Using Forward Reachability Analysis for Verification of Lossy Channel Systems

Parosh Aziz Abdulla
Dept. of Information Technology
Uppsala University
Sweden

Aurore Annichini
VERIMAG
Grenoble
France

Ahmed Bouajjani
LIAFA
University of Paris 7
France

Bengt Jonsson
Dept. of Information Technology
Uppsala University
Sweden

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 computing (i) 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 straightforwardly construct an algorithm which explores the set of reachable states of a protocol, in order to check various safety properties. We also show how one can perform model-checking of LTL properties, using a standard automata-theoretic construction. It should be noted that all these methods are by necessity incomplete, even for the class of protocols with lossy channels.

To illustrate the applicability of our methods, we have developed a tool prototype and used the tool for automatic verification of (a parameterized version of) the Bounded Retransmission Protocol.

1 Introduction

One of the most popular models for specifying and verifying communication protocols is that of *Communicating Finite State Machines (CFSM)* [BZ83, Boc78]. This model consists of finite-state processes that exchange messages via unbounded FIFO queues. Several verification methods have been developed for CFSMs [BZ83, CF87, GGLR87, Pac87, PP91, SZ91]. However, since all interesting verification problems are undecidable [BZ83], 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 [AJ96b], 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 performs a backward search, starting from a set of "bad" states and trying to reach some initial state. In contrast, many efficient verification algorithms [Hol91, CVWY90], perform a forward search, starting from the initial states.

While the two methods seem to be symmetric, they exhibit surprisingly different behaviours in many applications. For instance, for several classes of infinite-state systems, it turns out that backward analysis is feasible while forward analysis is not [AJ96b, May00]. Nevertheless, forward analysis is still practically very attractive from the point of view of model checking. Forward closure contains much more information about system behaviour than backward closure. This is due to the fact that forward closure characterizes the set of states which may arise during the execution of the system, while backward closure only describes the set from which the system can fail. More interestingly, forward analysis can often be used for constructing *symbolic graphs* which constitute finite abstractions of the systems, and on which standard techniques for finite state model checking can be applied (e.g. the BRP protocol reported 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 [BW94] and *perfect* channel systems [BG96, BGWW97]. 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 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 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 [CFI96] saying that there is no general algorithm for generating the set of reachable states.

To illustrate the applicability of our approach, we have implemented our techniques in the TReX tool [ABS01]. Given a lossy channel system, the tool generates automatically the set of reachable configurations described as SREs, and produces a *symbolic graph* which constitutes a finite-state abstract model of the system. Furthermore, the tool allows on-the-fly verification of safety properties given by finite-state labelled transition systems. The TReX tool is connected to the Cadp toolbox [FGK+96] which provides a variety of procedures on finite-states labelled transition systems, e.g., comparison and minimization w.r.t. behavioural equivalences, and model-checking for temporal logics. For instance, it is possible to generate automatically a finite abstract model of a system using TReX, and then apply standard finite-state verification techniques on the abstract model.

As an experiment, we have applied the tool for automatic verification of the Bounded Retransmission Protocol (BRP) of Phillips [GvdP93]. The BRP is a data link protocol which can be seen as an extended version of the well known alternating bit protocol. It consists of a sender and a receiver that communicate through two unbounded lossy channels. The service provided by the protocol is to transmit large files, where each file is a sequence of data of some arbitrary length. In addition, both the sender and receiver must indicate to their clients whether the whole file has been delivered successfully or not. The sender reads a sequence of data and transmits successively each datum in a separate frame following an alternating bit protocol-like procedure. However, the sender can resend a non-acknowledged frame up to a fixed number of retransmissions MAX, which is a parameter of the protocol. In our model, we assume that the value of MAX and the sizes of the transmitted sequences are arbitrary positive integers. The assumption concerning MAX leads to a model with unbounded channels representing a *family* of BRPs. Each member of the family operates on a certain given value of MAX. In other words, we use the model of unbounded channels to perform *parametric reasoning* on an infinite family of systems.

The TReX tool generates automatically the set of reachable configurations of the BRP and the corresponding finite symbolic graph (0.35 seconds/6.0

Mb on a Sun4 UltraSparc/SunOS ubac 5.8/Sun-Blade-100/1.5Gb). After projecting this graph on the set of external actions of the protocol and minimizing it w.r.t. observational trace equivalence, we get an abstract model with 5 states and 10 transitions which corresponds exactly to the expected external behaviour of the protocol.

Related Work There are several existing works on symbolic verification of *perfect* channel systems. Pachl [Pac87] 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 [FM96] 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 et al. [BG96, BGWW97] use finite-state automata, the so-called QDDs, to represent sets of channel contents. QDDs allow to represent recognizable sets of vectors of sequences. In [BGWW97] it has been shown that the effect of every loop is recognizable for a system with a *single* fifo-channel (recognizable means regular in this case). 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 [BGWW97] 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, representation structures called CQDDs (constrained QDDs), combining a *subclass* of finite automata with linear arithmetical constraints, are needed [BH97]. In the case of lossy channels, the links between the number of occurrences in different channels are broken due to lossiness. In this paper, we take advantage from this fact and show that in the case of lossy channels, computing the effect of iterating loops is simpler and more efficient than in the case of perfect channels.

We argue that SREs offer several advantages when used as a symbolic representation in the context of lossy channel systems. First, since QDDs and CQDD use deterministic automata, each operation on these representations requires a determinization step which is exponential in general, whereas all the needed operations on SREs can be performed in polynomial time.

Also, a nice feature of SREs is that they admit a canonical form which can be computed in quadratic time. While QDDs admit a canonical form via determinization and minimization, a corresponding result is not known for CQDDs. Moreover, the algorithms for computing iterations of loops for perfect channels using QDDs or CQDDs are quite nontrivial (see e.g. [BGWW, BH]) and hard to implement, especially those using CQDDs, whereas our algorithm for the analysis of lossy channels using SREs is much more simpler and easier to implement.

Finally, we observe that the algorithms in [BG96, BGWW97, BH97] based on computing iterations of control loops (using QDDs and CQDDs) in perfect channel systems cannot simulate the algorithm we present in this paper which is based on iterating control loops in the model of lossy channel systems. Indeed, while losses are implicit in our model, they have to be explicitly modeled if we adopt the standard model of FIFO-channel systems. This modeling introduces new control loops which have to be taken into the account *in combination* with the "original" control loop to analyze (sequence of operations not including explicit losses). So, a control loop in the lossy model would correspond to a combination of loops (a connected component) in the standard model since at each step of the considered sequence of operations, the system can choose nondeterministically either to perform the next operation in the sequence, or to simulate first the loss of some messages in the channels before continuing the execution of the sequence. However, the analysis of combined (or nested) loops is not feasible in the approaches of [BG96, BGWW97, BH97] since they only allow to reason about single loops.

Another way to see the fact that the algorithms of [BG96, BGWW97, BH97] cannot simulate the one of this paper is the following: the algorithm of [BH97] generalizes the ones in [BGWW97] by using representations combining a fragment of recognizable sets and arithmetical constraints. Actually, the algorithms of [BGWW97] do not use the full power of QDDs, but only the fragment of recognizable sets which are CQDD definable. This fragment corresponds to finite unions of finite products of languages definable by regular expressions of the form $e_1 + \cdots + e_m$ where each e_i an expression of the form $u_1^* v_1 u_2^* v_2 u_3^* \cdots v_{n-1}(u_n^* | A^*)$ the u_i's and the v_i's being words and A being a set of symbols. It is easy to see that this fragment is incomparable with the SRE definable sets (which are precisely the downward closed sets).

Several works have addressed the specification and verification of the BRP. To tackle the problem of unboundedness of the size of the transmitted files and the parameter MAX, these works propose proof-based approaches using theorem provers, combined with abstraction techniques and model checking.

In [GvdP93] the system and its external specification are described in μCRL and are proved to be (branching) bisimilar. The proof is carried out by hand and is checked using Coq. An approach based on proving trace inclusion (instead of bisimulation) on I/O automata is developed in [HSV94]. In [HS96] the theorem prover PVS is used to prove that the verification of the BRP can be reduced by means of abstraction to a finite-state problem that can be solved by model checking. In [GS97, BLO98] a more automated approach is applied based on constructing automatically a *finite* abstract model using PVS, for an *explicitly given* abstraction function.

Another way to look at our model is to consider the lossy channel system as an *abstraction* of an infinite family of the BRPs; namely the family of BRPs with all possible values of the two parameters: file sizes and value of MAX. The model is *infinite-state*: the unboundedness of the parameters is in some sense transformed into an unboundedness of the channels. Starting from this infinite-state system, our verification technique is fully automatic. It is based on an automatic generation of a finite abstract model, without giving explicitly the abstraction relation. So, our work provides a fully automatic (and efficient) verification of the (untimed) *parameterized* version of the BRP.

Finally, we mention two works where the BRP has been verified automatically but *only for some fixed instances of its parameters*. In [Mat96], an untimed version of the BRP is verified using both a bisimulation-based approach and a model checking approach using CADP. In [DKRT97] a timed version of the BRP is verified using the tools SPIN and UPPAAL. These works use standard finite-state techniques which cannot handle parametric/infinite-state models. The work in [DKRT97] consider timing aspects that we have abstracted here since our model is untimed.

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 different verification algorithms for lossy channel systems. In Section 8 we describe our tool prototype. In Section 9 we present our modeling and verification of the BRP. Finally, in Section 10 we give some conclusions.

2 Lossy Channel Systems

We consider system models consisting of asynchronous parallel compositions of finite-state machines that communicate through sending and receiving messages via a finite set of unbounded *lossy* fifo channels (in the sense that they can nondeterministically lose messages).

A *Lossy Channel System* (LCS) $\mathcal{L}$ is a tuple $(S, s_{init}, C, M, \Sigma, \delta)$, 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 $\langle s_{init_1}, \ldots, s_{init_n} \rangle$ of initial states of the components.
- C is a finite set of *channels*.
- M is a finite set of *messages*.
- Σ is a finite set of *transition (or action) labels*.
- δ is a finite set of *transitions*, each of which is of the form (s_1, ℓ, Op, s_2), where s_1 and s_2 are states, $\ell \in \Sigma$, 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 $a \in M$.

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 $\preceq$ to denote the subsequence relation on M^*, i.e., $x \preceq y$ denotes that x is a (not necessarily contiguous) substring of y.

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^* giving the contents of each channel. The *initial configuration* γ_{init} of $\mathcal{L}$ is the pair $\langle s_{init}, \epsilon \rangle$ where ϵ denotes the mapping where each channel is assigned the empty sequence ϵ.

We define a labelled transition relation on configurations in the following manner: $\langle s_1, w_1 \rangle \xrightarrow{\ell} \langle s_2, w_2 \rangle$ if and only if there exists a transition $(s_1, \ell, Op, s_2) \in \delta$ such that, for each $c \in C$, 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 $w_2(c) = w_1(c)$.

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 $c \in C$. Then, we introduce a *weak* transition relation on configurations in the following manner: $\langle s_1, w_1 \rangle \stackrel{\ell}{\Longrightarrow} \langle s_2, w_2 \rangle$ if and only if there are w_1' and w_2' such that $w_1' \preceq w_1$, $w_2 \preceq w_2'$, and $\langle s_1, w_1' \rangle \stackrel{\ell}{\longrightarrow} \langle s_2, w_2' \rangle$. Intuitively, $\langle s_1, w_1 \rangle \stackrel{\ell}{\Longrightarrow} \langle s_2, w_2 \rangle$ means that $\langle s_2, w_2 \rangle$ can be reached from $\langle s_1, w_1 \rangle$ by first losing messages from the channels and reaching $\langle s_1, w_1' \rangle$, then performing the transition $\langle s_1, w_1' \rangle \stackrel{\ell}{\longrightarrow} \langle s_2, w_2' \rangle$, and thereafter losing messages from channels.

Given a configuration γ, we let $post(\gamma)$ denote the set of immediate successors of γ, i.e., $post(\gamma) = \{\gamma' \ : \ \exists \ell \in \Sigma.\ \gamma \stackrel{\ell}{\Longrightarrow} \gamma'\}$. The function $post$ is generalized to sets of configurations in the obvious manner. We let $post^*$ denote the reflexive transitive closure of $post$, i.e., given a set of configurations Γ, $post^*(\Gamma)$ is the set of all reachable configurations starting from Γ. Let $Reach(\mathcal{L})$ be the set $post^*(\gamma_{init})$. For every control location $s \in S$, we define $\mathcal{R}(s) = \{w \ : \ \langle s, w \rangle \in Reach(\mathcal{L})\}$.

A *run* of $\mathcal{L}$ starting from a configuration γ is a finite or infinite sequence $\rho = \gamma_0 \ell_0 \gamma_1 \ell_1 \gamma_2 \ldots$ such that $\gamma_0 = \gamma$ and $\forall i \geq 0.\ \gamma_i \stackrel{\ell_i}{\Longrightarrow} \gamma_{i+1}$. The *trace* of the run ρ is the sequence of action labels $\tau = \ell_0 \ell_1 \ell_2 \ldots$. We denote by $Traces(\mathcal{L})$ the set of traces of all runs of $\mathcal{L}$ starting from the initial configuration γ_{init}.

We introduce two extensions of the basic model given above: the first one consists in introducing *channel emptiness testing*: we use enabling conditions on transitions involving a predicate *empty* on channels telling whether a channel is empty. The second extension consists in allowing the components of a system to test and set *boolean shared variables* (remember that we consider here asynchronous parallel composition following the interleaving semantics). The formal semantics of the extended model is an obvious adaptation of the one given above.

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 3.1 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 $x \in K(c)$ if and only if $x \in [\![R(c)]\!]$. 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.

A *C-indexed language* K induces a language $K^\bullet \subseteq 2^{C \mapsto M^*}$ such that $w \in K^\bullet$ if and only if $w(c) \in K(c)$ for each $c \in C$. We say that $K^\bullet$ is *simply recognizable* in case K is simply recognizable. □

Definition 3.2 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 3.3 For finite alphabets M and C and a C-indexed language L over M, if L is downward-closed then L is simply recognizable.

Proof. The proof can be found in the appendix. □

Since the set $\mathcal{R}(s)$ is downward-closed, we get the following.

Corollary 3.4 For a lossy channel system $\mathcal{L}$ and a state s in $\mathcal{L}$, the set $\mathcal{R}(s)$ is simply recognizable.

However, in [May00] the following is shown:

Theorem 3.5 **[May00]** For a lossy channel system $\mathcal{L}$ and a state s in $\mathcal{L}$, there is, in general, no algorithm for computing a C-indexed SRE characterizing $\mathcal{R}(s)$.

In other words, although we can compute a representation of the set of configurations from which a given configuration is reachable ([AJ96b]), we cannot in general compute a representation of the set of configuration which are reachable from a given configuration (Theorem 3.5). 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 4.1 For products $p, p_1, \ldots, p_n$, if $p \sqsubseteq p_1 + \ldots + p_n$ then $p \sqsubseteq p_i$ for some $i \in \{1 \ldots n\}$.

Proof. The proof can be found in the appendix. □

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

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

Lemma 4.2 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 $p_1 \sqsubseteq e_2 \bullet p_2$.

$\square$

Lemma 4.3 Entailment among SREs can be checked in quadratic time.

Proof. The proof follows from Lemma 4.1 and Lemma 4.2. $\square$

Corollary 4.4 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 5.1 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 $e_i \bullet e_{i+1} \not\sqsubseteq e_i$. $\square$

Lemma 5.2 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 5.1. $\square$

Similarly, we can define a normal form for SREs.

Definition 5.3 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 $i, j : 1 \leq i \neq j \leq n$. $\square$

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

Lemma 5.4 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 4.2, Lemma 4.1 and Lemma 5.2. $\square$

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 $\mathcal{L}$, and a set Γ of configurations in $\mathcal{L}$, the set $post(\Gamma)$ is given by $\bigcup_{(s_1,\ell,Op,s_2)\in\delta}\{\langle s_2, w\rangle \;:\; w \in (\Gamma \otimes Op)(s_1)\}$.

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

Lemma 6.1 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

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

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 6.1 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 6.4, 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 6.5 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. If $\epsilon \prec x, y$, then we use $x \preceq^+ y$ to denote that there is an integer $m \geq 1$ such that $x^m \preceq y^{m-1}$.

Lemma 6.2 For strings x, y, if $x \preceq^+ y$ then $x^m \preceq y^{m-1}$, for some $m : 1 \leq m \leq |y|$.

Proof. The proof can be found in the appendix. □

Corollary 6.3 For strings x, y, $x \preceq^+ y$ if and only if $x^{|y|} \preceq y^{|y|-1}$.

Proof. The *if*-direction follows trivially by definition. For the other direction, we observe that $x \preceq^+ y$ implies $x \preceq y$. Also, if $x \preceq y$ and $x^m \preceq y^{m-1}$ for any $m : 1 \leq m \leq |y|$ then $x^{|y|} \preceq y^{|y|-1}$. The result follows directly from Lemma6.2. □

From Corollary 6.3 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 6.4 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 $\lambda(ops!) = \{b_1, \ldots, b_k\}$. 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 $(a_1 + \ldots + a_m)^*$ where $\lambda(ops?) \subseteq \{a_1, \ldots, a_m\}$. 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, according to Lemma 6.2, $ops? \preceq^+ ops!$ implies that there is an $m : 1 \leq m \leq |ops!|$ such that $(ops?)^m \preceq (ops!)^{m-1}$. Hence that contents of the channel will grow by at least $ops!$ after each m 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 6.4 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 6.4 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 6.4. This gives us the following.

Corollary 6.5 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 $[\![R \otimes Ops^*]\!] = \cup_{0 \leq j} [\![R \otimes Ops^j]\!]$. Furthermore, $R \otimes Ops^*$ can be computed in quadratic time.

7 Use in Verification Algorithms

The SRE representation and the operations presented in the previous sections can be used in solving verification problems for lossy channel systems. First, we can use these operations in on-the-fly verification procedures where properties are checked during the generation of the set of reachable configurations. Another approach is to use reachability analysis to construct a finite abstract model of the system, which can be handled by means of standard finite-state techniques.

7.1 On-the-Fly Verification

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 6.1), and another which describes the effect of of control loops. Here, there is a choice in which 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 6.5. Actually, these loops can be detected automatically during the search.

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.

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 3.5, 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.

The procedure described can be used for checking safety properties since their verification problem is straightforwardly reducible to a reachability problem. 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 [VW86], 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 [AJ96a], 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 6.4 holds.

7.2 Generation of Finite Abstractions

Let ρ be a binary relation on configurations. We say that ρ is a *simulation* if for every pair of configurations γ_1 and γ_2, and every label $\ell \in \Sigma$, if $(\gamma_1, \gamma_2) \in \rho$ and there is a configuration γ_1' such that $\gamma_1 \xrightarrow{\ell} \gamma_1'$, then there exists γ_2' such that $\gamma_2 \xrightarrow{\ell} \gamma_2'$ and $(\gamma_1', \gamma_2') \in \rho$.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two systems and let γ_{init}^1 and γ_{init}^2 be their respective initial configurations. Then, we say that $\mathcal{L}_2$ *simulates* $\mathcal{L}_1$ if there is a simulation relation ρ between the configurations of $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $(\gamma_{init}^1, \gamma_{init}^2) \in \rho$.

It is well known that if $\mathcal{L}_2$ simulates $\mathcal{L}_1$, then $Traces(\mathcal{L}_1) \subseteq Traces(\mathcal{L}_2)$.

Now, we introduce the notion of a *symbolic graph*. Given a lossy channel system $\mathcal{L}$, we can use reachability analysis to construct a finite symbolic graph which simulates $\mathcal{L}$.

Let Φ be a finite set of symbolic states. Then, the *symbolic graph* associated with Φ is the finite-state labelled transition system $\mathcal{G}_\Phi$ such that the set of states is Φ and, $\forall \phi_1, \phi_2 \in \Phi.\ \forall \ell \in \Sigma.\ \phi_1 \xrightarrow{\ell} \phi_2$ iff $\exists \gamma_1 \in \phi_1, \gamma_2 \in \phi_2.\ \gamma_1 \xrightarrow{\ell} \gamma_2$. We consider as initial state in $\mathcal{G}_\Phi$ any configuration which contains the initial configuration γ_{init}.

In particular, we consider the partition of $Reach(\mathcal{L})$ according to the control states, i.e., $\Phi_{\mathcal{L}} = \{\langle s, R\rangle \ : \ s \in S \text{ and } [\![R]\!] = \mathcal{R}(s)\}$. The labelled transition system $\mathcal{G}_{\Phi_{\mathcal{L}}}$ is called the *canonical symbolic graph* of $\mathcal{L}$.

Lemma 7.1 For every finite set of symbolic states Φ, if $Reach(\mathcal{L}) \subseteq \bigcup_{\phi\in\Phi} [\![\phi]\!]$, then $\mathcal{G}_{\Phi}$ simulates $\mathcal{L}$.

Indeed, it is easy to see that the membership relation, i.e., the relation ρ such that $\gamma\rho\phi$ iff $\gamma \in [\![\phi]\!]$, is a simulation relation (using the fact that every reachable configuration of $\mathcal{L}$ belongs to at least one symbolic state in Φ).

Clearly, Lemma 7.1 holds for the canonical symbolic graph of $\mathcal{L}$. This means that if $Reach(\mathcal{L})$ can be constructed, we obtain directly a *finite-state abstraction* of the system $\mathcal{L}$. This abstract model can be used to check linear-time properties and, if the result is positive, to deduce that the same result holds for the *concrete* system $\mathcal{L}$[1]. More precisely, given an ∞-regular linear-time property Π, i.e., a set of finite or infinite traces over Σ, a system $\mathcal{L}$ satisfies Π if $Traces(\mathcal{L}) \subseteq \Pi$. By Lemma 7.1, we have $Traces(\mathcal{L}) \subseteq Traces(\mathcal{G}_{\Phi_{\mathcal{L}}})$. Hence, for every ∞-regular property Π, if $\mathcal{G}_{\Phi_{\mathcal{L}}}$ satisfies Π, then $\mathcal{L}$ satisfies Π too.

Notice that if $\mathcal{G}_{\Phi_{\mathcal{L}}}$ does not satisfy Π, this could be due to the fact that the abstraction corresponding to the partition of $Reach(\mathcal{L})$ according to the control states is too coarse. Then, one could try to check Π on refinements of this partition.

8 Implementation in TReX

We implemented our techniques in a the tool TReX. The input of the TReX is a finite set of communicating automata, given separately. Then, the tool allows the following options:

- **Generation of the reachability set:** The tool allows calling a procedure which computes a representation of the reachability set of the system by means of (normal) SREs. The computation is done according to a depth-first-search strategy, and uses the acceleration principle as described in Sections 6 and 7. Notice that the loops used for acceleration are found on-the-fly and are not given explicitly by the user.

[1]This approach can also be applied for branching-time properties expressed in universal positive fragments of temporal logics or μ-calculi like $\forall$CTL* [GL91] or $\Box L_\mu$ [BBLS92].

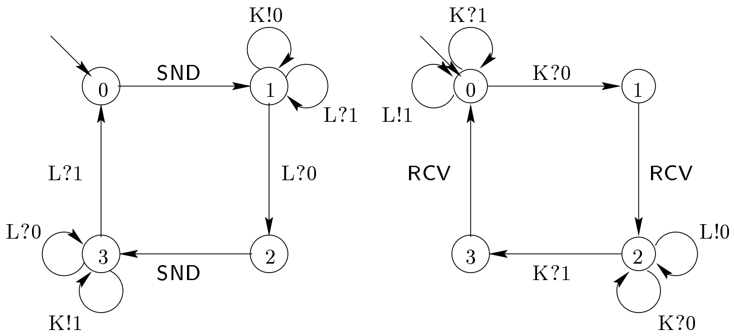


Figure 1: Alternating Bit Protocol

- **On-the-fly checking of safety properties:** Given a safety property described as a deterministic labelled transition system Π over a set *observable actions* $\Omega \subseteq \Sigma$, the tool checks whether the projection of the system on Ω (i.e., the system obtained after hiding all actions except those in Ω) satisfies the property Π. This verification is done on-the-fly (the procedure stops as soon as the property is falsified) following the principle described in Section 7.

- **Generation of the canonical symbolic graph:** During the computation the reachability set, TReX can construct the corresponding canonical symbolic graph (transitions between symbolic states).

  The symbolic graph is produced in the input format of the Cadp toolbox (Caesar/Aldebaran Development Package) [FGK+96] which contains several tools on finite-state labelled transition systems, e.g., graphical visualization, comparison with respect to various behavioural equivalences and preorders like observational bisimulation and simulation, minimization, on-the-fly automata-based verification, model-checking for an ACTL-like temporal logic (action-based variant of CTL) and the alternation-free modal μ-calculus.

Example: Alternating Bit Protocol

Let us illustrate the use of our tool on the Alternating Bit Protocol (ABP for short). We model the ABP by two finite-state machines, a sender and a receiver, communicating through two unbounded lossy channels K and L (see Figure 1). The procedure implemented in TReX terminates and generates automatically the reachability set of the ABP (see Table 1), as well as the

Sender	Receiver	Channel K	Channel L
0	0	1*	1*
1	0	1*0*	1*
1	1	0*	1*
1	2	0*	1*0*
2	2	0*	0*
3	2	0*1*	0*
3	3	1*	0*
3	0	1*	0*1*

Table 1: Reachability set of the ABP

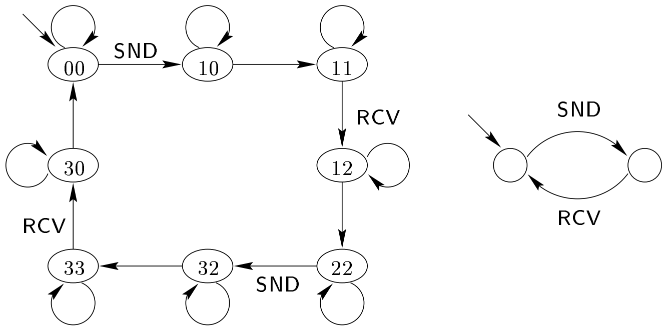


Figure 2: Canonical symbolic graph of the ABP and its minimal version

corresponding canonical symbolic graph (see Figure 2) which has 8 symbolic states and 16 transitions. The execution time is 0.04 seconds and the used memory is 4.1 Mb on a Sun4 UltraSparc/SunOS ubac 5.8/Sun-Blade-100/1.5 Gb. Then, using the ALDEBARAN tool [FM91], we minimize this graph according to observational trace equivalence by considering that SND and RCV are the only observable actions. The resulting minimal transition system is shown in Figure 2. It is clear from this transition system that the external behaviour of the ABP is equivalent to the behaviour of a one-place buffer.

Remark The minimization we considered removes all silent (non-observable) transitions, including silent loops. This can be done since we are only interested in safety properties on observable actions. This is, however, not sound for checking liveness properties (silent loops leading to divergence must be taken into account in this case).

9 The Bounded Retransmission Protocol

In this section we describe the verification of the Bounded Retransmission Protocol using the TReX tool.

9.1 Specification of the Service

The Bounded Retransmission Protocol (BRP for short) is a data link protocol. The service it delivers is to transmit large files (sequences of data of arbitrary lengths) from one client to another one. Each datum is transferred in a separate frame. Both clients, the sender and the receiver, obtain an indication whether the whole file has been delivered successfully or not.

More precisely, at the sender side, the protocol requests a sequence of data $s = d_1, \ldots, d_n$ (action REQ) and communicates a *confirmation* which can be SOK, SNOK, or SDNK. The confirmation SOK means that the file has been transferred successfully, SNOK means that the file has not been transferred completely, and SDNK means that the file may not have been transferred completely. This occurs when the last datum d_n is sent but not acknowledged. Now, at the receiver side, the protocol delivers each correctly received datum with an indication which can be RFST, RINC, or ROK. The indication RFST means that the delivered datum is the first one and more data will follow, RINC means that the datum is an intermediate one, and ROK means that this was the last datum and the file is completed. However, when the connection with the sender is broken, an indication RNOK is delivered (without datum). Properties the service must satisfy are:

1. a request REQ must be followed by a confirmation (SOK, SNOK, or SDNK) before the next request,
2. a RFST indication (delivery of the first datum) must be followed by one of the two indications ROK or RNOK before the beginning of a new transmission (next request of the sender),
3. a SOK confirmation must be preceded by a ROK indication,
4. a ROK indication can be followed by either a SOK or a SDNK confirmation, but never by a SNOK (before next request),
5. a RNOK indication must be preceded by SNOK or SDNK (abortion),

6. if the first datum has been received (with the RFST indication), then a SNOK confirmation is followed by a RNOK indication before the next request.

9.2 Description of the Protocol

The BRP consists of two processes, the sender S and the receiver R, that communicate through two unbounded lossy fifo channels K and L: messages can either be lost or arrive in the same order in which they are sent. The BRP can be seen as an extended version of the alternating bit protocol. Messages sent from the sender S to the receiver R through the channel K are frames of the form $(first, last, toggle, datum)$ where a datum is accompanied by three bits: $first$ and $last$ indicate whether the datum is the first or the last one of the considered file, $toggle$ is the alternating bit allowing to detect duplications of intermediate frames. As for the acknowledgments (sent from R to S through L), they are frames of the form $(first, last, toggle)$. Notice that in the description we consider of the BRP, the value of $toggle$ is relevant only for intermediary frames. Indeed, the first and last frames can be distinguished from the intermediary ones using the booleans $first$ and $last$.

The behaviours of S and R are the following: The sender S starts by reading (action REQ) a sequence $s = d_1, \ldots, d_n$. We consider here that $n \geq 2$, the case $n = 1$ does not introduce any difficulty. Then, S sends to R through K the first data frame $(1, 0, 0, d_1)$, and waits for the acknowledgment. Let us consider first the ideal case where frames are never lost. When R receives the frame from K, it delivers to its client the datum d_1 with the indication RFST, and sends to S an acknowledgment frame $(1, 0, 0)$ through the channel L. When S receives this acknowledgment, it transmits to R the second frame $(0, 0, 0, d_2)$ ($toggle$ is still equal to 0 since its value is relevant for intermediate frames). Then, after reception, R delivers d_2 with the indication RINC and sends the acknowledgment $(0, 0, 0)$ to S. Then, the next frame sent by S is $(0, 0, 1, d_3)$ (now $toggle$ has flipped), and the same procedure is repeated until the last frame $(0, 1, -, d_n)$ is sent (here again, like in the case of the first frame, the value of $toggle$ is not relevant). When R receives the last frame, it delivers d_n with the indication ROK, and acknowledges receipt. Then, the sender S communicates to its client the confirmation SOK meaning that the whole sequence s has been successfully transmitted.

Now, let us consider the case where frames are lost. When S send a data and realizes that it may be lost (a timer T_s expires and it did not receive a corresponding acknowledgment from R), it retransmits the same frame and waits again for the acknowledgment. However, it can try only up to a fixed maximal number of retransmissions MAX which is a parameter of the protocol. So, the sender maintains a counter of retransmissions CR, and when CR reaches the value MAX, it gives up and concludes that the connection with the receiver is broken. Then, it informs its client that a failure occurred by communicating one of the two confirmations: SNOK if the frame in consideration is not the last frame of the sequence, or SDNK if it is the last one (the sender cannot know if the frame was lost or if its acknowledgment was lost). On the other side, the receiver R uses also a timer T_r to measure the time elapsed between the arrival of two different frames. When R receives a new frame, it resets T_r and, it delivers the transmitted datum with the corresponding indication, otherwise it resends the last acknowledgment. If the timer expires, it concludes that the connection with the sender is broken and delivers an indication RNOK meaning that the transmission failed. Notice that if the first frame is continuously lost, the receiver has no way to detect that the sender is trying to start a new file transmission. In addition, two assumptions are made on the behaviour of S and R:

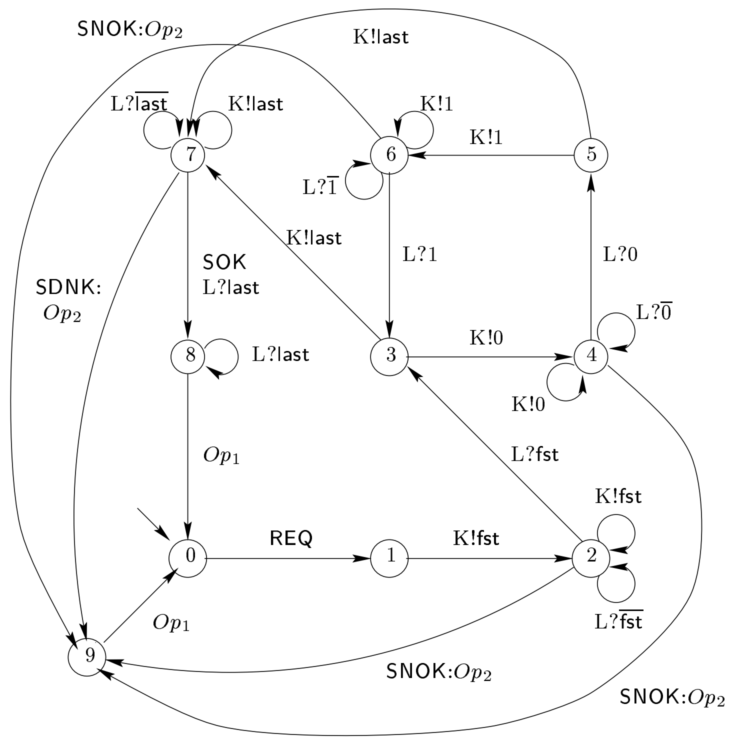
- **A1** R must not conclude prematurely that the connection with S is broken.
- **A2** In case of abortion, S cannot start transmitting frames of another file until R has reacted to abortion and informed its client.

Assumption **A1** means that T_r must be large enough to allow MAX retransmissions of a frame. Assumption **A2** can be implemented for instance by imposing to S to wait enough time after abortion to be sure that T_r has expired.

9.3 Modeling the BRP as a Lossy Channel System

We model the BRP as a lossy channel system which consists of two communicating finite-state machines, the sender S and the receiver R represented in Figures 3 and 4 (with obvious notational conventions). For that, we proceed as follows:

Frames: Since the control of the BRP does not depend on the transmitted data, we hide their values and consider only the informations $(first, last, toggle)$. The set of relevant informations of such form corresponds to a finite alphabet $M = \{\mathsf{fst}, \mathsf{last}, 0, 1\}$, where fst (resp. last) represents the first (resp. last)



$Op_1 = \neg\mathsf{rtrans} \wedge empty(K) \wedge empty(L) \mapsto \mathsf{abort} := \text{false}$
$Op_2 = empty(L) \mapsto \mathsf{abort} := \text{true}$

Figure 3: The sender S

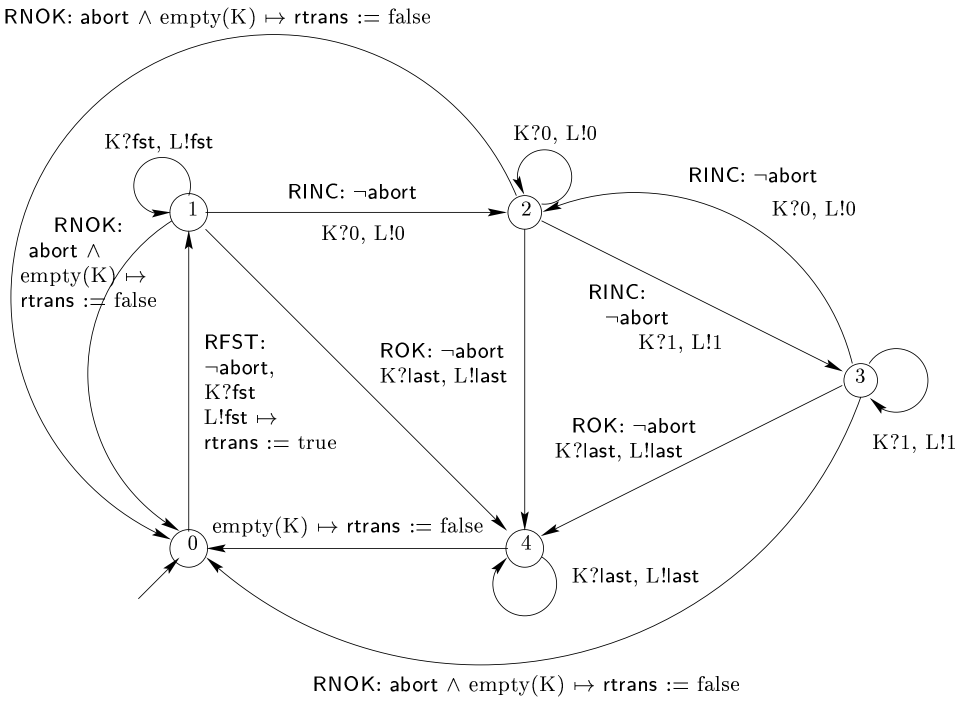


Figure 4: The receiver R

frame, and 0 and 1 represents the intermediate frames since only *toggle* is relevant in this case.

To model the protocol we use two extensions of the basic model described in Section 2. The semantics of these two operations can easily be expresses in the basic model of Section 2. the first operation consists in introducing *channel emptiness testing*: we use enabling conditions on transitions involving a predicate *empty* on channels telling whether a channel is empty. The second extension consists in allowing the components of a system to test and set *boolean shared variables* (recall that we consider here asynchronous parallel composition following the interleaving semantics).

The number of transmitted frames: The only relevant information is whether a frame is the first one, the last one, or an intermediate one. We abstract from the actual value n corresponding to the size of the transmitted sequence of frames, and assume that it can be any positive integer, chosen nondeterministically (by the sender).

Time-outs: Since our model is *untimed*, we cannot express time-outs explicitly, we assume that the sender and the receiver decide nondeterministically when time-outs occur, provided that their corresponding input channels are empty (we use the channel emptiness testing operation).

The counter CR and the value MAX: The only relevant information is whether $\mathsf{CR} < \mathsf{MAX}$ or $\mathsf{CR} \geq \mathsf{MAX}$. We assume that the sender can resend frames an arbitrary number of times before deciding that MAX is reached and aborting the transmission. This makes the size of the channels K and L unbounded. Our model is an abstraction of the whole family of BRPs for arbitrary values of MAX.

Assumptions A1 and A2: Again, since our model is untimed, we cannot impose real-time constraints to implement the assumptions **A1** and **A2**. Instead, we use boolean shared variables to synchronise the sender and the receiver. We consider the two following variables: abort which tells whether the sender has decided abortion, and rtrans which tells whether the receiver considers that the transmission of a sequence of frames has started and is not finished yet, i.e., from the moment it receives the first frame until it informs its client that the transmission is terminated, either successfully or not.

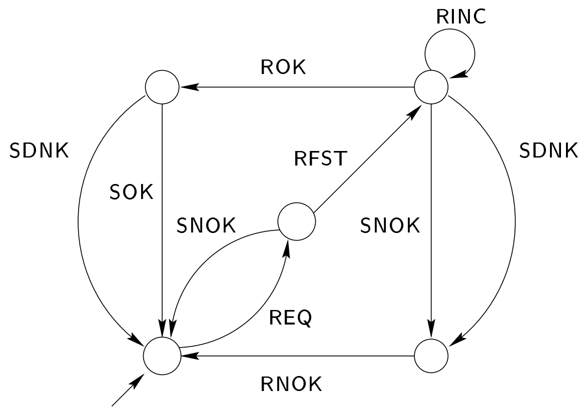


Figure 5: The minimized symbolic graph of the BRP

9.4 Verifying the BRP

To verify the BRP, we proceed as follows: First, we use TReX to generate automatically the set of reachable configurations of the BRP and the corresponding canonical symbolic graph. The obtained graph has 24 symbolic states and 61 transitions. The execution time is 0.35 seconds and the used memory is 6.0 Mb on a Sun4 UltraSparc/SunOS ubac 5.8/Sun-Blade-100/1.5Gb.

Then, we use the tool Aldebaran to minimize this graph according to the observational trace equivalence where the set of observable actions is {REQ, SOK, SNOK, SDNK, RFST, RINC, ROK, RNOK}. We obtain the finite-state labelled transition system with 5 states and 10 transitions given in Figure 5. Properties such as those given in Section 9.1 are expressible in ACTL (the action-based variant of CTL) and can be automatically model checked on the obtained finite-state abstract model of the BRP.

10 Conclusions

We present a method for performing symbolic forward reachability analysis of *lossy channel systems*: systems which consist of finite-state machines communicating over unbounded lossy channels. 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 a 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.

To perform the reachability analysis, we define a subclass of regular expressions which we call SREs, and show that the set of reachable configurations in any lossy channel system can always be described as an SRE. Furthermore, we describe the reachability algorithm by means of a set of operations on SREs each of which can be performed in polynomial time

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 have applied our approach to the non-trivial example of the BRP. We show how to use unbounded channels in order to perform parametric reasoning: unboundedness of the channels models the fact that the number of retransmissions can be any arbitrary positive integer. Our experimentation with the LCS tool suggests that the verification algorithms give quite satisfactory performances in practice.

11 Acknowledgements

This work was partially supported by the European project ADVANCE, Contract No. IST – 1999-29082.

We would like to thank the referees of the paper for many interesting comments.

References

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A Tool for Reachability Analysis of Complex Systems. In *Proc. 13th Intern. Conf. on Computer Aided Verification (CAV'01)*, Paris, France, July 2001. Lecture Notes in Computer Science 2102, Springer-Verlag.

[AJ96a] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.

[AJ96b] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[BBLS92] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property-preserving simulations. In *CAV'92*. LNCS 663, 1992.

[BG96] 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.

[BGWW] 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`.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, Lecture Notes in Computer Science. Springer Verlag, 1997.

[BH] 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.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97, 24^{th} International Colloquium on Automata, Lnaguages, and Programming*, volume 1256 of *Lecture Notes in Computer Science*, 1997.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Invest : A tool for the verification of invariants. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 505–510. Springer-Verlag, 1998.

[Boc78] G. V. Bochman. Finite state description of communicating protocols. *Computer Networks*, 2:361–371, 1978.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6^{th} Int. Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer Verlag, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, April 1983.

[CF87] 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.

[CFI96] 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.

[CVWY90] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *Proc. Workshop on Computer Aided Verification*, 1990.

[DKRT97] P. D'Argenio, J-P. Katoen, T. Ruys, and G.J. Tretmans. The Bounded Retransmission Protocol must be on Time. In *TACAS'97*. LNCS 1217, 1997.

[FGK+96] J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In *CAV'96*. LNCS 1102, 1996.

[FM91] J.Cl. Fernandez and L. Mounier. A tool set for deciding behavioural equivalences. In *CONCUR'91*. LNCS 527, August 1991.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Technical report, LIFAC, Ecole Normale Supérieure de Cachan, 1996. Technical Report.

[GGLR87] 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.

[GL91] O. Grumberg and D.E. Long. Model checking and modular verification. In J.C.M. Baseten and J.F. Groote, editors, *Proc. CONCUR '91, Theories of Concurrency: Unification and Extension*, volume 527 of *Lecture Notes in Computer Science*, pages 250–265, Amsterdam, Holland, 1991. Springer Verlag.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. 9^{th} Int. Conf. on Computer Aided Verification*, volume 1254, Haifa, Israel, 1997. Springer Verlag.

[GvdP93] J.F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets. Technical report, Department of Philosophy, Utrecht University, October 1993.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[HS96] K. Havelund and N. Shankar. Experiments in Theorem Proving and Model Checking for Protocol Verification. In *FME'96.* LNCS 1051, 1996.

[HSV94] L. Helmink, M.P.A. Sellink, and F. Vaandrager. Proof checking a Data Link Protocol. In *Types for Proofs and Programs.* LNCS 806, 1994.

[Mat96] R. Mateescu. Formal Description and Analysis of a Bounded Retransmission Protocol. Technical report no. 2965, INRIA, 1996.

[May00] R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.

[Pac87] 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.

[PP91] 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.

[SZ91] 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 *Lecture Notes in Computer Science.* Springer Verlag, 1991.

[VW86] 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.

A Appendix - Proofs of Some Lemmas

Proof of Theorem 3.3

First, we show some auxiliary lemmas.

Lemma A.1 For a finite alphabet M and string $x \in M^+$, there is a product p such that $y \in [\![p]\!]$ if and only if $x \not\preceq y$.

Proof. Let x be of the form $a_1 a_2 \cdots a_m$. Let e_i be the atomic expression $(b_1^i + \ldots + b_k^i)^*$ where $\{b_1^i, \ldots, b_k^i\} = M \setminus \{a_i\}$. Notice that if $M = \{a_i\}$ then $[\![e_i]\!] = \epsilon$. We define $p = e_1 \bullet (a_1 + \epsilon) \bullet e_2 \bullet (a_2 + \epsilon) \cdots (a_{m-1} + \epsilon) \bullet e_m$.

We show that p satisfies the property stated in the Lemma. Suppose that $x \preceq y$. We prove that $y \notin [\![p]\!]$. By definition of $\preceq$ we know that y is of the form $y_1 \bullet a_1 \bullet y_2 \bullet a_2 \bullet y_3 \bullet \cdots \bullet y_m \bullet a_m \bullet y_{m+1}$, where $y_i \in M^*$. By definition of e_i we know that $a_i \notin [\![e_i]\!]$ and hence $y_i \bullet a_i \notin [\![e_i]\!]$. This implies that $y_1 \bullet a_1 \bullet y_2 \bullet a_2 \bullet y_3 \bullet \cdots \bullet y_\ell \bullet a_\ell \notin [\![e_1 \bullet (a_1 + \epsilon) \bullet e_2 \bullet (a_2 + \epsilon) \cdots (a_{\ell-1} + \epsilon) \bullet e_\ell]\!]$ for any ℓ; and in particular $y_1 \bullet a_1 \bullet y_2 \bullet a_2 \bullet y_3 \bullet \cdots \bullet y_m \bullet a_m \notin [\![e_1 \bullet (a_1 + \epsilon) \bullet e_2 \bullet (a_2 + \epsilon) \cdots (a_{m-1} + \epsilon) \bullet e_m]\!] = [\![p]\!]$.

Now, suppose that $x \not\preceq y$. We show that $y \in [\![p]\!]$. Let ℓ be the largest natural number such that $a_1 a_2 \cdots a_\ell \preceq y$. Obviously $0 \leq \ell < m$. This means that y is of the form $y_0 \bullet a_1 \bullet y_1 \bullet a_2 \bullet \cdots \bullet y_{\ell-1} \bullet a_\ell \bullet y_\ell$, where y_i is a string over $M \setminus \{a_{i+1}\}$ for $i : 0 \leq i < \ell$. Furthermore, we know that $a_{\ell+1}$ does not occur in y_ℓ (otherwise, we would have $a_1 a_2 \cdots a_{\ell+1} \preceq y$ which violates the maximality of ℓ). This implies that $y_i \in [\![e_{i+1}]\!]$ for $i : 0 \leq i \leq \ell$, and hence $y \in [\![p]\!]$. $\square$

In Lemma A.2 we show that each downward-closed language can be characterized through a finite set of counter-examples. The proof of the lemma relies on Higman's theorem [Hig52] 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 $x_i \preceq x_j$.

Lemma A.2 For a finite alphabet M and a downward-closed language L over M, there is a finite set $\{x_1, \ldots, x_n\}$ of strings over M, such that for any string $y \in M^*$, it is the case that $y \in L$ if and only if $x_i \not\preceq y$, for each $i : 1 \leq i \leq n$.

Proof. Consider the complement L' of L. It is clear that L' is upward-closed. We show that there is a finite set $\{x_1, \ldots, x_n\}$ of strings over M, such that that for any string y, we have $y \in L'$ if and only if $x_i \preceq y$, for some $i : 1 \leq i \leq n$. The result follows immediately.

Suppose that no such a set exists. We derive an infinite sequence $x_0, x_1, x_2, \ldots$ violating Higman's lemma. We take x_0 any element of L. We define x_j to be any string in L such that $x_i \not\preceq x_j$ for all $i : 0 \leq i < j$. The string x_j exists by assumption. $\square$

Lemma A.3 For products p_1 and p_2, there is (and can compute) an SRE $p_1 \wedge p_2$ such that $[\![p_1 \wedge p_2]\!] = [\![p_1]\!] \cap [\![p_2]\!]$.

Proof. We define $p_1 \wedge p_2$ as follows.

$$p_1 \wedge p_2 =$$

$$\begin{cases}
\emptyset & \text{if either } r_1 = \emptyset \text{ or } r_2 = \emptyset \\[2ex]
\epsilon & \text{if either } r_1 = \epsilon \text{ and } r_2 \neq \emptyset \text{ or} \\
 & r_2 = \epsilon \text{ and } r_1 \neq \emptyset \\[2ex]
(a+\epsilon) \bullet (r_1' \wedge r_2') & \text{if } r_1 = (a+\epsilon) \bullet r_1' \text{ and } r_2 = (a+\epsilon) \bullet r_2' \\[2ex]
(r_1' \wedge r_2) + (r_1 \wedge r_2') & \text{if } r_1 = (a_1+\epsilon) \bullet r_1' \text{ and} \\
 & r_2 = (a_2+\epsilon) \bullet r_2' \text{ and } a_1 \neq a_2 \\[2ex]
(a+\epsilon) \bullet (r_1' \wedge r_2) & \text{if } r_1 = (a+\epsilon) \bullet r_1' \text{ and} \\
 & r_2 = (a_1 + \cdots + a_m)^* \bullet r_2' \text{ and} \\
 & a \in \{a_1, \ldots, a_m\} \\[2ex]
(a+\epsilon) \bullet (r_1 \wedge r_2') & \text{if } r_1 = (a_1 + \cdots + a_m)^* \bullet r_1' \text{ and} \\
 & r_2 = (a+\epsilon) \bullet r_2' \text{ and} \\
 & a \in \{a_1, \ldots, a_m\} \\[2ex]
(d_1 + \cdots + d_k)^* \bullet (r_1 \wedge r_2') + & \text{if } r_1 = (a_1 + \cdots + a_m)^* \bullet r_1' \text{ and} \\
(d_1 + \cdots + d_k)^* \bullet (r_1' \wedge r_2) & r_2 = (b_1 + \cdots + b_n)^* \bullet r_2' \text{ and} \\
 & \{d_1, \ldots, d_k\} = \{a_1, \ldots, a_m\} \cap \{b_1, \ldots, b_n\}
\end{cases}$$

$\square$

Proof of Theorem 3.3 We show that if a language L is downward-closed then L is simply regular. The result follows immediately. Assume that L is downward-closed. By Lemma A.2 it follows that there is a finite set

$\{x_1, \ldots, x_n\}$ of strings over M, such that for any string y, it is the case that $y \in L$ if and only if $x_i \not\preceq y$, for each $i : 1 \leq i \leq n$. If $x_i = \epsilon$ for some $i : 1 \leq i \leq n$ then $L = \emptyset$. Otherwise, $x_i \in M^+$ for each $i : 1 \leq i \leq n$. By Lemma A.1 it follows that there are products $p_1, \ldots, p_n$ such that $L = [\![p_1]\!] \cap \cdots \cap [\![p_n]\!]$. The result follows from Lemma A.3. $\square$

Proof of Lemma 4.1

In this proof, we refer to atomic expressions of the forms $(a + \epsilon)$ and $(a_1 + \cdots + a_\ell)^*$ as *non-star* and *star* atomic expressions respectively.

Given any natural number k, we define a string x such that $x \in [\![p]\!]$ and $x \notin [\![p']\!]$, for any product p', where $p \not\sqsubseteq p'$ and p' contains at most k atomic expressions. The result follows immediately.

Let p' be of the form $e'_1 \bullet e'_2 \bullet \cdots \bullet e'_k$.

First, we prove the claim for the special case, where p is an atomic expression e. If e is of the form $(a + \epsilon)$ then we define $x = a$. If e is of the form $(a_1 + \ldots + a_\ell)^*$ then we define $x = (a_1 \bullet \cdots \bullet a_\ell)^{k+1}$. We show that x satisfies the stated property. If e is a non-star atomic expression $(a+\epsilon)$ then we know that $a \notin [\![e'_i]\!]$ for each $i : 1 \leq i \leq k$, and hence $a \notin [\![p']\!]$. If e is a star atomic expression $(a_1 + \ldots + a_\ell)^*$, then we notice that $\ell \geq 1$ (otherwise $e \sqsubseteq p'$ which contradicts the assumption). We use induction on k to show that $x \notin [\![p']\!]$. The base case (with $k = 0$) is trivial. We consider the induction step and assume that $k > 0$. There are two possible cases: (i) if e'_k is non-star. By the induction hypothesis we have $(a_1 \bullet \cdots \bullet a_\ell)^k \notin [\![e'_1 \bullet e'_2 \bullet \cdots \bullet e'_{k-1}]\!]$. Since e'_k is non-star and $\ell \geq 1$, it follows that $(a_1 \bullet \cdots \bullet a_\ell)^{k+1} \notin [\![e'_1 \bullet e'_2 \bullet \cdots \bullet e'_k]\!]$. (ii) if e'_k is star. We know that $e \not\sqsubseteq e'_k$ (otherwise $e \sqsubseteq p'$ which is a contradiction). Since e'_k is star and $e \not\sqsubseteq e'_k$, there exists an $i : 1 \leq i \leq \ell$ such that $a_i \notin [\![e'_k]\!]$. This implies that $(a_1 \bullet \cdots \bullet a_\ell) \notin [\![e'_k]\!]$. By the induction hypothesis we know that $(a_1 \bullet \cdots \bullet a_\ell)^k \notin [\![e'_1 \bullet e'_2 \bullet \cdots \bullet e'_{k-1}]\!]$. This implies that $(a_1 \bullet \cdots \bullet a_\ell)^{k+1} \notin [\![e'_1 \bullet e'_2 \bullet \cdots \bullet e'_k]\!]$.

Now we show the general case. Let p be of the form $e_1 \bullet \cdots \bullet e_m$. We define $x = x_1 \bullet \cdots \bullet x_m$, where x_i is derived form e_i in the same manner to the derivation of x from e in the special case above. We show that x satisfies the stated property. We use induction on m. The base case (when $m = 0$) is trivial. We consider the induction step and assume that $m > 0$. If $e_1 \not\sqsubseteq p'$ then the proof is reduced to the special case above. Otherwise, we know that $m > 1$. Let k_1 be the minimum natural number such that $e_1 \sqsubseteq e'_1 \bullet \cdots \bullet e'_{k_1}$. There are two possible cases: (i) if e'_{k_1} is non-star. By the induction hypothesis $x_1 \notin [\![e'_1 \bullet \cdots \bullet e'_{k_1-1}]\!]$. Furthermore, we know that $e_2 \bullet \cdots \bullet e_m \not\sqsubseteq e'_{k_1+1} \bullet \cdots \bullet e'_k$

(otherwise $p \sqsubseteq p'$ which contradictions the assumption). By the induction hypothesis it follows that $x_2 \bullet \cdots \bullet x_m \notin [\![e'_{k_1+1} \bullet \cdots \bullet e'_k]\!]$. Since e'_{k_1} is non-star, we conclude that $x_1 \bullet x_2 \bullet \cdots x_m \notin [\![e'_1 \bullet \cdots \bullet e'_k]\!]$. (ii) If e'_{k_1} is star. By the induction hypothesis $x_1 \notin [\![e'_1 \bullet \cdots \bullet e'_{k_1-1}]\!]$. Furthermore, since e'_{k_1} is star, we know that $e_2 \bullet \cdots \bullet e_m \not\sqsubseteq e'_{k_1} \bullet \cdots \bullet e'_k$ (otherwise $p \sqsubseteq p'$ which contradictions the assumption). By the induction hypothesis it follows that $x_2 \bullet \cdots \bullet x_m \notin [\![e'_{k_1} \bullet \cdots \bullet e'_k]\!]$. We conclude that $x_1 \bullet x_2 \bullet \cdots x_m \notin [\![e'_1 \bullet \cdots \bullet e'_k]\!]$.
$\square$

Proof of Lemma 6.2

First, we show some auxiliary definitions and lemmas. For strings x and y, we define $y \ominus x$ as follows.

$$y \ominus x \quad = \quad \begin{cases} y & \text{if } x = \epsilon \\ \text{undefined} & \text{if } y = \epsilon \text{ and } x \neq \epsilon \\ y_1 \ominus x_1 & \text{if } y = a \bullet y_1 \text{ and } x = b \bullet x_1 \text{ and } a = b \\ y_1 \ominus x & \text{if } y = a \bullet y_1 \text{ and } x = b \bullet x_1 \text{ and } a \neq b \end{cases}$$

Observe that $y \ominus x$ is defined if and only if $x \preceq y$.

Lemma A.4 For strings x and y, if $x \prec y$ and $y^{m+1} \ominus x^{m+1} = y^m \ominus x^m$, then $y^n \ominus x^n = y^m \ominus x^m$ for each $n \geq m$.

Proof. We use induction on $n - m$. The base case is trivial. Since $x \prec y$ we know that $y^n \ominus x^n$ is defined and hence, by the definition of $\ominus$, it follows that $y^{n+1} \ominus x^{n+1} = ((y^n \ominus x^n) \bullet y) \ominus x$. By the induction hypothesis it follows that $y^n \ominus x^n = y^m \ominus x^m$, so $y^{n+1} \ominus x^{n+1} = ((y^m \ominus x^m) \bullet y) \ominus x = y^{m+1} \ominus x^{m+1} = y^m \ominus x^m$.

$\square$

Proof of Lemma 6.2 We show that either $y \not\preceq y^m \ominus x^m$ for all $m \geq 0$, or there is $m \leq |y|$ such that $y \preceq y^m \ominus x^m$. The result follows immediately.

Notice that if there is $j : 0 \leq j < |y|$ such that $y \preceq y^j \ominus x^j$, then the claim holds trivially. Otherwise, there are two cases.

- If there exists $k : 0 \leq k < |y|$ such that $y^{k+1} \ominus x^{k+1} = y^k \ominus x^k$. By Lemma A.4 it follows that $y^m \ominus x^m = y^k \ominus x^k$ for all $m \geq k$. This means that $y \not\preceq y^m \ominus x^m$ for all $m \geq 0$.

- If for each $k : 0 \leq k < |y|$ it is the case that $y^{k+1} \ominus x^{k+1} \neq y^k \ominus x^k$. This implies $x \prec y$. It follows that $y^k \ominus x^k \prec y^{k+1} \ominus x^{k+1}$, and hence $k \leq |y^k \ominus x^k|$, for each $k : 0 \leq k \leq |y|$. This implies that $y \preceq y^m \ominus x^m$, where $m = |y|$.

$\square$