Reasoning about probabilistic lossy channel systems

Parosh Abdulla
Dept of Computer Systems,
Uppsala University,
Uppsala, Sweden

Christel Baier
Dept of Computer Science
University of Bonn
Bonn, Germany

Purushothaman Iyer*
Dept of Computer Science
North Carolina State University
Raleigh, NC 27695, USA

Bengt Jonsson
Dept of Computer Systems,
Uppsala University,
Uppsala, Sweden

May 8, 2001

Abstract

We consider the problem of deciding whether an infinite-state system (expressed as a Markov chain) satisfies a correctness property with probability 1. This problem is, of course, undecidable for general infinite-state systems. We focus our attention on the model of *probabilistic lossy channel systems* consisting of finite-state processes that communicating over unbounded lossy FIFO channels. Abdulla and Jonsson have shown that safety properties are decidable while progress properties are undecidable for non-probabilistic lossy channel systems. Under assumptions of "sufficiently high" probability of loss, Baier and Engelen have shown how to check whether a property holds of probabilistic lossy channel system with probability 1.

In this paper we show that the problem of checking whether a progress property holds with probability 1 is undecidable, if the assumption about "sufficiently high" probability of loss is omitted. More surprisingly, we show that checking whether safety properties hold with probability 1 is undecidable too. Our proof depends upon simulating a perfect channel, with a high degree of confidence, using lossy channels.

1 Introduction

Finite state machines which communicate over unbounded FIFO channels have been used as an abstract model of computation for reasoning about communication protocols [4], and they form the backbone of ISO protocol specification languages Estelle and SDL. However,

*Suppoted in part by ARO under grant DAAG55-98-1-03093 and by STINT.

the model is Turing-powerful which makes most verification problems of interest undecidable. Ever since the publication of the Alternating bit protocol it has been customary to assume, while modeling a protocol, that the communication channels between processes are free of errors. Possible errors in the communication channels are treated separately, or are completely ignored. However, over the past few years there have been attempts to rectify this situation to allow modeling of imperfections in the communication medium. Abdulla and Jonsson, in [2], considered a model where messages could be lost from the queue, while waiting in a queue to be delivered. They showed that the *reachability* problem (and consequently, the problem of checking for safety properties) is decidable. However, in [1], they also showed that checking for progress properties is undecidable – a consequence of fairness arguments necessary to prove progress properties.

Randomization is an oft-used technique to provide tractable, approximate solutions to intractable problems [5, 8]. Given that we are dealing with imperfections in the communication medium it is then natural to consider models of communicating processes where the probability of message loss is taken into account. The reason for considering such an approach is to supplant fairness arguments, necessary for showing progress properties, by probablistic arguments. Furthermore, by considering a probabilistic model we would be making our model more closer to reality. In [6], Iyer and Narasimha considered whether a probabilistic lossy channel systems satisfies an LTL formula with a probability greater than p, with in a tolerance ϵ. In [3] Baier and Engelen considered the following more general problem (which is the topic of this paper):

> given a probabilistic lossy channel system $\mathcal{L}$ and a linear-time temporal logic formula ϕ, does the set of sequences of $\mathcal{L}$ that satisfy ϕ have probability 1.

A partial answer to this question was given by Baier and Engelen who showed that it is decidable if the probability of message loss is sufficiently high. Here "sufficiently high" was taken to mean roughly "at least $\frac{1}{2}$". Under this restriction, it can be shown that (with probability 1) the number of messages in the channel cannot grow unboundedly, and that the general model checking problem can be reduced to checking reachability (which is decidable by [2]).

In this paper we will settle a question left open by Baier and Engelen – that the problem of model checking progress properties with probability 1 is undecidable, when we relax the assumption that messages are lost with "sufficiently high" probability. Our proof is a construction which shows that if the probability of losing each message is lower than $\frac{1}{2}$ then a lossy channel system can (with probability 1) simulate a finite state machine which operates on *perfect* FIFO channels. The main idea is to let each message transmission of the perfect channel system be simulated by a number of retransmissions of the message in the lossy channel system. If the scheme for retransmissions is chosen appropriately, the simulation will be faithful in a certain well-defined sense – a fact which allows us to establish the undecidability result.

The rest of the paper is organized as follows. In the next two sections we present the necessary definitions for probabilistic lossy channel systems and the probabilistic reachability problem. In Section 4 we provide an overview of our undecidability proof, which we follow in Section 5 with the necessary constructions and in Section 6 with the proofs for the main lemmata.

2 Communicating Finite-state Machines

In this section we introduce communicating finite-state machines (CFSMs) and some of their properties.

For a set M we use M^* to denote the set of finite strings of elements in M. For $x \in M^*$, let $x(i)$ denote the i^{th} element of x. For $x, y \in M^*$ we let $x \bullet y$ denote the concatenation of x and y. The empty string is denoted by ε. For sets C and M, a *string vector from C to M* is a function in $C \mapsto M^*$. For a string vector w from C to M we use $w[c := x]$ for the string vector w' such that $w'(c) = x$, and $w'(d) = w(d)$, for $d \neq c$. The string vector which maps all elements in C to the empty string is denoted ε.

Definition 2.1 A *Communicating Finite-State Machine (CFSM)* $\mathcal{C}$ is a tuple $\langle S, s_{init}, C, M, \delta \rangle$, where

S is a finite set of *control states*,

$s_{init} \in S$ is an *initial control state*,

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 control states, and op is either a label of form $c!m$, $c?m$, or *empty*, where $c \in C$ and $m \in M$.

Given this finite description of a system, we can now talk about the *global state* γ of a $\mathcal{C}$ as a pair $\langle s, w \rangle$, where $s \in S$ and w is a string vector from C to M. The *initial global state* γ_0 of $\mathcal{C}$ is the pair $\langle s_0, \varepsilon \rangle$. The progression of the system from one global state to another can now be formalized as a transition relation $\longrightarrow \subseteq (S \times [C \mapsto M^*]) \times \delta \times (S \times [C \mapsto M^*])$, with a typical member written as a triple of the form $\langle \gamma_1, t, \gamma_2 \rangle$, where γ_1 and γ_2 are global states and $t \in \delta$. . The relation $\gamma_1 = \langle s_1, w_1 \rangle \xrightarrow{t} \gamma_2 = \langle s_2, w_2 \rangle$ holds iff one of the following conditions is satisfied.

1. $t = \langle s_1, c!m, s_2 \rangle$ and $w_2 = w_1[c := w_1(c) \bullet m]$.
2. $t = \langle s_1, c?m, s_2 \rangle$ and $w_1 = w_2[c := m \bullet w_2(c)]$.
3. $t = \langle s_1, empty, s_2 \rangle$ and $w_2 = w_1$.

For a global state γ, we define the set $en(\gamma)$ of *enabled* transitions at γ such that $t \in en(\gamma)$ iff $\gamma \xrightarrow{t} \gamma'$ for some γ'. We let $\longrightarrow = \cup_{t \in \delta} \xrightarrow{t}$ and let $\xrightarrow{*}$ denote the reflexive transitive closure of $\longrightarrow$. For global states γ_1 and γ_2, we say that γ_2 is *reachable* from γ_1 if $\gamma_1 \xrightarrow{*} \gamma_2$. For a global state γ, we say that γ is *reachable* if γ is reachable from γ_0.

A *computation* π is a sequence (either finite or infinite) of states and transitions: $\gamma_1\ t_1\gamma_2\ t_2\ \gamma_3\ \cdots\ t_{n-1}\gamma_n \cdots$ such that $\gamma_i \xrightarrow{t_i} \gamma_{i+1}$ for each $i : 1 \leq i$. We will use $\pi(i)$ to denote the i^{th} state γ_i visited in π. If a computation π is finite we say that π *leads* from γ_1 to γ_n[1].

We define the control state reachability problem for CFSMs (Reach-CFSM) as follows

Instance A CFSM $\mathcal{C}$ and a control state s in $\mathcal{C}$.

Question Is there a w such that $\langle s, w\rangle$ is reachable?

The following result is well-known (e.g. [4]).

Theorem 2.2 *Reach-CFSM is undecidable.*

The idea behind the proof is to use one of the channels to simulate the tape of a Turing machine. In fact, this construction implies that the problem is undecidable even for the class of CFSMs with only one channel. We can make a further restriction on CFSMs without affecting the undecidability result, namely we can assume that each control state has at least one outgoing transition which is not a receive transition (if such a state q exist, then add a self-loop $\langle q, empty, q\rangle$). In the sequel, we work only with CFSMs which have a single channel, and where all control states have at least one outgoing transition which is not a receive transition.

3 Probabilistic Lossy Channel systems

In this section we consider probabilistic lossy channel systems (PLCSs).

Definition 3.1 A *lossy channel system (LCS)* $\mathcal{L}$ is of the same form as a CFSM. However, the transition relation $\longrightarrow$ is extended such that $\langle s_1, w\rangle \xrightarrow{t} \langle s_2, w\rangle$ if $t = \langle s_1, c!m, s_2\rangle$. In other words, we allow a message sent to the channel to be lost without ever being appended to the contents of the channel.

Note that this semantics of loss is slightly different from the one used in [2, 6, 3], where a message can be lost at any time from the queue. In contrast in the current paper a message can only be lost as it is being placed in the queue. However, the set of reachable states under both semantics is the same [2].

Definition 3.2 A *probabilistic Lossy Channel system (PLCS)* is a tuple

$$\langle S, s_{init}, C, M, \delta, \mathsf{W}, \lambda\rangle$$

where $\langle S, s_{init}, C, M, \delta\rangle$ is an LCS, λ is real number in the interval $[0, 1]$ representing the probability of losing messages, and W is a *weight function* which assigns to each transition $t \in \delta$ a positive real number $\mathsf{W}(t)$.

[1] Note that γ' is reachable from γ iff there is a computation leading from γ to γ'.

Given that a PLCS is also an LCS we will lift the transition relation $\longrightarrow$, from LCS, to obtain the transition relation for PLCS by customizing the weights assigned to transitions.. More precisely, for a global state γ and a transition t we define $\mathsf{W}'(\gamma)(t)$ to be equal to $\mathsf{W}(t)$ if $t \in en(\gamma)$ and to be equal to 0 otherwise. We define $P(\gamma_1 \xrightarrow{t} \gamma_2)$, where $\gamma_1 = \langle s_1, w_1 \rangle$ and $\gamma_2 = \langle s_2, w_2 \rangle$, to be equal to

- $(1-\lambda) \cdot (\mathsf{W}'(\gamma_1)(t)/\sum_{t' \in \delta} \mathsf{W}'(\gamma_1)(t'))$, if t is of the form $\langle s_1, c!m, s_2 \rangle$ and $w_1 \neq w_2$. This corresponds to performing a non-lossy send operation.
- $\lambda \cdot (\mathsf{W}'(\gamma_1)(t)/\sum_{t' \in \delta} \mathsf{W}'(\gamma_1)(t'))$, if t is of the form $\langle s_1, c!m, s_2 \rangle$ and $w_1 = w_2$. This corresponds to a lossy send operation.
- $(\mathsf{W}'(\gamma_1)(t)/\sum_{t' \in \delta} \mathsf{W}'(\gamma_1)(t'))$, if t is of the form $\langle s_1, c?m, s_2 \rangle$ or of the form $\langle s_1, empty, s_2 \rangle$.

For a finite computation $\pi = \gamma_1\ t_1\ \gamma_2\ t_2\ \gamma_3\ \cdots\ t_{n-1}\ \gamma_n$, we define

$$\mathcal{B}_\pi = \{\pi' | \pi \text{ is a prefix of } \pi'\}$$

as a basic cylinder set with probability $P(\mathcal{B}_\pi) = \prod_{0<i<n} P(\gamma_i \xrightarrow{t} \gamma_{i+1})$. We will assume the standard measure space based on the Borel field generated from these basic cylinder sets [7], by taking closure under denumerable unions, denumerable intersection and complementation. For global states γ_1 and γ_2, we define $P(\gamma_1, \gamma_2) = P\left(\bigcup_{\pi \text{ leads from } \gamma_1 \text{ to } \gamma_2} \mathcal{B}_\pi\right)$. For a global state γ and a control state s, we define $P(\gamma, s) = P(\bigcup_{\pi \text{ leads from } \gamma \text{ to some } \langle s,w \rangle} \mathcal{B}_\pi)$. We define $P(\gamma) = P(\gamma_0, \gamma)$ and $P(s) = P(\gamma_0, s)$.

The reachability problem for PLCS (Reach-PLCS) is defined as follows:

Instance A PLCS $\mathcal{L}$ and control state s in $\mathcal{L}$.

Question Is $P(s) = 1$?

Note that checking safety properties can be reduced to checking reachability of a bad control state [2]. Furthermore, checking for liveness properties amounts to checking for repeated reachability of a control state [3]. Consequently, our attention to the probabilistic reachability problem is appropriate.

4 Overview of our Undecidability Proof

In this section, we give an overview of the undecidability proof for Reach-PLCS. The undecidability result is achieved through a reduction from Reach-CFSM.

Suppose that we are given an instance of Reach-CFSM, i.e., a CFSM $\mathcal{C}$ and a control state s_F. Recall that we assume that $\mathcal{C}$ has a single channel, which we will call c_M. We shall construct an equivalent instance of Reach-PLCS, i.e., a PLCS $\mathcal{L}$ which "simulates" $\mathcal{C}$ such that s_F is reached with probability 1 if and only if s_F is reachable in $\mathcal{C}$. The main idea of the construction is to implement two protocols which allow $\mathcal{L}$ to simulate the computations of $\mathcal{C}$. One of the protocols generates multiple copies of messages, while the other one restarts the system.

Generating multiple copies Due to the risk of losing messages, each send operation $c_M!m$ of $\mathcal{L}$ is simulated by a sequence of retransmissions of the message m on channel c_M. This means that each receive operation must be simulated by a "receive loop" where all copies of a message are received.

Retransmitting a message several times decreases the probability that the message is lost in the channel (i.e., that all copies of the message are lost), but the probability is still nonzero. We will therefore construct a scheme in which the number of retransmissions of a message is increased as the execution proceeds. This means that the probability of losing a message will decrease and converge to zero. To be more precise, let $\mathsf{perfectafter}k$ denote the event that among messages queued, in $\mathcal{L}$, to simulate a transmission of a message in $\mathcal{C}$ all message losses are restricted to the first k message retransmissions. We construct a retransmission scheme in which $P(\mathsf{perfectafter}k)$ converges to 1 as k goes to infinity. Having constructed this scheme, let us consider the probability that in an execution there is a k such that no messages are lost after the first k messages. This event is denoted $\exists k\ \mathsf{perfectafter}k$ and satisfies

$$\exists k\ \mathsf{perfectafter}k = \bigvee_{k=0}^{\infty} \mathsf{perfectafter}k$$

Hence, for any k we have

$$P(\exists k\ \mathsf{perfectafter}k) \geq P(\mathsf{perfectafter}k)$$

which, since $P(\mathsf{perfectafter}k)$ can be made arbitrary close to 1, implies that

$$P(\exists k\ \mathsf{perfectafter}k) = 1 \tag{1}$$

In other words, we can deduce that with probability 1 there is a k such that after the first k messages no more messages will get lost. Hence the simulation will be "eventually perfect" in the sense that for each transmitted message in $\mathcal{C}$ at least one of its simulating retransmissions will not be lost in the channel. The ability to construct an eventually perfect simulation of a CFSM by a PLCS is the central concept in the proof. Once the simulation becomes perfect we can let the PLCS simulate the CFSM, implying that the PLCS will reach s_F if and only if the CFSM does so. We observe that the strings generated by simulating different send operations need to be separated by a special symbol # (say) which we does not belong to the original alphabet. Otherwise, given two consecutive occurences of the same messages m in the channel we will not be able to tell whether they correspond to two send operations in $\mathcal{C}$, or to two retransmissions of the same send operation.

Restarting During the early parts of the simulation we have little control over what happens in the PLCS. Therefore we will devise a mechanism whereby the simulation is "restarted" periodically in order to recover from the possibility of losing all retransmissions of a message. The periods between restarts should be longer and longer, so that eventually they are sufficiently long to simulate the entire computation from initial state to final state in the CFSM (in case the state s_F is reachable). Notice that this implies that $\mathcal{L}$ should never deadlock, since this would mean that the restarting procedure cannot be continued.

To carry out the construction just outlined, the PLCS $\mathcal{L}$ uses, in addition to c_M, two lossy channels called the *retransmission counter* c_T and the *resetting counter* c_S.

The counter c_T The number of retransmissions (copies) of each message will be controlled by the counter c_T. This counter is implemented by a (lossy) channel with two messages, 0 and 1 (say). When the simulation of a send operation is about to start the contents of c_T will be of the form 0^k, denoting a counter value k. The operation of sending a message m in $\mathcal{C}$ is replaced in $\mathcal{L}$ by the following sequence of operations. First, we send k copies of # to c_M. This is done by receiving the 0s in the head of c_T. Each time a 0 is received, we send two 1s to c_T and send a copy of # to c_M. This means that while sending the #s, the content of the counter will be of the form $0^{k_1}1^{k_2}$, where k_1 is the number of times the symbol # still have to be transmitted. This operation will continue until a 1 is received, indicating that all 0s have been received. At this point the content of c_T will be of the form 1^ℓ, where ℓ is the number of times the message m will be copied to c_M. The copies of m are sent in a similar manner to transmission of #s, with the difference that the system now receives 1s from c_T and send 0s. Observe that the above procedure produces k copies of # followed by ℓ copies of m. We also observe that in the long run (even taking into account the possibility of losing messages), the value of c_T increases, and therefore the number of copies produced will also increase. Receiving a message m is simulated by a loop which first removes all #s in front of the queue and then removes all copies of m.

The counter c_S The mechanism for restarting the simulation must be devised so that the system $\mathcal{L}$ is restarted infinitely often, with increasing periods between the restarts. This is controlled by the counter c_S. The counter c_S should satisfy three properties

- Its value should be increasing at successive restarting points.
- Its value should be decreasing between two restarting points.
- the value of c_S should grow more quickly than the value of c_T.

The first two conditions are not contradictory (although they might seem to be). The trick is to store both the current value and the next value of c_S. During a simulation, we continuously decrease the current value and increase the next value of the counter. When the restarting procedure is initiated the contents of c_T is either of the form 0^k or of the form 1^k. The current value k of c_S represents the number of steps for which the system $\mathcal{C}$ is simulated, before the next restart is performed. Suppose that the content of c_S is of the form 0^k. When simulating a step of $\mathcal{C}$ the current value of c_S is decreased. In a manner similar to c_T, this is done by receiving a 0 and sending two 1s. This means that, during the simulation procedure, the content of c_S will be of the form $0^{k_1}1^{k_2}$, where k_1 is the current value of the counter (the number of steps of $\mathcal{C}$ that still have to be simulated) and k_2 is the next value of the counter. This implies that the current value of the counter will decrease, while the next value will increase during the simulation. The main difficulty in implementing c_S is to guarantee that its value will decrease even if the rest of the system has deadlocked. This can happen if we try to receive a message m from c_M which has been lost. To avoid this, we add to each receive loop an additional loop decreasing the value c_M. This guarantees that the system is reset infinitely often with increasing intervals according to the following

- If the message we attempt to receive is not in c_M then the loop decreasing c_S will eventually make the current value of c_S equal to 0 and the system is restarted.

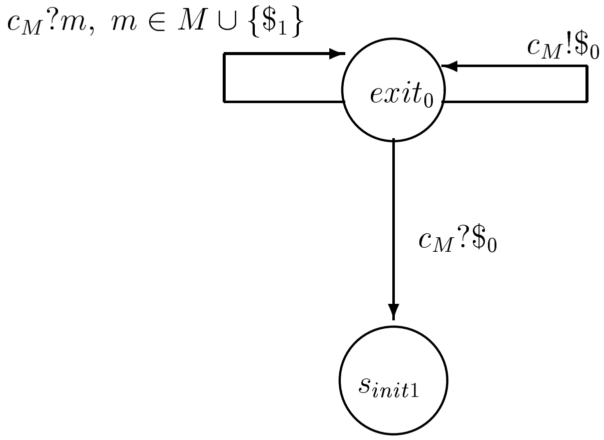


Figure 1: The restarting operation

- If the message we attempt to receive is available then the loop decreasing c_S will still be running. However, since c_S grows more quickly than c_T, the number of copies of the message (which is bound by the value of c_T) will all be received before c_S has decreased to 0. Therefore, more steps of the simulation can be performed.

Observe that the contents of c_T is always either of the form $0^{k_1}1^{k_2}$ in which case we say that the system is in mode 0, or of the form $1^{k_1}0^{k_2}$ in which case we say that the system is in mode 1.

5 Implementation of Operations

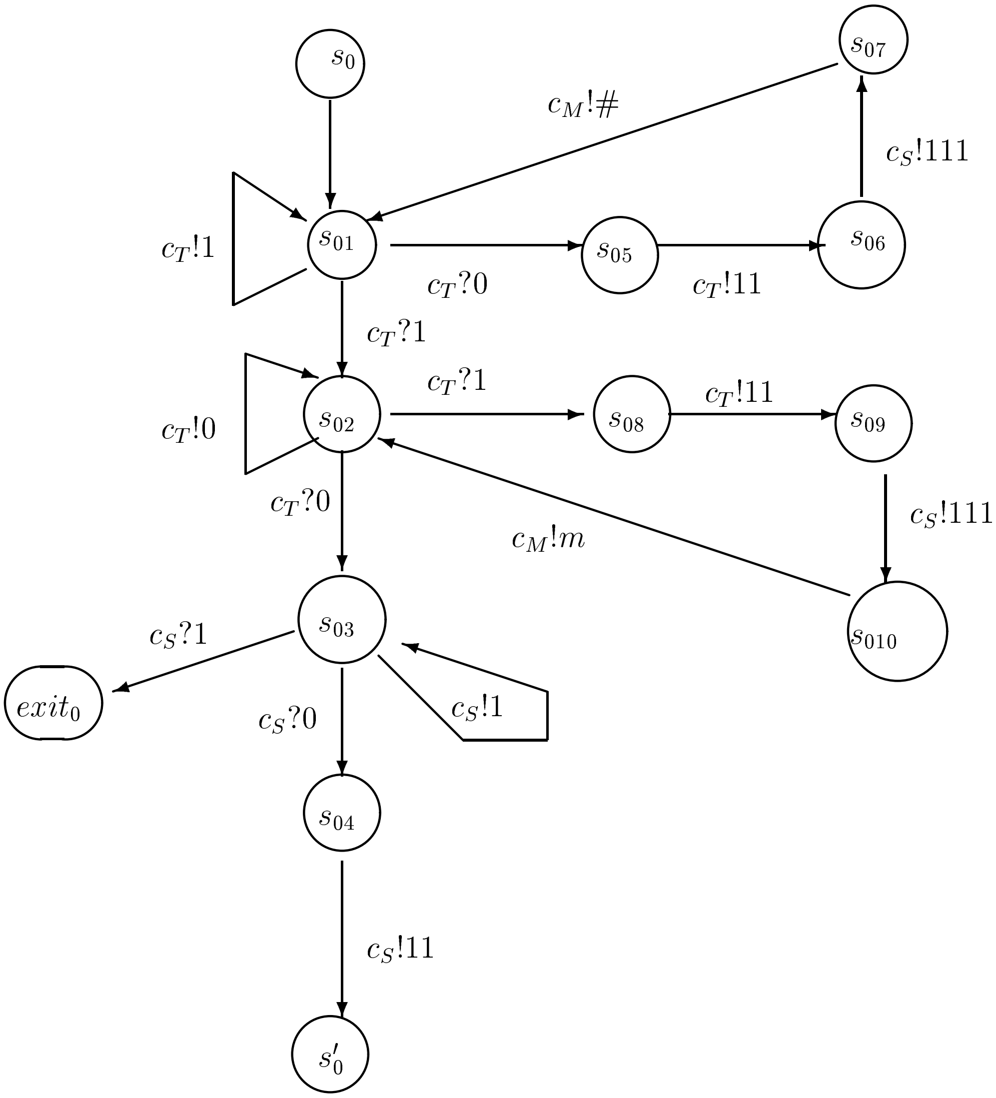
In this section we show in greater detail how to implement $\mathcal{L}$ such that its behaviour satisfies the properties described in Section 4. Each control state s in $\mathcal{C}$ has two copies s_0 and s_1 in $\mathcal{L}$ corresponding to the two modes 0 and 1. There are also two special states $exit_0$ and $exit_1$, from which the restarting procedure starts. Furthermore, there are a number of "intermediate" states which do not correspond to any states in $\mathcal{C}$. Below we describe how we implement the restarting procedure and the operations of sending and receiving messages. We assume mode 0. The behaviour during mode 1 can be derived by interchanging the roles of 0s and 1s. We assume that each transition of $\mathcal{L}$ has a positive weight, and that the probability of losing messages is less than 0.5. Furthermore, we will assume that if there is a self-loop containing only a send transition (in $\mathcal{L}$) then its probability is also less than 0.5.

Restarting: To implement this operation, we use two new symbols $\$_0$ and $\$_1$ not in Σ as end markers. This operation (Figure 1) resets the content of c_M. It starts from one of the exit states and empties c_M of all messages except $\$_0$ or $\$_1$. Assuming mode 0, we start from $exit_0$ and send $\$_0$ to c_M and then start receiving all messages until we receive an $\$_0$. In such a case we know that the content of c_M is a string in $\$_0^*$. The reason why we need two symbols (rather than only one), is the fact that some end markers may be left inside the channel from previous time we performed the resetting procedure. By alternating the use of the end markers (using $\$_0$ in mode 0 and $\$_1$ in mode 1), we can distinguish between the current end marker and the previous one.

Send: Suppose that we have a transition of the form $(s, c_M!m, s')$ in $\mathcal{C}$. This transition is simulated in mode 0 by the sequence of transitions as depicted in Figure 2. The empty transition from s_0 to s_{01} commites $\mathcal{L}$ to simulating the send transition $(s, c_M!m, s')$ of $\mathcal{C}$. First we generate as many copies of # as the value of c_T. To implement this we start by receiving a 0 from c_T. To avoid decreasing the value of c_T we replace the 0 by two 1s. Furthermore, to avoid decreasing the ratio between the values of the counters we send three 1s to c_S. Finally, we send a copy of # to c_M. This procedure is repeated until we receive a 1 from c_T indicating that we have consumed all the 0s in the front of c_T and thus produced all the necessary copies of #. In such a case we move to the state s_{02} and start sending copies of m in a similar manner. The aim of the self-loop, as is the case else where, from states s_{02} and s_{03} is to prevent deadlocks when trying to receive from c_T and c_S, respectively. After we have produced the required number of copies of m we move to state s_{03}. Observe that c_T will now contain a string of 0s. At state s_{03}, we reduce the value of c_S. If c_S is empty, i.e., if there is a 1 at the head of c_T, we go to state $exit_0$ to change the mode to 1, and to restart the system. Otherwise, we receive a 0 from c_S replace it by two 1s, and proceed to state s'_0, where we continue with the simulation. The reason we send two 1s is to guarantee that the value of the counter c_S for the next round (represented by 1s) will be increasing. We also need the self-loop from state s_{01} to prevent deadlock if all messages inside c_M have been lost.

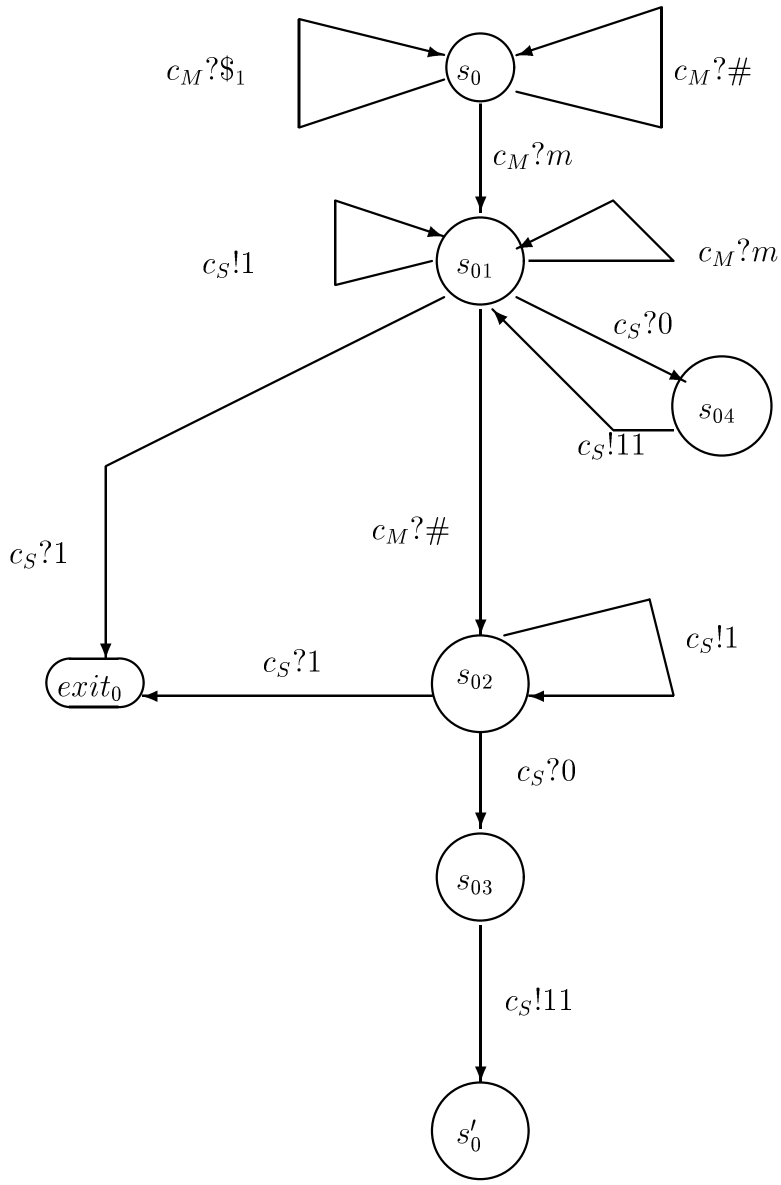
Receive: A transition $\langle s, c_M?m, s' \rangle$ is simulated as in Figure 3. We have two loops from s_0 which receive # and $\$_1$. This procedure is needed since copies of $\$_1$ may have been left from the restarting procedure (see above), and since the sending procedure produces multiple copies of #. We receive m from c_M and move to s_{01}. Observe that the system cannot deadlock at state s_0, since s_0 has at least one outgoing transition which is not a receive transition (by the assumption mentioned at the end of Section 2). From s_{01} there are two loops. In one of the loops we receive copies of m from c_M, until we receive a # indicating that all copies of m have been consumed. The other loop decreases the value of c_S in a similar manner to that described above. This prevents deadlock in case no copies of # reside at the head of c_M. Notice that this self-loop is activated even if # is available. However, we know that the value of c_S is much larger than the number of copies of m (bounded by the value of c_T), and hence all copies of these messages will be received before c_S becomes empty. In the long run, it will always be the case that the system will eventually receive a # and move to state s_{02}. In s_{02} we decrease the value of c_S, move to s'_0, and continue with the simulation.

Empty transition: The translation for an empty transitions is similar to the translation of a send transition, except that the only task of the translation is to receive a 0 from c_S (followed by an output of two 1s onto the channel c_S). Clearly, if the head of c_S is a 1 then the execution should be restarted.



Note: $c_T!11$ should be read as two send command, each of them sending an 1 to c_T.

Figure 2: Translation of send transition $\langle s, c_M!m, s'\rangle$ in mode 0



Note: $c_S!1^k$ means k sends of 1 on channel c_S.

Figure 3: Translation of receive transition $\langle s, c_M?m, s' \rangle$ in mode 0

6 Correctness

We now show the correctness of the reduction described in the previous sections. Assume that s_{init0} and s_{init1} are the start states of the two modes. Our first observation is the following: every time the system is at a restart point the probability of reaching the next restart point is 1. Formally,

Lemma 6.1 *For every* $\gamma_{i0} = (s_{init0}, w_{M0}, w_{T0}, w_0 S)$ *and* $\gamma_{i1} = (s_{init1}, w_{M1}, w_{T1}, w_{S1})$ *we have* $P(\gamma_{i0}, s_{init1}) = 1$ *and* $P(\gamma_{i1}, s_{init0}) = 1$.

Proof: By inspection of the translation we can see that any path starting from s_{init0} will reach s_{init1} in at most k steps, where k is encoded in the channel c_S. Similar argument holds for paths starting from s_{init1}. We thus have: $P(s_{init0}) = 1$ and $P(s_{init0}) = 1$. ∎

We could now extend this lemma to computations. To that end define $visit_\infty$ as follows:

Definition 6.2 Given a control state s define $visit_\infty(s)$ to be the set of computations π such that the control state s appears infinitely often in π.

From our previous lemma we now have:

Corollary 6.3 $P(visit_\infty(s_{init0})) = P(visit_\infty(s_{init1})) = 1$.

We will now concentrate on the behavior of the counters c_T and c_S. Recall from our discussions about c_T (on Page 7) that we wish it to grow unboundedly. We will now prove that our implementation indeed does satisfy that requirement. Clearly, from the construction of our PLCS system we see that whenever we remove a 0 from c_T we attempt to place two 1s in c_T. It is possible that a subset of these two messages could be lost (by our semantics). A simple calculation shows us for every 0 received the probability that the size of the chanel will increase by 1 is given by the formula $(1-\lambda)^2$. Given that we are allowed to pick a λ to make our proof work, if we assume that $\lambda < \frac{1}{2}$ then the probability of increasing the size in every cycle of action on c_T is at least $\frac{1}{2}$. Now, consider the subprocess of the entire system concentrating on the counter c_T (this amounts to taking a projection of the computation sequences onto the actions on the counter c_T); we are justified in taking such a projection because the actions on c_T are independent of actions on other queues (note that the system never deadlocks). The advantage of this view is that it can be viewed as a random walk in a single dimension. By classic results [7] on random walk we can infer that when the probability of increasing the size is greater than $\frac{1}{2}$ the probability of the system drifting away with larger sizes is 1. This observation provides us with a justification for the following:

Lemma 6.4 *For every natural number* n *we have*

$$P(\{\pi | \exists \ell : \pi(\ell) = \langle s_{init0}, w_M, w_S, w_T \rangle \wedge w_T = 0^k \wedge k > n\}) = 1$$

.

We now provide justification for the requirement on counter c_S that we expressed in Page 7, viz., that it grows faster than the counter c_T. By inspection of the construction we see that we always add more to the counter c_S than we do to c_T. To keep the fraction $\frac{w_S}{w_T}$ bounded we will have to lose increasing number of messages, whose probability, of course, tends towards zero. Thus, we have:

Lemma 6.5 *For every natural number n we have*

$$P(\{\pi | \exists \ell : \pi(\ell) = \langle s_{init0}, w_M, w_S, w_T \rangle \wedge \frac{w_S}{w_T} > n\}) = 1$$

.

■

Note that both of the lemmata above also hold for s_{init1}. It follows from Corollary 6.3 that if a control state s is reachable in a CFSM then it should be reachable with probability 1 in the corresponding PLCS. To see this let s be reachable control state in the CFSM. Then the set of execution sequences that neither visit s_0 nor s_1 can be expressed as an intersection of an infinite sequence of sets, B_i, where the i^{th} set does not take the first i opportunities to visit s_0 and s_1. Given that the probability of the intersection of the infinite sequence of sets, B_i, is the limit of an ever decreasing sequence of probabilities $P(B_i)$, we can infer that both s_0 and s_1 can only be avoided with probability 0. Formally, we have:

Lemma 6.6 *If a control state s is reachable in a CFSM then $P(\gamma_0, s_0) = 1$ and $P(\gamma_0, s_1) = 1$.*

We will now consider the other direction of our simulation. Let s be a control state which is unreachable in the $\mathcal{C}$. Now it is possible, due to the lossy nature of the buffer, that the control state s is visited. We will show that even though s might be visited in the $\mathcal{L}$ it can not be visited with probability one. To that end consider $\Pi(k) = \{\pi | \pi \mathsf{perfectafter} k\}$. Clearly, we know that these sets are not empty. We now claim that the there exists a k and a non-empty $\Pi'(k) \subseteq \Pi(k)$ such that the sequences in $\Pi'(k)$ do not visit the control state s and that $P(\Pi'(k)) > 0$. If there were no such k we would be contradicting Equation 1, which holds because the counters c_S and c_T grow unboundedly (from Lemma 6.4 and Lemma 6.5). Given that there is a set of sequences with non-zero measure that do not visit state s_0 or s_1 we can infer the following:

Lemma 6.7 *If a control state s is unreachable in a CFSM then in its corresponding PLCS the probability of visiting s_0 or s_1 is less than 1.*

This leads to the main result of the paper:

Theorem 6.8 *The problem Reach-PLCS is undecidable.*

7 Conclusion

In this paper we have shown that reachability with probability 1 is undecidable for probabilistic lossy channel systems. This in turn implies that model checking properties of lossy channel systems with probability 1 is also undecidable. The proof uses a model of loss where a message is lost as it being placed in the queue. Consequently, the probability to lose a message is bounded both from above and from below – a fact that we implicilty make use of in our proof.

The model of computation where messages can be lost after being placed in the queue is not significantly different from the model we consider here. The two models (with no-probabilistic information) yield the same reachability set (a fact that is made use of by

Abdulla and Jonsson [2]). However, a probabilistic version of the "arbitrary loss" model does not preclude transition probabilities to get arbitrarily close to 0. Thus we expect the probabilistic reachability problem to be undecidable for the arbitrary loss model too.

Acknowledgments We are grateful to Marta Kwiatkowska for prompting this work by an invitation to PROBMIV 1999.

References

[1] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996. A Preliminary Version appeared in Proc. ICALP '94, LNCS 820.

[2] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996. A Preliminary Version appeared in Proc. 8^{st} IEEE Int. Symp. on Logic in Computer Science.

[3] C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems. In Katoen, editor, *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th Int. AMAST Workshop*, volume 1601 of *Lecture Notes in Computer Science*, pages 34–52. Springer Verlag, 1999.

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

[5] Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar. On randomizatioon in sequential and distributed algorithms. *ACM Computing Surveys*, 26(1):7–86, March 1994.

[6] Purush Iyer and Murali Narasimha. Probabilistic lossy channel systems. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 667–681. Springer-Verlag, 1997.

[7] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Van Nostrand, New Jersey, 1966.

[8] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, 1976. This classic paper on probabilistic algorithms features algorithms for primality testing and nearest neighbors.