On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels

Parosh Aziz Abdulla[1], Ahmed Bouajjani[2], and Bengt Jonsson[1]

[1] Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden, {parosh,bengt}@docs.uu.se
[2] VERIMAG, Centre Equation, 2 av. de Vignate 38610 Gieres, France, Ahmed.Bouajjani@imag.fr

Abstract. We consider symbolic on-the-fly verification methods for systems of finite-state machines that communicate by exchanging messages via unbounded and lossy FIFO queues. We propose a novel representation formalism, called *simple regular expressions* (SREs), for representing sets of states of protocols with lossy FIFO channels. We show that the class of languages representable by SREs is exactly the class of downward closed languages that arise in the analysis of such protocols. We give methods for (i) computing inclusion between SREs, (ii) an SRE representing the set of states reachable by executing a single transition in a system, and (iii) an SRE representing the set of states reachable by an arbitrary number of executions of a control loop of a program. All these operations are rather simple and can be carried out in polynomial time. With these techniques, one can construct a semi-algorithm which explores the set of reachable states of a protocol, in order to check various safety properties.

1 Introduction

One of the most popular models for specifying and verifying communication protocols is that of *Communicating Finite State Machines (CFSM)* [10, 8]. This model consists of finite-state processes that exchange messages via unbounded FIFO queues. Several verification methods have been developed for CFSMs [10, 11, 15, 18–20]. However, since all interesting verification problems are undecidable [10], there is in general no completely automatic verification method for this class of systems.

A way to obtain a decidable verification problem is to consider *lossy channel systems*, where the unbounded FIFO channels are assumed to be *lossy*, in the sense that they can at any time lose messages. This restricted model covers a large class of communication protocols, e.g., link protocols. In our earlier work [2], we showed the decidability and provided algorithms for verification of safety properties and some forms of liveness properties for lossy channel systems. Our algorithm for verifying safety properties is *global*, in the sense that it performs a backward search, starting from a set of "bad" states and trying to reach some initial state. In contrast, many efficient verification methods are so-called on-the-fly algorithms [17, 13], in which the state-space is explored in a forward search, starting from the initial states. In this paper, we therefore consider how forward verification can be carried out for lossy channel systems.

For that we adopt a symbolic verification approach. One of the main challenges in developing verification methods for a class of systems is to choose a symbolic representation of (possibly infinite) sets of states of a system. The symbolic representation should be expressive, yet allow efficient performance of certain operations which are often used in symbolic verification algorithms. Examples of such operations include checking for inclusion, and computing the states that can be reached by executing a transition of the system. In order to speed up the search through the state space, it is also desirable to be able to calculate, in one step, the set of states that can be reached by executing *sequences* of transitions. For instance, we can consider the set of sequences corresponding to an arbitrary number of executions of a control loop. This technique to speed up the reachability search has been applied e.g. for systems with counters[9] and *perfect* channel systems [3, 5]. Once a symbolic representations has been obtained it can used for many types of verification and model checking problems.

In this paper, we propose a novel representation formalism, called *simple regular expressions* (SREs), for use in verifying protocols modelled as lossy channel systems. SREs constitute a simple subclass of regular expressions. To our knowledge, this class has not been studied before. Because of the lossiness, we need only to represent sets of channel contents that are closed with respect to the subsequence relation. For example, if a channel can contain the sequence abc, then it can also contain the sequences ab, ac, bc, a, b, c, and ϵ. It is well-known that downward closed languages are always regular. We strengthen this result and show that in fact the class of downward closed languages corresponds exactly to those recognized by SREs. This implies that for any lossy channel system we represent the set of reachable states as an SRE. We suggest methods for computing:

- inclusion between SREs, which can be done in quadratic time,
- an SRE obtained by executing a single transition, and
- an SRE obtained by an arbitrary number of executions of a control loop of a program. It turns out that this operation is not very complicated and can be carried out in polynomial time.

With these techniques, one can straightforwardly construct an algorithm which explores the set of reachable states of a protocol, in order to check various properties. This algorithm is parametrized by the set of control loops that are used to speed up the reachability set computation. We also show how one can perform model-checking of LTL properties, using a standard construction of taking the cross-product of the protocol and a Büchi automaton that recognizes the complement of the LTL property in question. It should be noted that all these methods are incomplete, i.e., they may sometimes not terminate. The incompleteness of our methods is unavoidable despite the facts that reachability is decidable for lossy channel systems, and that the set of reachable states is representable by an SRE. This is due to a basic result [12] saying that there is no general algorithm for generating the set of reachable states.

As an illustration of the applicability of our methods and the SRE representation, we look at a few communication protocols that have been verified earlier in the literature. It turns out that the sets of reachable states of these protocols can be conveniently represented as SREs.

Related Work There are several other results on symbolic verification of perfect channel systems. Pachl [18] proposed to represent the set of reachable states of a protocol as a recognizable set. A recognizable set is a finite union of Cartesian products of regular sets. Pachl gave no efficient algorithms for computing such a representation. In [14] a symbolic analysis procedure is proposed using a class of regular expressions which is not comparable with SRE's. However, the computed reachability set by this procedure is not always exact.

Boigelot and Godefroid [3, 5] use finite automata (under the name QDDs) to represent recognizable sets of channel contents. In [5] it has been shown that the effect of every loop is recognizable for a system with a *single* fifo-channel. As soon as two channels are considered, the effect of a loop may be non-recognizable (i.e., not QDD representable). This is due to the fact that the repeated execution of a loop may create constraints between the number of occurrences of symbols in different channels. For instance, the iteration of a loop where a message is sent to two different channels generates pairs of sequences with the *same* length (assuming the channel is initially empty). In [5] a complete characterization is given of the types of loops which preserve recognizability. To compute and represent the effect of any loop in a perfect fifo-channel, a representation structure, called CQDDs (constrained QDDs), combining finite automata with linear arithmetical constraints is needed [7]. In the case of lossy channels, the links between the number of occurrences in different channels are broken due to lossiness, and this simplifies the computation of the effect of loops, conceptually and practically (i.e., from the complexity point of view).

We argue that SREs offer several advantages when used as a symbolic representation in the context of lossy channel systems. First, the operations on QDD's and CQDD's are of exponential complexity and are performed by quite non-trivial algorithms (see e.g. [4, 6]), whereas all operations on SRE's can be performed by much simpler algorithms and in polynomial time. Moreover, we describe a normal form for SREs, and provide a polynomial procedure to transform an SRE to an equivalent normal SRE. While QDD's admit a canonical form via minimization, a corresponding result is not known for CQDD's. Also, SREs are closed under the performance of any loop, while QDDs are closed only under certain restricted types of loops.

Finally, although the data structures (QDDs and CQDDs) used in [3, 5, 7] are more general than SREs, the algorithms in [3, 5, 7] are not able to simulate the ones we present in this paper. The reason is that the lossy transitions are implicit in our model, whereas all transitions are explicitly represented in the algorithms in [3, 5, 7]. Thus to simulate in [3, 5, 7] the effect of iteration of a loop in the lossy channel model, we have to add transitions explicitly to model the losses. These transitions add in general new loops to the system, implying that a loop in the lossy channel system is simulated by a *nested* loop in the perfect channel system. However analysis of nested loops is not feasible in the approaches of [3,5,7].

Outline In the next section we give some preliminaries. In Section 3 we introduce the class Simple Regular Expressions (SREs). In Section 4 we describe how to check entailment among SREs. In Section 5 we give a normal form for SREs. In Section 6 we define operations for computing post-images of sets of configurations, represented as SREs. In Section 7 we show how to use SREs to perform on-the-fly verification algorithms for lossy channel systems. In Section 8 we illustrate our method with an example. Finally, in Section 9 we present conclusions and directions for future work.

2 Preliminaries

Assume a finite alphabet M. For $x, y \in M^*$ we let $x \bullet y$ denote the concatenation of x and y. We use x^n to denote the concatenation of n copies of x. The empty string is denoted by ϵ. We use $x \preceq y$ to denote that x is a (not necessarily contiguous) substring of y.

Consider a system modeled by a finite set of finite-state machines, that communicate through sending and receiving message via a finite set of unbounded FIFO channels. The channels are assumed to be *lossy* in the sense that they can nondeterministically lose messages. We model such a system as a lossy channel system.

Definition 1. *A* Lossy Channel System $\mathcal{L}$ *is a tuple* $\langle S, s_{init}, C, M, \delta \rangle$, *where*

- S *is a finite set of* (control) states. *The control states of a system with n finite-state machines is formed as the Cartesian product* $S = S_1 \times \cdots \times S_n$ *of the control states of each finite-state machine.*
- s_{init} $\in S$ *is an* initial state, *The initial state of a system with n finite-state machines is a tuple* $\langle s_{init1}, \cdots, s_{initn} \rangle$ *of initial states of the components.*
- 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* $\langle s_1, Op, s_2 \rangle$, *where* s_1 *and* s_2 *are states, and* Op *is a mapping from* C *to* (channel) operations. *An operation is either a* send *operation* $!a$, *a* receive *operation* $?a$, *or an empty operation* nop, *where* $a \in M$. □

A transition of form $\langle s_1, Op, s_2 \rangle$ represents a change of the control state from s_1 to s_2 while performing all the operations in Op. The operations $!a$, $?a$, nop represent sending a to the channel, receiving a from the channel, and not changing the content of the channel, respectively.

Global states of a lossy channel system are represented by configurations. A *configuration* γ of $\mathcal{L}$ is a pair $\langle s, w \rangle$, where $s \in S$ is a control state and w is a mapping from C to M^*. For two mappings w and w' from C to M^*, we use $w \preceq w'$ to denote that $w(c) \preceq w'(c)$ for each $c \in C$. We use ϵ to denote the mapping where each channel is assigned ϵ. The *initial configuration* γ_{init} of $\mathcal{L}$ is the pair $\langle s_{init}, \epsilon \rangle$. For each transition $\langle s_1, Op, s_2 \rangle \in \delta$, we define a *transition* *relation* $\xrightarrow{\langle s_1,Op,s_2\rangle}$ on configurations, such that $\langle s_1, w_1\rangle \xrightarrow{\langle s_1,Op,s_2\rangle} \langle s_2, w_2\rangle$ if and only if for each channel $c \in C$ we have

- if $Op(c) = !a$, then $w_2(c) = w_1(c) \bullet a$.
- if $Op(c) = ?a$, then $a \bullet w_2(c) = w_1(c)$.
- if $Op(c) = nop$, then $w_2(c) = w_1(c)$.

We define a *weak* transition relation on configurations: $\langle s_1, w_1\rangle \overset{\langle s_1,Op,s_2\rangle}{\Longrightarrow} \langle s_2, w_2\rangle$ if and only if there are w_1' and w_2' such that $w_1' \preceq w_1$ and $w_2 \preceq w_2'$ and $\langle s_1, w_1'\rangle \xrightarrow{\langle s_1,Op,s_2\rangle} \langle s_2, w_2'\rangle$. Intuitively, $\langle s_1, w_1\rangle \overset{\langle s_1,Op,s_2\rangle}{\Longrightarrow} \langle s_2, w_2\rangle$ denotes that $\langle s_2, w_2\rangle$ can be obtained from $\langle s_1, w_1\rangle$ by first losing messages from the channels, then performing the transition $\langle s_1, Op, s_2\rangle$ and thereafter losing messages from channels. We let $\langle s_1, w_1\rangle \Longrightarrow \langle s_2, w_2\rangle$ denote that there is a transition $\langle s_1, Op, s_2\rangle$ such that $\langle s_1, w_1\rangle \overset{\langle s_1,Op,s_2\rangle}{\Longrightarrow} \langle s_2, w_2\rangle$. We let $\overset{*}{\Longrightarrow}$ denote the reflexive transitive closure of $\Longrightarrow$. A configuration γ' is said to be *reachable* from a configuration γ if $\gamma \overset{*}{\Longrightarrow} \gamma'$. A configuration γ is said to be *reachable* if γ is reachable from the initial configuration γ_{init}. For a state s, we define $\mathcal{R}(s) = \left\{ w \mid \gamma_{init} \overset{*}{\Longrightarrow} \langle s, w\rangle \right\}$.

In symbolic verification, we are interested in manipulating sets of configurations, e.g., in order to compute $\mathcal{R}(s)$. Let Γ be a set of configurations. We use $\Gamma(s)$ to denote the set $\{w \mid \langle s, w\rangle \in \Gamma\}$. and $post(\langle s_1, Op, s_2\rangle, \Gamma)$ to denote the set $\left\{ \gamma' \mid \exists \gamma \in [\![\Gamma]\!]. \gamma \overset{\langle s_1,Op,s_2\rangle}{\Longrightarrow} \gamma' \right\}$.

3 Simple Regular Expressions (SREs)

We define a class of languages which can be used to describe the set of reachable configurations of a lossy channel system. Let M be a finite alphabet. We define the set of *regular expressions (REs)*, and the languages generated by them in the standard manner. For a regular expression r, we use $[\![r]\!]$ to denote the language defined by r. For regular expressions r_1 and r_2, we use $r_1 \equiv r_2$ ($r_1 \sqsubseteq r_2$) to denote that $[\![r_1]\!] = [\![r_2]\!]$ ($[\![r_1]\!] \subseteq [\![r_2]\!]$). By $r_1 \sqsubset r_2$ we mean that $r_1 \sqsubseteq r_2$ and $r_1 \not\equiv r_2$. In case $r_1 \sqsubseteq r_2$ we say that r_1 *entails* r_2. We use $\lambda(r)$ to denote the set of elements of M appearing in r.

We define a subset of the set of regular expressions, which we call the set of *simple regular expressions*, as follows.

Definition 2. *Let M be a finite alphabet. An* atomic expression *over M is a regular expression of the form*

- *$(a + \epsilon)$, where $a \in M$, or of the form*
- *$(a_1 + \ldots + a_m)^*$, where $a_1, \ldots, a_m \in M$.*

A product *p over M is a (possibly empty) concatenation $e_1 \bullet e_2 \bullet \cdots \bullet e_n$ of atomic expressions $e_1, \ldots, e_n$ over M. We use ϵ to denote the empty product, and assume that $[\![\epsilon]\!] = \{\epsilon\}$.*

A simple regular expression (SRE) *r over M is of the form $p_1 + \ldots + p_n$, where $p_1, \ldots, p_n$ are products over M. We use $\emptyset$ to denote the empty SRE, and* *assume that* $[\![\emptyset]\!]$ *is the empty language* $\emptyset$. *A language* L *is said to be* simply regular *if it is representable by an SRE.*

Let C *and* M *be finite alphabets. A* C-indexed language over M *is a mapping from* C *to languages over* M. *A* C-indexed RE (SRE) R *over* M *is a mapping from* C *to the set of REs (SREs) over* M. *The expression* R *defines a* C*-indexed language* K *over* M *where* $w \in K$ *if and only if* $w(c) \in [\![R(c)]\!]$ *for each* $c \in C$. *The entailment relation is extended to indexed REs in the obvious manner. An indexed language is said to be* simply recognizable *if it is a finite union of languages recognized by indexed SREs.* □

Definition 3. *Let* M *and* C *be finite alphabets. For a language* $L \subseteq M^*$, *we say that* L *is* downward closed *if* $x \in L$ *and* $y \preceq x$ *imply* $y \in L$. *The definition is generalized in the natural way to* C*-indexed languages over* M. □

Theorem 1. *For a finite alphabets* M *and* C *and a* C*-indexed language* L *over* M, *if* L *is downward-closed then* L *is simply recognizable.*

Proof. It is well-known that each downward-closed language is regular. The result follows from Higman's theorem [16] which states the following: for any finite alphabet M, and for any infinite sequence $x_1, x_2, \ldots$ of strings over M, there are $i < j$ such that $x_i \preceq x_j$.

Using induction on the set of REs, we can show that for each RE r, if $[\![r]\!]$ is downward-closed, then there is an SRE r' such that $r' \equiv r$. The result follows immediately. □

Since the set of reachable configurations of a lossy channel system is downward-closed, we get the following.

Corollary 1. *For a lossy channel system* $\mathcal{L}$ *and a state* s *in* $\mathcal{L}$, *the set* $\mathcal{R}(s)$ *is simply recognizable.*

However, it is shown in [12] that we cannot in general compute a representation of $\mathcal{R}(s)$. The uncomputability of $\mathcal{R}(s)$ is shown through a reduction to an undecidable problem reported in [1]. More precisely, in [1] we show the undecidability of the *recurrent state problem*: given a lossy channel system $\mathcal{L}$ and a state s in $\mathcal{L}$, is there a computation of $\mathcal{L}$ visiting s infinitely often? In [12] the uncomputability of a representation of $\mathcal{R}(s)$ is reduced to the recurrent state problem as follows. We add a new channel c to the lossy channel system. Whenever a computation reaches s, an arbitrary message is sent to c. Suppose that we can compute an indexed SRE R such that $[\![R]\!] = \mathcal{R}(s)$. It is clear that the existence of a computation visiting s infinitely often is equivalent to the finiteness of $[\![R(c)]\!]$.

Theorem 2. [12] *For a lossy channel system* $\mathcal{L}$ *and a state* s *in* $\mathcal{L}$, *there is, in general, no algorithm for computing a representation of* $\mathcal{R}(s)$.

Although we can compute a representation of the set of configurations from which a given configuration is reachable ([2]), we cannot in general compute a representation of the set of configuration which are reachable from a given configuration (Theorem 2). This means that we can have a complete algorithm for performing backward reachability analysis in lossy channel systems, while any procedure for performing forward reachability analysis will necessarily be incomplete.

4 Entailment among SREs

In this section, we consider how to check entailment between SREs. First, we show a preliminary lemma about entailment.

Lemma 1. *For products $p, p_1, \dots, p_n$, if $p \sqsubseteq p_1 + \dots + p_n$ then $p \sqsubseteq p_i$ for some $i \in \{1 \dots n\}$.*

Proof. Given any natural number k, we define a sequence x such that $x \in [\![p]\!]$ and $x \notin [\![p']\!]$, for any product p', where $p \not\sqsubseteq p'$ and where p' contains at most k atomic expressions. The result follows immediately. Let $p = e_1 \bullet \cdots \bullet e_m$. We define $x = y_1 \bullet \cdots \bullet y_m$, where y_i is defined as follows. If $e_i = (a + \epsilon)$ then $y_i = a$. If $e_i = (a_1 + \dots + a_\ell)^*$ then $y_i = (a_1 \bullet \cdots \bullet a_\ell)^{k+1}$. □

Let us identify atomic expressions of form $(a_1 + \dots + a_m)^*$ which have the same set $a_1, \dots, a_m$ of symbols. Then $\sqsubseteq$ is a partial order on atomic expressions. It is the least partial order which satisfies

$$\begin{array}{ll} (a + \epsilon) \sqsubseteq (a_1 + \dots + a_m)^* & \text{if } a \in \{a_1, \dots, a_m\} \\ (a_1 + \dots + a_m)^* \sqsubseteq (b_1 + \dots + b_n)^* & \text{if } \{a_1, \dots, a_m\} \subseteq \{b_1, \dots, b_n\} \end{array}$$

Lemma 2. *Entailment among products can be checked in linear time.*

Proof. The result follows from the fact that $\epsilon \sqsubseteq p$, $p \not\sqsubseteq \epsilon$ if $p \neq \epsilon$, and $e_1 \bullet p_1 \sqsubseteq e_2 \bullet p_2$ if and only if one of the following holds:

- $e_1 \not\sqsubseteq e_2$ and $e_1 \bullet p_1 \sqsubseteq p_2$.
- $e_1 = e_2 = (a + \epsilon)$ and $p_1 \sqsubseteq p_2$.
- $e_2 = (a_1 + \cdots + a_n)^*$, $e_1 \sqsubseteq e_2$, and $p_1 \sqsubseteq e_2 \bullet p_2$. □

Lemma 3. *Entailment among SREs can be checked in quadratic time.*

Proof. The proof follows from Lemma 1 and Lemma 2. □

Corollary 2. *Entailment among indexed SREs can be checked in quadratic time.*

5 Normal Forms for SREs

In this section, we show how to compute normal forms for SREs. First we define a normal form for products.

Definition 4. *A product $e_1 \bullet \cdots \bullet e_n$ is said to be* normal *if for each $i : 1 \leq i < n$ we have $e_i \bullet e_{i+1} \not\sqsubseteq e_{i+1}$ and $e_i \bullet e_{i+1} \not\sqsubseteq e_i$.* □

Lemma 4. *For each product p, there is a unique normal product, which we denote $\overline{p}$, such that $\overline{p} \equiv p$. Furthermore, $\overline{p}$ can be derived from p in linear time.*

Proof. We can define $\overline{p}$ from p by simply deleting atomic expressions which are redundant according to Definition 4. □

Similarly, we can define a normal form for SREs.

Definition 5. *An SRE $r = p_1 + \ldots + p_n$ is said to be* normal *if each p_i is normal for $i : 1 \leq i \leq n$, and $p_i \not\sqsubseteq p_j$, for $i, j : 1 \leq i \neq j \leq n$.* □

In the following, we shall identify SREs if they have the same sets of products.

Lemma 5. *For each SRE r, there is a unique (up to commutativity of products) normal SRE, which we denote by $\overline{r}$, such that $\overline{r} \equiv r$. Furthermore, $\overline{r}$ can be derived from r in quadratic time.*

Proof. The proof follows from Lemma 2, Lemma 1 and Lemma 4. □

6 Operations on SREs

In this section, we will define operations for computing post-images of sets of configurations, represented as SREs, with respect to transitions of a lossy channel system. We will also define operations for computing post-images of sets of configurations with respect to an arbitrary number of repetitions of an arbitrary control loop in a lossy channel system.

Throughout this section, we assume a fixed finite set C of channels and a finite alphabet M. We will first consider operations on SREs corresponding to single transitions, and thereafter consider loops.

6.1 Computing the Effect of Single Transitions

Consider a language L and an operation $op \in \{!a, ?a, nop\}$. We define $L \otimes op$ to be the smallest downward closed language such that $y \in (L \otimes op)$ if there is an $x \in L$ satisfying one of the following three conditions: (i) $op = !a$, and $y = x \bullet a$; or (ii) $op = ?a$, and $a \bullet y = x$; or (iii) $op = nop$, and $y = x$.

For an indexed language K, and a mapping Op from C to operations, we define $K \otimes Op$ to be the indexed language where $(K \otimes Op)(c) = K(c) \otimes Op(c)$, for each $c \in C$. Notice that, for a lossy channel system $\mathcal{L}$, a transition $\langle s_1, Op, s_2 \rangle$, and a set Γ of configurations in $\mathcal{L}$, the set $post(\langle s_1, Op, s_2 \rangle, \Gamma)$ is given by $\{\langle s_2, w \rangle \mid w \in (\Gamma(s_1) \otimes Op)\}$.

The following propositions show how to compute the effect of single operations on SREs.

Lemma 6. *For an SRE r and an operation op, there is an SRE, which we denote $r \otimes op$, such that $[\![r \otimes op]\!] = [\![r]\!] \otimes op$. Furthermore, $r \otimes op$ can be computed in linear time.*

Proof. For a product p and an operation op, we have $p \otimes (!a) = p \bullet (a + \epsilon)$, and $p \otimes (nop) = p$. Furthermore, $\epsilon \otimes (?a) = \emptyset$. and if $p = e \bullet p_1$, then

$$p \otimes (?a) = \begin{cases} p & \text{if } e = (a_1 + \ldots + a_n)^* \text{ and } a \in \{a_1 + \ldots + a_n\} \\ p_1 & \text{if } e = (a + \epsilon) \\ p_1 \otimes (?a) & \text{otherwise} \end{cases}$$

For an SRE $p_1 + \ldots + p_m$ we have

$$(p_1 + \ldots + p_m) \otimes op = (p_1 \otimes op) + \ldots + (p_m \otimes op)$$

Lemma 6 can be generalized in the obvious manner to indexed SREs.

6.2 Computing the Effect of Loops

We study methods to accelerate reachability analysis of lossy channels systems. The basic idea is that, rather than generating successor configurations with respect to single $\Longrightarrow$-transitions, we shall consider the effect of performing sets of *sequences* of transitions in each step. We consider *control loops*, i.e., sequences of transitions starting and ending in the same control state. If *ops* is the sequence of channel operations associated with a control loop, then we shall calculate the effect on an SRE of performing an arbitrary number of iterations of *ops*. In Lemma 7, we show that for each SRE and sequence *ops*, there is an n such that the set of all strings which can be obtained through performing n or more iterations of *ops* on the SRE can be characterized by a (rather simple) SRE. In other words, the effect of the loop "stabilizes" after at most n iterations, in the sense it only generates strings belonging to a single SRE. This implies that the effect of performing an arbitrary number of iterations of the loop can be represented as the union of n SREs: one of them represents all iterations after n, while the remaining SREs each represents the effect of iterating the loop exactly j times for $j : 1 \leq j \leq n-1$. In Corollary 3 we generalize the result to indexed SREs.

For strings x and y, we use $x \preceq_c y$ to denote that there are x_1 and x_2 such that $x = x_1 \bullet x_2$ and $x_2 \bullet x_1 \preceq y$. The relation $\preceq_c$ can be decided in quadratic time. We use $x \preceq^+ y$ to denote that there is a natural number $m \geq 1$ such that $x^{m+1} \preceq y^m$. It can be shown that if m exists then m can be found in the interval $1 \leq m \leq |y|$. It follows that the relation $\preceq^+$ can be checked in quadratic time. For a sequence $ops = op_1 op_2 \cdots op_n$ of operations, we define $L \otimes ops$ to be $L \otimes op_1 \otimes op_2 \otimes \cdots \otimes op_n$. We use ops^m (Ops^m) to denote the concatenation of m copies of *ops* (*Ops*). By *ops*! (*ops*?) we mean the subsequence of *ops* which contains only send (receive) operations. For a product p, let $|p|$ denote the number of atomic expressions in p.

Lemma 7. *For a product p and a sequence ops of operations, the following holds. There is a natural number n and a product p' such that either $p \otimes ops^n = \emptyset$ or $p' = \cup_{j \geq n} [\![p \otimes ops^j]\!]$. Furthermore, the value of n is linear in the size of p, and p' can be computed in quadratic time.*

Proof. Let $\lambda(ops!) = \{b_1, \dots, b_k\}$. There are four cases. In the first two cases the loop can be iterated an infinite number of times and the channel contents will be unbounded. In case 3 the loop can be iterated an infinite number of times but the channel contents will be bounded. In case 4 deadlock occurs after at most n iterations.

1. If $(ops?)^* \subseteq [\![p]\!]$. This means that either $ops?$ is empty or there is an atomic expression in p of the form $(a_1 + \dots + a_m)^*$ where $\lambda(ops?) \subseteq \{a_1, \dots, a_m\}$. In case $ops?$ is empty, we let $n = 0$ and $p' = p \bullet (b_1 + \cdots + b_k)^*$. Otherwise, let e be the first expression in p (starting from the left) which satisfies the above property, and let $p = p_1 \bullet e \bullet p_2$. We define $n = |p_1|$ and $p' = e \bullet p_2 \bullet (b_1 + \cdots + b_k)^*$.
   Intuitively, after consuming the words in p_1, the loop can be iterated an arbitrary number of times producing and adding to the right a corresponding number of $ops!$. Hence, due to lossiness, the global effect is obtained by concatenating to the right of $e \bullet p_2$ the downward closure of $(ops!)^*$, which is precisely $(b_1 + \cdots + b_k)^*$.
2. If $(ops?)^* \not\subseteq [\![p]\!]$, $ops? \preceq^+ ops!$, and $p \otimes ops \neq \emptyset$, then we define $n = |p|$ and $p' = (b_1 + \cdots + b_k)^*$.
   Intuitively, since $(ops?)^* \not\subseteq [\![p]\!]$, the original contents of the channel will be consumed after at most n iterations. Furthermore, $ops? \preceq^+ ops!$ implies that there is an m such that $(ops?)^{m+1} \preceq (ops!)^m$. Hence that contents of the channel will grow by at least $ops!$ after each $m + 1$ iterations. By iterating the loop sufficiently many times we can concatenate any number of copies of $ops!$ to the end of the channel. Again, by lossiness, the total effect amounts to $(b_1 + \cdots + b_k)^*$. The condition $p \otimes ops \neq \emptyset$ guarantees that the first iteration of the loop can be performed. This is to cover cases where e.g. the channel is initially empty and the receive operations are performed first in the loop.
3. If $(ops?)^* \not\subseteq [\![p]\!]$, $ops? \not\preceq^+ ops!$, $ops? \preceq_c ops!$, and $p \otimes ops^2 \neq \emptyset$, then $n = |p| + 1$ $p' = p \otimes ops^{n+1}$.
   Although the loop can be iterated any number of times, the contents of the channel will not grow after the n^{th} iteration. Observe that we demand $p \otimes ops^2 \neq \emptyset$. The condition $p \otimes ops \neq \emptyset$ (in case 2) is not sufficient here. A counter-example is $p = ba$ and $ops = (?b)(?a)(!a)(!b)$. We get $p \otimes ops = ab$ and $p \otimes ops^2 = \emptyset$. An explanation is that, for strings x and y, the relation $x \preceq^+ y$ (a condition of case 2) implies $x \preceq y$, while $x \preceq_c y$ (the corresponding condition in case 3) implies $x \preceq y^2$ but not $x \preceq y$.
4. If conditions 1, 2, or 3 are not satisfied, then $n = |p|+1$. We have $p \otimes ops^n = \emptyset$. In this case the loop can be executed at most n times, after which the channel becomes empty, and we deadlock due to inability to perform receive operations.

Notice that the proof of Lemma 7 gives us a complete characterization of whether a loop can be executed infinitely often from a certain configuration (i.e., in cases 1. - 3.), and whether in such a case the contents of channel grows unboundedly or stays finite.

Also, observe that in case we have an SRE (instead of a product) then we can apply the lemma to each product separately.

The result of Lemma 7 can be generalized to indexed SREs in a straightforward manner: The loop can be executed infinitely often if and only if the loop can be executed infinitely often with respect to each channel. If the loop can be executed infinitely often, then we take the Cartesian products of the expressions computed according to Lemma 7. This gives us the following.

Corollary 3. *For an indexed SRE R and a sequence Ops of indexed operations, there is an indexed SRE, which we denote by $R \otimes Ops^*$, such that $[\![R \otimes Ops^*]\!] = \bigcup_{0 \leq j} [\![R \otimes Ops^j]\!]$. Furthermore, $R \otimes Ops^*$ can be computed in quadratic time.*

7 Use in Verification Algorithms

The SRE representation and the operations presented in this paper can be used in on-the-fly verification algorithms for lossy channel systems. The techniques are rather standard, so here we only provide a sketch.

Suppose we want to check whether some set Γ_F of configurations is reachable. We then search through the (potentially infinite) set of reachable configurations, as follows.

We use *symbolic states* to represent sets of configurations. A *symbolic state* ϕ is a pair $\langle s, R \rangle$, where s is a control state , and R is an indexed SRE describing the contents of the channels. The language $[\![\phi]\!]$ defined by ϕ is the set of configurations $\{\langle s, w \rangle\ ;\ w \in [\![R]\!]\}$. We extend the entailment relation in the obvious way so that $\langle s, R \rangle \sqsubseteq \langle s', R' \rangle$ if and only if $s = s'$ and $R \sqsubseteq R'$.

We maintain a set V which we use to store symbolic states which are generated during the search. At the start, the set V contains one unexplored symbolic state representing the initial configuration. From each unexplored element in V, we compute two sets of new elements: one which corresponds to performing single transitions (Lemma 6), and another which describes the effect of a selected set of control loops. When a new element ϕ is generated, it is compared with those which are already in V. If $\phi \sqsubseteq \phi'$ for some $\phi' \in V$, then ϕ is discarded (it will not add new configurations to the searched state space). It is also checked whether ϕ has a non-empty intersection with Γ_F. This is easy if e.g., Γ_F is a recognizable set. If the intersection is non-empty, the algorithm terminates. Otherwise, the algorithm is terminated when no new symbolic states can be generated.

When performing control loops during the analysis, there is a choice in how many loops to explore. A reasonable strategy seems to be to investigate the sequences of transitions which correspond to *simple control loops* in the program. A simple control loop is a loop which enters each control state at most once. By applying these control loops we get new symbolic states which can be computed according to Corollary 3.

During our search, it can happen that a new element ϕ is added to V, although ϕ will not add any new configurations to the explored state space. This is due to the fact that even if $\phi \not\sqsubseteq \phi'$ for all $\phi' \in V$, the relation $[\![\phi]\!] \subseteq \bigcup_{\phi' \in V} [\![\phi']\!]$ may still hold. The test for discarding new SREs can therefore be modified so that ϕ is discarded if and only if $[\![\phi]\!] \subseteq \bigcup_{\phi' \in V} [\![\phi']\!]$. This would make the algorithm terminate more often (fewer elements need to be added to V). However, for indexed SREs (and hence for symbolic states), the above test has an exponential complexity in the number of channels.

From Theorem 2, we know that our algorithm is incomplete. The algorithm will always find reachable configurations in Γ_F, but it will not necessarily terminate if all configurations in Γ_F are unreachable.

In fact, we can use a slight extension of this procedure to check whether a lossy channel system satisfies a linear temporal logic formula over the control states of the system. By standard techniques [21], we can transform this problem into checking whether a lossy channel system, in which some control states are designated as "accepting", has an infinite computation which visits some accepting control state infinitely often. In our earlier work [1], we showed that this problem is undecidable. However, an incomplete check can be performed as part of the state-space generation in the previous paragraph. More precisely, when exploring a set of configurations with an accepting control state we can, as part of exploring the loops, check whether there is a control loop that can be executed an infinite number of times. We only need to check whether one of the three first conditions in the proof of Lemma 7 holds.

8 Example

In this section we apply our algorithm (Table 1) to a sliding window protocol (shown in Figure 1). We use a symbolic representation of the form $\langle s_i, q_j, r_1, r_2 \rangle$, where s_i and q_j are the control states of the sender and the receiver, respectively, and r_1 and r_2 are SREs which describe the contents of the message and acknowledgement channels. We explore the state space as described in the preceding section, investigating the effect of simple control loops in the program.

In Figure 1, we start from $\langle s_1, q_1, \epsilon, \epsilon \rangle$ and apply the speed-up operation obtaining ϕ_0. From ϕ_0 we perform a single transition moving from q_1 to q_2, and then perform the speed-up operation obtaining ϕ_1. In a similar manner we obtain ϕ_2 and ϕ_3 from ϕ_1, etc. Observe that, e.g. ϕ_5 entails ϕ_7, so ϕ_5 is discarded.

9 Conclusions

We present a method for performing symbolic forward reachability analysis of *unbounded lossy channel systems.* In spite of the restriction of lossiness, we can model the behaviour of many interesting systems such as link protocols which are designed to operate correctly even in the case where the channels are lossy and can lose messages. Also lossy channel systems offer conservative approximations when checking linear time properties of systems with *perfect* channels. This is because the set of computations of a lossy channel system is a superset of the set of computations of the corresponding system with perfect channels, and hence if a linear time property holds in the first it will also hold in the second.

In this paper, we accelerate the forward search of the state space, by considering (besides single transitions) the effect of "meta-transitions" which are simple loops entering each control state at most once. We intend to investigate more

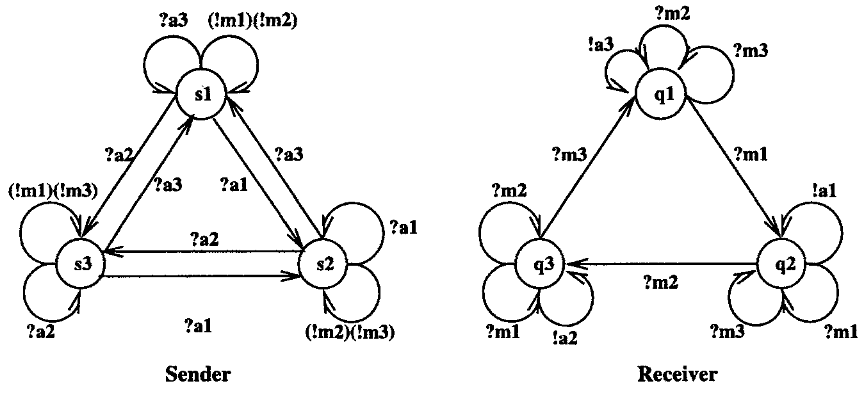

Fig. 1. Example: A Sliding Window Protocol

general types of meta-transitions. For example consider the case where we have two loops sending two different messages (say a_1 and a_2) to the same channel. In the algorithm we propose we cannot cover the fact that the combination of the two loops would give the expression $(a_1 + a_2)^*$ in the channel. We are currently carrying out experiments to evaluate the performance of our algorithm. It would be particularly interesting to compare the forward reachability algorithm we present here with the performance of the backward reachability algorithm reported in [2].

References

[1] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Inform. and Comput.*, 130(1):71–90, 1996.

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

[3] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *CAV'96*, LNCS 1102.

[4] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. Available at `http://www.montefiore.ulg.ac.be/~biogelot/research/BGWW97.ps`.

[5] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS'97*, LNCS. 1997.

[6] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. http://www.imag.fr/VERIMAG/PEOPLE/Peter.Habermehl.

[7] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *ICALP '97*, LNCS 1256. 1997.

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

[9] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*, LNCS 818. 1994.

ϕ_0	$\langle s_1, q_1 \ , \ (m_1 + m_2)^* \ , \ (a_3)^* \rangle$	ϕ_1
ϕ_1	$\langle s_1, q_2 \ , \ (m_1 + m_2)^* \ , \ (a_3)^* \bullet (a_1)^* \rangle$	ϕ_2, ϕ_3
ϕ_2	$\langle s_2, q_2 \ , \ (m_1 + m_2)^* \bullet (m_2 + m_3)^* \ , \ (a_1)^* \rangle$	ϕ_4
ϕ_3	$\langle s_1, q_3 \ , \ (m_1 + m_2)^* \ , \ (a_3)^* \bullet (a_1)^* \bullet (a_2)^* \rangle$	ϕ_4, ϕ_5
ϕ_4	$\langle s_2, q_3 \ , \ (m_1 + m_2)^* \bullet (m_2 + m_3)^* \ , \ (a_1)^* \bullet (a_2)^* \rangle$	ϕ_6, ϕ_7
ϕ_5	$\langle s_3, q_3 \ , \ (m_1 + m_2)^* \bullet (m_1 + m_3)^* \ , \ (a_2)^* \rangle$	Ent ϕ_7
ϕ_6	$\langle s_2, q_1 \ , \ (m_2 + m_3)^* \ , \ (a_1)^* \bullet (a_2)^* \bullet (a_3)^* \rangle$	ϕ_8, ϕ_9
ϕ_7	$\langle s_3, q_3 \ , \ (m_1 + m_2)^* \bullet (m_2 + m_3)^* \bullet (m_1 + m_3)^* \ , \ (a_2)^* \rangle$	ϕ_8
ϕ_8	$\langle s_3, q_1 \ , \ (m_2 + m_3)^* \bullet (m_1 + m_3)^* \ , \ (a_2)^* \bullet (a_3)^* \rangle$	ϕ_{10}, ϕ_{11}
ϕ_9	$\langle s_1, q_1 \ , \ (m_2 + m_3)^* \bullet (m_1 + m_2)^* \ , \ (a_3)^* \rangle$	Ent ϕ_{11}
ϕ_{10}	$\langle s_3, q_2 \ , \ (m_1 + m_3)^* \ , \ (a_2)^* \bullet (a_3)^* \bullet (a_1)^* \rangle$	ϕ_{12}, ϕ_{13}
ϕ_{11}	$\langle s_1, q_1 \ , \ (m_2 + m_3)^* \bullet (m_1 + m_3)^* \bullet (m_1 + m_2)^* \ , \ (a_3)^* \rangle$	ϕ_{12}
ϕ_{12}	$\langle s_1, q_2 \ , \ (m_1 + m_3)^* \bullet (m_1 + m_2)^* \ , \ (a_3)^* \bullet (a_1)^* \rangle$	ϕ_3, ϕ_{14}
ϕ_{13}	$\langle s_2, q_2 \ , \ (m_1 + m_3)^* \bullet (m_2 + m_3)^* \ , \ (a_1)^* \rangle$	Ent ϕ_{14}
ϕ_{14}	$\langle s_2, q_2 \ , \ (m_1 + m_3)^* \bullet (m_1 + m_2)^* \bullet (m_2 + m_3)^* \ , \ (a_1)^* \rangle$	ϕ_4

Table 1. Reachability Analysis of the Sliding Window Protocol

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

[11] A. Choquet and A. Finkel. Simulation of linear FIFO nets having a structured set of terminal markings. In *Proc. 8th European Workshop on Applications and Theory of Petri Nets*, 1987.

[12] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inform. and Comput.*, 124(1):20–31, 10 January 1996.

[13] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *CAV'90*.

[14] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Technical report, LIFAC, ENS de Cachan, 1996. Tech. Rep.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] A.P. Sistla and L.D. Zuck. Automatic temporal verification of buffer systems. In Larsen and Skou, editors, *CAV'91*, LNCS 575. 1991.

[21] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS'86*, IEEE, 1986.