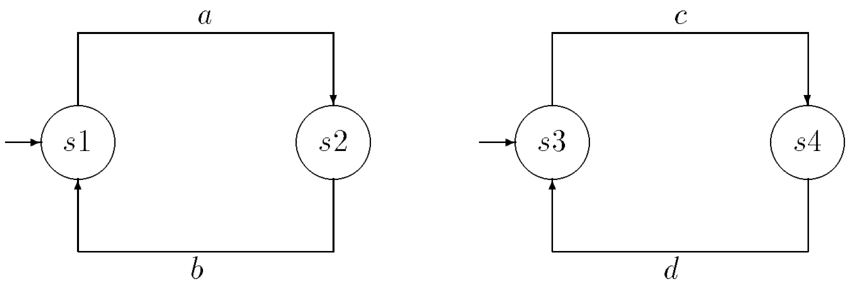

Figure 2: A system of two concurrent processes.

It can be shown that selecting a set $ample(\gamma)$ satisfying conditions **B0** and **B1** is not guaranteed to preserve reachability of the initial state. The reason is that because of independence between operations, we may reach a local state in one process before doing so in a second process; we may then continue going backwards in the first process, before reaching the control initial state in the second process and thus we miss the configuration where all processes are in their initial control states. This is a problem inherent in partial order reductions and not particular for lossy channel systems.

As an example, consider the system in Figure 2. It consists of two processes, executing concurrently. All the transitions (and hence the operations) are local. When trying to search for the initial state (s_1, s_3) from the state (s_2, s_3), the righthand process has already reached its initial control state. However, according to condition **B0**, we may select the operation which is the reverse of transition d. We then reach the state (s_2, s_4). In fact, we may then continue to take the reverse of transition c and reach (s_2, s_3) again, terminating the search with the wrong conclusion that the initial state is not reachable.

We have several alternative ways to remedy the problem. One of the simplest ways is to ensure that it is not possible to leave an inital control states once the search has reached it. This can be accomplished by making the system *rooted*.

Definition 7 A system is *rooted* if none of its operation is enabled from its initial state.

Thus we have our first search strategy:

STRAT1 A backward search in a rooted system (or a system transformed to become rooted), where each ample set satisfies conditions **B0** and **B1**.

Theorem 6 The correctness of the backward reachability problem is preserved under strategy STRAT1.

Proof. It is easy to see that the reduced state space generates only states that can be generated by the full search. Thus, if an initial state is not reachable, the

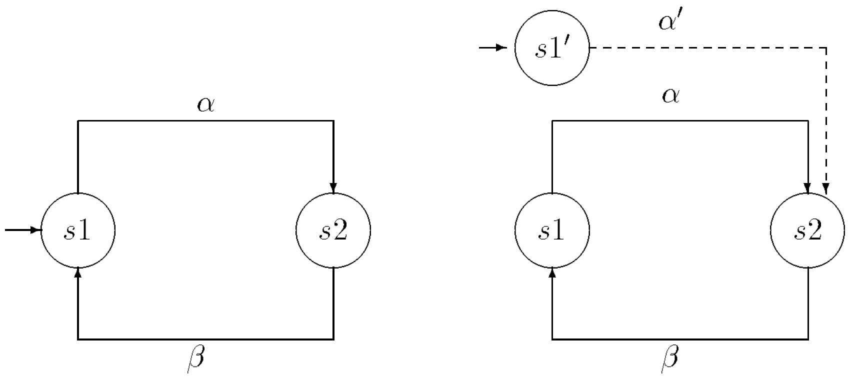

Figure 3: A process transformed to become rooted.

STRAT1 search will terminate (with the same termination argument of the original algorithm [2]) with a negative answer.

For the other direction, assume that an initial state is reachable.

Let γ be a state of the system, with $\alpha \in enabled(\gamma) \setminus ample(\gamma)$, such that there is a path $\alpha\rho$ from γ to an initial state ι.

Consider first the case where we partition $\alpha\rho$ into a sequence $\sigma\beta\sigma'$ such that the sequence σ does not contain any operation from $ample(\gamma)$. Then according to condition **B1** and Lemma 5, $\beta\sigma\sigma'(\gamma) \preceq \sigma\beta\sigma'(\gamma)$. Hence $\beta\sigma\sigma'$ is a sequence leading to the initial state ι.

Consider now the case that $\alpha\rho$ does not contain any operation from $ample(\gamma)$. Then, according to condition **B0** and **B1**, $ample(\gamma)$ contains some operation β that is contributory to $\alpha\rho$. But then according to Lemma 5, β is enabled from the initial state ι, contradicting the fact that the system is rooted. $\square$

It is easy to see that one can convert any system to a rooted one (if it is not rooted already). The disadvantage of having a rooted system is that it may introduce additional non-determinism. Figure 3 demonstrate a system that is not rooted (on the right) that is converted into a rooted system.

Another strategy, trying to avoid enlarging the state space in order not to miss the initial state, is the following.

Definition 8 A *delayed* operation is one that can lead out of an initial state. $\square$

It is not always simple to check which operations are delayed, yet it is easy to give a crude estimate. Thus, in its use in the sequel, it is safe to assume that an operation is delayed even if it is not. The price of assuming too many operations to be delayed is a smaller reduction. A simple heuristic is to define an operation to be delayed if it executes from an initial control state of a process.

For a system with identified delayed operations, we add the following condition on ample sets:

B2 Either $ample(\gamma)$ contains no delayed operations or $ample(\gamma) = enabled(\gamma)$.

For example, in the left process in Figure 3, the operation β is delayed, since one can take a β operation from the initial state. Thus, we have the following search strategy:

STRAT2— A backward search where each ample set satisfies conditions **B0**, **B1**, and **B2**.

Theorem 7 The correctness of the backward reachability problem is preserved under strategy STRAT2.

Proof. Similar to the proof of Theorem 6. The only difference is that in the contradiction case, if β is contributory to the sequence $\alpha\rho$ that transforms γ into ι, β can still be enabled from ι (there is no requirement that the system is rooted). However, this means that β is a delayed operation. But then according to **B2**, we must have selected $ample(\gamma) = enabled(\gamma)$, contradicting the fact that we did not select α. □

6 Algorithms for Ample Sets

Having defined conditions for ample sets, we need to provide an algorithm that guarantees selection of a subset of the enabled operations satisfying **B0** and **B1** and, depending upon the strategy, also **B2**. Conditions **B0** and **B2** require merely simple checks. However, to guarantee **B1**, we need to 'predict' what can happen along sequences of operations, starting from the current state. Deciding condition **B1** for an arbitrary subset of the enabled transitions can be easily shown to be as difficult as reachability. Instead, some heuristics have to be used.

One simple algorithm is the following.

Algorithm 2 *(Simple Algorithm)*
Input: A set of processes $P_1, \ldots, P_n$ and a state γ.
Output: A set $ample(\gamma)$ being a subset of $enabled(\gamma)$.

For each process P_i
    If each operation is either lcl, rcv d, m, or snd c, m with channel c non-empty
        select $ample(\gamma)$ as the enabled operations of P_i, exit
If there were no process satisfying the above conditions
    select $ample(\gamma) = enabled(\gamma)$

Lemma 8 The rules **B0** and **B1** are satisfied when using Algorithm 2 to compute $ample(\gamma)$.

Proof sketch. It is trivial to check that **B0** is satisfied. Observe that **B1** is satisfied since all operations belonging to a process are in $ample(\gamma)$ and that no two processes may receive from the same channel. Hence, it cannot happen that an alternative operation of the same process is disabled but will become enabled by the execution of some snd or rcv event in another process. □

To use strategy STRAT2 we alter steps 3 and 6 in Algorithm 2: **B2** is applied by selecting the transitions of a certain process only if they also satisfy the condition that they are not delayed.

Theorem 9 Algorithm 1 correctly decides reachability under STRAT1 when using Algorithm 2 to compute $ample(\gamma)$.

Proof. Follows from Theorem 6 and Lemma 8.

7 Experiments

In this section we consider two examples and show the reductions obtained when using the improved algorithm for reachability.

First we consider the *go back n* protocol of the well known Sliding Window protocol family. We model the protocol as a lossy channel system with two unbounded and unreliable channels.

In the protocol, n corresponds to the size of the sender window. The receiver window is always of size 1. We have added a distinguished initial state to make the system rooted.

In Table 1 we compare the performances of Algorithm 1 with and without partial order reduction, for different sizes of the sender window.

Window	No Partial Order		With Partial Order		Reduction (%)	
Size	Memory	Time	Memory	Time	Memory	Time
1	16152	0.02	14744	0.02	9%	0%
2	92416	0.10	67024	0.08	27%	20%
3	431096	0.42	297016	0.34	31%	19%
4	1446040	1.67	968184	1.26	33%	25%
5	4022080	4.94	2718224	4.24	32%	14%
6	9656504	13.34	6435560	11.22	33%	16%
7	21013480	32.71	13840888	29.81	34%	9%

Table 1: Verification of *Go Back n*

In the second example, we consider a token ring to implement mutual exclusion among a number of processes. A number of processes communicate through lossy channels. In Table 2 we compare the performances of Algorithm 1 with and without partial order reduction, for different number of processes in the ring. The high reduction achieved in this example compared to the first example, is due to the fact that here we have more parallel processes instead of only two processes in the case of the "go back n" protocol.

	Algorithm 1		Algorithm 2		Reduction (%)	
Processes	Memory	Time	Memory	Time	Memory	Time
4	66072	0.08	48264	0.04	27%	50%
5	733200	4.18	238416	0.40	67%	90%
6	5154440	161.16	1201856	5.63	77%	97%
7	34531204	6913.97	6555268	222.07	81%	97%

Table 2: Verification of the Mutex Protocol

8 Conclusion & Future Work

We have considered the application of partial order reduction techniques for lossy channel systems.

In contrast to most existing partial order methods, our approach is applied to a backward reachability algorithm. The choice of transitions selected during the analysis is guided by a preorder which we call the *better than relation*. This is a weaker relation than the *independence relation* used in traditional partial order methods, and leads consequently to a more effective analysis. The idea of using the "better than" relation is general as it is applicable to all systems which have "monotone" behaviour, e.g. Petri nets, real-time automata, relational automata, etc. As part of our future research, we shall try to extend our theory to these classes of systems. We also intend to carry out experiments to study the performance of our algorithms on more advanced examples.

References

[1] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proc. 8^{th} IEEE Int. Symp. on Logic in Computer Science*, pages 160–170, 1993.

[2] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[3] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11^{th} IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.

[4] R. Alur, R. Brayton, T. Henzinger, S. Qadeer, and S. Rajamani. Partial order reduction in symbolic state space exploration. In *Proc. 9^{th} Int. Conf. on Computer Aided Verification*, 1997. to appear.

[5] 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.

[6] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmissions over half duplex lines. *Communications of the ACM*, 2(5):260–261, 1969.

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

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

[9] 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. 5^{th} IEEE Int. Symp. on Logic in Computer Science*, 1990.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] P. Degano, R. de Nicola, and U. Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In de Bakker, de Roever, and Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 438–466. Springer Verlag, 1989.

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

[17] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[18] P. Godefroid and P. Wolper. Using partial orders to improve automatic verification methods. In *Proc. Workshop on Computer Aided Verification*, 1990.

[19] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

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

[21] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[22] ISO. Data communications – HDLC procedures – elements of procedures. Technical Report ISO 4335, International Standards Organization, Geneva, Switzerland, 1979.

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

[24] 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.

[25] K. Larsen and W. Yi. Time-abstracted bisimulation: Implicit specification and decidability. In *Mathematical Foundations of Programming Semantics*, Lecture Notes in Computer Science, New Orleans, April 1993. Springer Verlag.

[26] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1983.

[27] D. Peled. Combining partial order reductions with on-the-fly model checking. *Formal Aspects of Computing*, 8:39–64, 1996.

[28] D. Peled. Linear and branching temporal logics and process algebras. In *POMIV'96, Partial Order Method in Verification,*, pages 233–257. American Mathematical Society, 1996.

[29] 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.

[30] A. Valmari. On-the-fly verification with stubborn sets. In Courcoubetis, editor, *Proc.* 5^{th} *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, pages 59–70, 1993.

[31] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc.* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[32] 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.

[33] 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.

[34] 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.