Backward coverability with pruning for lossy channel systems

Thomas Geffroy, Jérôme Leroux, Grégoire Sutre

Backward Coverability with Pruning for Lossy Channel Systems

Thomas Geffroy
Jérôme Leroux
Grégoire Sutre
University of Bordeaux & CNRS, LaBRI, UMR 5800
Talence, France

ABSTRACT

Driven by the concurrency revolution, the study of the coverability problem for Petri nets has regained a lot of interest in the recent years. A promising approach, which was presented in two papers last year, leverages a downward-closed forward invariant to accelerate the classical backward coverability analysis for Petri nets. In this paper, we propose a generalization of this approach to the class of well-structured transition systems (WSTSs), which contains Petri nets. We then apply this generalized approach to lossy channel systems (LCSs), a well-known subclass of WSTSs. We propose three downward-closed forward invariants for LCSs. One of them counts the number of messages in each channel, and the other two keep track of the order of messages. An experimental evaluation demonstrates the benefits of our approach.

CCS CONCEPTS

• **Theory of computation → Verification by model checking**; **Logic and verification**; **Invariants**; *Program verification*;

KEYWORDS

Model-Checking, Well-Structured Transition Systems, Lossy Channel Systems, Coverability Problem

ACM Reference format:
Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. 2017. Backward Coverability with Pruning for Lossy Channel Systems. In *Proceedings of SPIN'17, Santa Barbara, CA, USA, July 13-14, 2017,* 10 pages.
https://doi.org/10.1145/3092282.3092292

1 INTRODUCTION

Context. Lossy channel systems (LCSs) have been introduced by Abdulla and Jonsson to verify protocols that are designed to operate correctly even in presence of messages losses [4]. In recent years, LCSs have regained some interest because they are closely connected to weak-memory models [1, 6]. Indeed, concurrent systems operating under the TSO memory model can be simulated by LCSs, and vice versa [6]. Some business protocols [23] were also tested and improved to work correctly with unreliable channels. Another use of LCSs is to see them as over-approximations of systems with perfect channels. If a safety property is satisfied by a system with unreliable communications, then the property is also satisfied by the same system with perfect communications. The verification of a safety property on a given LCS can often be reduced to a coverability question on a (potentially larger) LCS. Coverability for LCSs is essentially the same as control-state reachability.

Related work. The coverability problem for LCSs was shown to be decidable in [4]. The precise computational complexity of the problem remained open for fifteen year. It was shown to be hyper-Ackermann-complete in [11]. This complexity could leave us hopeless for practical implementations. However, even though the coverability problem for Petri nets is ExpSpace-complete, tools were recently implemented with great success in practice [7, 15, 17, 21]. We can hope to repeat this success for LCSs since they are somewhat close to Petri nets (both are well-structured transition systems [16]).

Some tools already exist to solve the coverability problem for LCSs. The most prominent one is *TReX* [5]. The tools *LASH* [8] and *McScM* [19] are primarily designed to verify systems with perfect channels, but they can also be used for LCSs by explicitly permitting message losses. *TReX* and *LASH* iteratively compute the forward reachability set using (1) a symbolic representation of channel contents and (2) so-called *acceleration* techniques to speed-up the computation. *McScM* is based on counterexample-guided abstraction refinement [12] and abstract regular model-checking [9]. Our approach can be seen as an explicit backward search combined with a symbolic forward abstraction.

Our contribution. We present a generic backward coverability algorithm that relies on downward-closed forward invariants to prune the exploration of the state space. This algorithm works not only for LCSs but for the larger class of well-structured transition systems. We present three invariants for LCSs that can be used by this algorithm. The one we call state inequation ignores the order of messages but counts the number of occurrences of each message in each channel. The other two invariants ignore how many messages are in the channels, but focus on their order. One is based on *simple regular expressions* [2] and the other is based on quasi-orderings over messages. The second one is as an abstraction of the first one that provides an interesting trade-off between precision and efficiency. We have implemented the algorithm and the three invariants. Our experimental evaluation demonstrates the benefits of our approach.

Outline. Section 2 recalls the coverability problem for well-structured transition systems. Sections 3 and 4 present our generalized backward coverability algorithm with pruning based on downward-closed invariants. Section 5 recalls lossy channel systems and simple regular expressions. Sections 6 and 7 present two invariants that keep track of the order of messages in the channels. Section 8

SPIN'17, July 13-14, 2017, Santa Barbara, CA, USA

ACM ISBN 978-1-4503-5077-8/17/07…$15.00
https://doi.org/10.1145/3092282.3092292

presents an invariant that counts the messages in the channels. Section 9 provides our experimental evaluation. Section 10 concludes the paper.

2 COVERABILITY FOR WELL-STRUCTURED TRANSITION SYSTEMS

The class of well-structured transition systems (WSTS) is a large class of systems with many decidable properties [16]. In fact many classical problems (some termination and safety properties), can be decided with generic algorithms. The coverability problem is such a problem. It is motivated by the formal verification of some safety properties. In this section, we recall in three sub-sections (1) classical results about well quasi-orders, (2) the definition of WSTS, and (3) the coverability problem.

2.1 Well Quasi-Ordering

We first recall some properties about well quasi-orders (see [16] for additional properties and definitions).

A *quasi-ordering* over a set S is a binary relation $\leq$ over S that is transitive and reflexive. Given a subset $X \subseteq S$, we let $\uparrow X$ and $\downarrow X$ denote its *upward closure* and *downward closure*, respectively. These sets are defined as follows.

$$\uparrow X = \{s \in S \mid \exists x \in X : x \leq s\}$$
$$\downarrow X = \{s \in S \mid \exists x \in X : s \leq x\}$$

A subset $U \subseteq S$ is called an *upward-closed* set when $U = \uparrow U$, and a subset $D \subseteq S$ is called a *downward-closed* set when $D = \downarrow D$. A *basis* of an upward-closed set U is a set $B \subseteq U$ such that $\uparrow B = U$. Upward closed sets can be denoted by finite bases when $\leq$ is a well quasi-order.

A *well quasi-ordering* (wqo) over S is a quasi-ordering $\leq$ over S such that, for every infinite sequence $x_0, x_1, \ldots \subseteq S$, there exist i and j with $i < j$ and $x_i \leq x_j$. This implies that any wqo is well-founded: it admits no strictly decreasing infinite sequence.

Proposition 2.1 ([16, 20]). *For a quasi-ordering $\leq$ over a set S, the following propositions are equivalent*

- *$\leq$ is wqo.*
- *Every upward-closed set has a finite basis.*
- *Every infinite non-decreasing sequence $U_0 \subseteq U_1 \subseteq \cdots$ of upward-closed sets eventually stabilizes: there exists $k \in \mathbb{N}$ such that $U_k = U_{k+1} = U_{k+2} = \cdots$*

Notation: For the remainder of the paper, we will simply write x in place of $\{x\}$ for singletons, when it causes no confusion. In particular $\uparrow x$ denotes $\{s \in S \mid x \leq s\}$.

2.2 WSTS

We recall the definition of Well-Structured Transition Systems.

Definition 2.2 (Transition System). A *transition system* is a tuple $\mathcal{S} = (S, s_{init}, \rightarrow)$ where S is a set of *states*, s_{init} is the *initial state* and $\rightarrow \subseteq S \times S$ is the *transition relation.*

We note $\xrightarrow{*}$ the reflexive and transitive closure of $\rightarrow$.

Definition 2.3 (Well-Structured Transition System [3, 16]). A *Well-Structured Transition System* (WSTS) is a transition system $\mathcal{S} = (S, s_{init}, \rightarrow)$ equipped with a wqo $\leq$ over S that is *compatible* with $\rightarrow$: for all states s_1, s_2, t_1 with $s_1 \rightarrow s_2$ and $s_1 \leq t_1$, there exists $t_2 \in S$ such that $t_1 \xrightarrow{*} t_2$ and $s_2 \leq t_2$.

2.3 Coverability Problem

The coverability problem is a natural problem of safety verification. Formally, a state s_{final} of a WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ is said to be *coverable* if there exists a state s such that $s_{init} \xrightarrow{*} s$ and $s \geq s_{final}$. The coverability problem consist in deciding if a state is coverable. The set of coverable configurations is denoted by $Cov_{\mathcal{S}}$ and it is called the *coverability set.*

Notice that $Cov_{\mathcal{S}}$ is a downward closed set and in particular its complement is an upward closed set that admits a finite basis as any upward closed set. It follows that if B is a finite basis of that set, the coverability problem for a state s_{final} reduces to check that s_{final} is not in $\uparrow B$, i.e. check if $\bigwedge_{b \in B} b \not\leq s_{final}$. Unfortunately, for some natural classes of WSTSs (like the lossy channel systems introduced in the sequel), such a finite basis B is not computable [22].

Anyway, the coverability problem can be decided thanks to a generic algorithm. This algorithm is based on the following definitions of predecessor sets. Formally, the one-step predecessors function $pre_{\mathcal{S}}$ and the many-step predecessors function $pre_{\mathcal{S}}$ are functions from $\mathcal{P}(S)$ to $\mathcal{P}(S)$, defined for any $X \subseteq S$ by:

$$pre_{\mathcal{S}}(X) = \{s \in S \mid \exists x \in X : s \rightarrow x\}$$
$$pre^*_{\mathcal{S}}(X) = \{s \in S \mid \exists x \in X : s \xrightarrow{*} x\}$$

We recall from [16] that $pre^*_{\mathcal{S}}(U)$ is an upward closed set for any upward closed set U, and a state s_{final} is coverable if, and only if, s_{init} is in $pre^*_{\mathcal{S}}(\uparrow s_{final})$. The coverability problem is shown to be decidable for WSTSs, thanks to an algorithm computing inductively a basis of the upward closed set $pre^*_{\mathcal{S}}(\uparrow s_{final})$. In the next section we introduce a new generic algorithm for deciding the coverability problem that takes benefit of an over-approximation of the coverability sets.

3 BACKWARD COVERABILITY ANALYSIS WITH PRUNING

The coverability problem is a fundamental question for formal verification and there have been a lot of different methods proposed to solve this problem efficiently. For Petri Nets, a class of WSTS, in recent years, methods were proposed using structural analysis mixed with SMT solving [7, 15], and the use of continuous coverability in Petri Nets to accelerate the backward coverability decision [7].

Inspired by those new methods, we proposed in [17] one of the top most efficient algorithm for deciding the coverability problem for Petri nets. In that paper, the correctness of our approach relies on the following two properties satisfied by Petri nets:

- *strong compatibility*: for all states $s_1 \leq t_1$ such that $s_1 \rightarrow s_2$, there exists $t_2 \in S$ such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$.
- *upward closed predecessors* : for any state s the set of predecessors $pre_{\mathcal{S}}(\uparrow s)$ is upward-closed.

In this section we propose to generalize our approach to the full class of WSTS. The generalization is obtained by adapting proofs in

such a way we do not rely anymore on those two previously given properties.

The classical backward coverability algorithm [3, 16] for WSTS computes a growing (meaning non-decreasing in the sequel) sequence $U_0 \subseteq U_1 \subseteq \cdots$ of upward-closed subsets of S that converges to $pre^*_{\mathcal{S}}(\uparrow s_{final})$. In [17] we proposed a way to improve the convergence with the help of a known over-approximation of the coverability set. The idea is to prune U_i with the help of that over-approximation.

Formally, an *invariant* for a WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ is a downward-closed set of S that contains the coverability set. In Sections 6 to 8 we present efficient algorithms for computing useful invariants.

For the remainder of this section, we consider a WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ with a target state s_{final} and we assume that we are given an invariant I for $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$. We introduce the sequence $U_0, U_1, \ldots$ subsets of S defined as follows:

$$\begin{aligned} U_0 &= \uparrow(s_{final} \cap I) \\ U_{k+1} &= \uparrow(pre_{\mathcal{S}}(U_k) \cap I) \cup U_k \end{aligned}$$

Observe that each U_k is upward-closed and that $U_0 \subseteq U_1 \subseteq \cdots$. On the contrary to the classical backward coverability approach [3, 16], U_{k+1} does not consider all one-step predecessors of U_k, but discards those that are not in I. Note that by taking $I = S$, which is trivially an invariant, we obtain the same growing sequence as in the classical backward coverability approach [3, 16].

The two following lemmas are adapted from [17]. They show how to use this new sequence to solve the coverability problem.

Lemma 3.1. *The sequence $U_0 \subseteq U_1 \subseteq \cdots$ is ultimately stationary.*

Proof. Every infinite non-decreasing (w.r.t. $\subseteq$) sequence of upward-closed sets is ultimately stationary since $\leq$ is a wqo. □

Lemma 3.2. *It holds that $s_{final} \in Cov_{\mathcal{S}}$ if, and only if, $s_{init} \in \bigcup_k U_k$.*

Proof. If $s_{final} \in Cov_{\mathcal{S}}$ then $s_{init} \xrightarrow{*} s \geq s_{final}$ for some states s in S. Since $s_{init} \xrightarrow{*} s$, there exists $s_0, \ldots, s_n \in S$ such that $s_{init} = s_n$, $s_n \rightarrow s_{n-1} \cdots \rightarrow s_0$ and $s_0 \geq s_{final}$. First observe that $s_i \in I$ for every $i \in \{0, \ldots, n\}$ because I is an invariant for $\mathcal{S}$. Moreover, as $s_{final} \in I$ we get $U_0 = \uparrow s_{final}$. We prove, by induction on i, that $s_i \in U_i$ for all $i \in \{0, \ldots, n\}$. The basis $s_0 \in U_0$ follows from the facts that $s_0 \geq s_{final}$ and $U_0 = \uparrow s_{final}$. For the induction step, let $i \in \{0, \ldots, n-1\}$ and assume that $s_i \in U_i$. Recall that $s_{i+1} \in I$ and $s_{i+1} \rightarrow s_i$. It follows that $s_{i+1} \in (pre_{\mathcal{S}}(U_i) \cap I) \subseteq U_{i+1}$. We have shown that $s_n \in U_n$, hence, $s_{init} = s_n$ belongs to $\bigcup_k U_k$. Now, let us assume that $s_{init} \in \bigcup_k U_k$ and let us prove that $s_{final} \in Cov_{\mathcal{S}}$. We first prove, by induction on k, that $U_k \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. The basis follows from the observation that $U_0 \subseteq \uparrow s_{final} \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. For the induction step, let $k \in \mathbb{N}$ and assume that $U_k \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. Recall that $U_{k+1} = \uparrow(pre_{\mathcal{S}}(U_k) \cap I) \cup U_k$, hence, $U_{k+1} \subseteq \uparrow pre_{\mathcal{S}}(U_k) \cup U_k$. As $U_k \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$, it follows that $pre_{\mathcal{S}}(U_k) \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. As $pre^*_{\mathcal{S}}(\uparrow s_{final})$ is upward closed, it follows that $\uparrow pre_{\mathcal{S}}(U_k) \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. Hence $U_{k+1} \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$. We have thus shown that $U_k \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$ for every $k \in \mathbb{N}$. We obtain that s_{init} is in $pre^*_{\mathcal{S}}(\uparrow s_{final})$, therefore $s_{final} \in Cov_{\mathcal{S}}$. □

We have presented in this section a growing sequence of upward-closed sets whose limit contains enough information to decide the coverability problem. Our next step is to transform this sequence into an algorithm.

4 THE ALGORITHM

Now we want to design an algorithm computing with some finite bases the sequence $U_0 \subseteq U_1 \subseteq \cdots$ of upward closed sets introduced in the previous section. The classical sequence without the invariant (or with invariant $I = S$) can be computed [16] for a WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ when the relation $\leq$ is computable, and when there exists an *effective pre-basis* for this WSTS, i.e. an algorithm that can compute for every state s, a finite basis $cpre_{\mathcal{S}}(s)$ of the upward closure of $pre_{\mathcal{S}}(\uparrow s)$. It follows that the following equality holds:

$$\uparrow cpre_{\mathcal{S}}(s) = \uparrow pre_{\mathcal{S}}(\uparrow s)$$

We present the backward coverability algorithm with pruning for solving the coverability problem for any WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ with a computable $\leq$ and a computable function *cpre*.

Given a finite set X of S, we denote by $cpre_{\mathcal{S}}(X)$ the union $\bigcup_{x \in X} cpre_{\mathcal{S}}(x)$.

Lemma 4.1. *Let D be a downward-closed set of S. For every finite subset $X \subseteq S$, it holds that $\uparrow pre_{\mathcal{S}}((\uparrow X) \cap D) = \uparrow(cpre_{\mathcal{S}}(X) \cap D)$.*

Proof. We first prove that the equality $\uparrow((\uparrow Y) \cap D) = \uparrow(Y \cap D)$ holds for every finite set Y and for every downward closed set D. First of all, notice that $Y \cap D \subseteq (\uparrow Y) \cap D \subseteq \uparrow((\uparrow Y) \cap D)$. It follows that $\uparrow(Y \cap D) \subseteq \uparrow((\uparrow Y) \cap D)$. Conversely, let $s \in \uparrow((\uparrow Y) \cap D)$. There exists $d \in (\uparrow Y) \cap D$ such that $d \leq s$. As $d \in \uparrow Y$, there exists $y \in Y$ such that $y \leq d$. As D is downward closed, we derive from $y \leq d$ that $y \in D$. Thus $y \in Y \cap D$. From $y \leq d \leq s$ we get $s \in \uparrow y \subseteq \uparrow(Y \cap D)$. We have proved the other inclusion.

Now, just let $Y = cpre_{\mathcal{S}}(X)$, and observe that $pre_{\mathcal{S}}(\uparrow X) = \uparrow cpre_{\mathcal{S}}(X)$. □

We now have everything to propose the pruned backward coverability algorithm for WSTS.

Remark 4.2. Line 10 of the algorithm can be implemented just as the assignment $B \leftarrow B \cup P$. However, in order to improve the efficiency of the algorithm, we remove non minimal elements as follows. An element x in a set X of states is said to be *redundant* if there exists $y \in X$ with $y \neq x$ such that $y \leq x$. Notice that in that case $\uparrow X = \uparrow Y$ where $Y = X \backslash \{x\}$. Line 10 of the algorithm can be implemented just by inductively removing redundant states one by one. When $\leq$ is a partial order, this computation returns the set of minimal elements. □

Let $\ell_B, \ell_P \in \mathbb{N} \cup \{\infty\}$ denote the numbers of executions of lines 5 and 8, respectively. It is understood that $l_P \leq l_B \leq l_P + 1$, with the convention that $\infty + 1 = \infty$. Let $(B_k)_{k < \ell_B}$ and $(P_k)_{k < \ell_P}$ denote the successive values at lines 5 and 8 of the variables B and P, respectively.

ALGORITHM 1: BCwP$(\mathcal{S}, s_{final}, I)$.

Input: A WSTS $\mathcal{S} = (S, s_{init}, \rightarrow, \leq)$ with $\leq$ computable and an effective pre-basis, a target state $s_{final} \in S$ and a downward-closed invariant I for $\mathcal{S}$
Output: Whether there exists a state $s \in S$ such that $s_{init} \xrightarrow{*} s$ and $s \geq s_{final}$.

```
1  if s_final ∈ I then
2  |  B ← {s_final}
3  else
4  |  B ← ∅
5  while s_init ∉ ↑B do
6  |  N ← (cpre_S(B)) \ ↑B            /* new predecessors */
7  |  P ← N ∩ I                       /* prune uncoverable states */
8  |  if P = ∅ then
9  |  |  return False
10 |  Let B be a finite basis of ↑(B ∪ P)
11 return True
```

Lemma 4.3. *For every k with $0 \leq k < \ell_B$, the set B_k is a finite basis of U_k. For every k with $0 \leq k < \ell_P$, the set P_k is a finite basis of $\uparrow(U_{k+1} \setminus U_k)$.*

Proof. It is readily seen that B_k and P_k are finite subsets of *states* for every k. We first observe that, for every k with $0 \leq k < \ell_P$,

$$\begin{aligned} \uparrow P_k &= \uparrow((cpre_{\mathcal{S}}(B_k) \setminus \uparrow B_k) \cap I) && \text{[Lines 6–7]} \\ &= \uparrow(cpre_{\mathcal{S}}(B_k) \cap (I \setminus \uparrow B_k)) \\ &= \uparrow(pre_{\mathcal{S}}(\uparrow B_k) \cap (I \setminus \uparrow B_k)) && \text{[Lemma 4.1]} \\ &= \uparrow((pre_{\mathcal{S}}(\uparrow B_k) \cap I) \setminus \uparrow B_k) \end{aligned}$$

Let us now prove, by induction on k, that $U_k = \uparrow B_k$ for every k with $0 \leq k < \ell_B$. The basis $U_0 = \uparrow B_0$ follows from lines 1–5 of BCwP and from the definition of U_0. For the induction step, let $k \in \mathbb{N}$ with $k + 1 < \ell_B$, and assume that $U_k = \uparrow B_k$. Line 10 entails that $\uparrow B_{k+1} = \uparrow B_k \cup \uparrow P_k$. It follows that

$$\begin{aligned} \uparrow B_{k+1} &= \uparrow B_k \cup \uparrow P_k \\ &= \uparrow B_k \cup \uparrow((pre_{\mathcal{S}}(\uparrow B_k) \cap I) \setminus \uparrow B_k) \\ &= U_k \cup \uparrow((pre_{\mathcal{S}}(U_k) \cap I) \setminus U_k) && [U_k = \uparrow B_k] \\ &= U_k \cup \uparrow(pre_{\mathcal{S}}(U_k) \cap I) \\ &= U_{k+1} \end{aligned}$$

This concludes the proof that $U_k = \uparrow B_k$ for every k with $0 \leq k < \ell_B$. Moreover, coming back to the characterization of $\uparrow P_k$, we get that

$$\begin{aligned} \uparrow P_k &= \uparrow((pre_{\mathcal{S}}(\uparrow B_k) \cap I) \setminus \uparrow B_k) \\ &= \uparrow((pre_{\mathcal{S}}(U_k) \cap I) \setminus U_k) \\ &= \uparrow(U_{k+1} \setminus U_k) \end{aligned}$$

for every k with $0 \leq k < \ell_P$. □

Theorem 4.4. *The procedure* BCwP *terminates on every input and is correct.*

Proof. Let us first prove termination. We need to show that any maximal execution of BCwP$(\mathcal{S}, s_{final}, I)$ is finite. By contradiction, assume that $\ell_B = \infty$. According to Lemma 3.1, there exists an index $h \in \mathbb{N}$ such that $U_h = U_{h+1}$. We derive from Lemma 4.3 that $P_h = \emptyset$. Therefore, the execution should terminate at line 9 during the $(h+1)^{\text{th}}$ iteration of the **while** loop. This contradicts our assumption that $\ell_B = \infty$.

We now turn our attention to the correctness of BCwP. As it is finite, any maximal execution of BCwP$(\mathcal{S}, s_{final}, I)$ either returns False at line 9 or returns True at line 11.

- If it returns False then $P_{\ell_P - 1} = \emptyset$ and it follows from Lemma 4.3 that $U_{\ell_P} \subseteq U_{\ell_P - 1}$. It follows that $U_{\ell_P - 1} = \bigcup_k U_k$. Moreover, $s_{init} \notin \uparrow B_{\ell_P - 1}$ because the condition of the **while** loop had to hold. It follows hat $s_{init} \notin \bigcup_k U_k$. We derive from Lemma 3.2 that $s_{final} \notin Cov_{\mathcal{S}}$.
- If it returns True then $s_{init} \in \uparrow B_{\ell_B - 1}$ and it follows from Lemma 4.3 that $s_{init} \in U_{\ell_B - 1} \subseteq \bigcup_k U_k$. We derive from Lemma 3.2 that $s_{final} \in Cov_{\mathcal{S}}$.

This concludes the proof of the theorem. □

In the next section we focus our algorithm on lossy channel systems.

5 LOSSY CHANNEL SYSTEMS

Lossy channel systems (LCSs) have been introduced by Abdulla and Jonsson to verify protocols that are designed to operate correctly even in presence of messages losses [4]. Informally, an LCS is a finite-state automaton equipped with finitely many first-in first-out channels that are unreliable in the sense that messages can be lost at any time. This section recalls the definition of LCS and shows how to verify them using the BCwP algorithm of the previous section.

5.1 Syntax and Semantics of LCSs

A *lossy channel system* is a tuple $\mathcal{S} = (Q, q_{init}, M, d, \Delta)$ where Q is the finite set of *locations*, $q_{init} \in Q$ is the *initial location*, M is the finite alphabet of *messages* that can be exchanged over the channels, d is the *number of channels*, and $\Delta \subseteq Q \times Op \times Q$ is the set of transition *rules*, where $Op = \{1, \ldots, d\} \times \{!, ?\} \times M$ is the set of *operations* over the channels. An operation is

- either a *transmission* $i!m$ of a message m into a channel i,
- or a *reception* $i?m$ of a message m from a channel i.

Example 5.1. Figure 1 represents the LCS with locations $Q = \{q_1, q_2, q_3, q_{bad}\}$, initial location $q_{init} = q_1$, messages $M = \{a, b\}$, a single channel, and rules $\Delta = \{q_1 \xrightarrow{1!a} q_2, q_2 \xrightarrow{1!b} q_1, q_2 \xrightarrow{1?a} q_3, q_3 \xrightarrow{1?a} q_{bad}\}$. □

The operational *semantics* of an LCS $\mathcal{S} = (Q, q_{init}, M, d, \Delta)$ is given by an infinite-state transition system $[\![\mathcal{S}]\!] = (S, s_{init}, \rightarrow)$ defined as follows. The set of states is $S = Q \times (M^*)^d$. So a state $s = (q, w_1, \ldots, w_d)$ is composed of a location q and of words w_i describing the contents of the channel i. All channels are empty initially, so the initial state is $s_{init} = (q_{init}, \varepsilon, \ldots, \varepsilon)$.

For each rule $\delta = (q, op, q')$ in Δ, we define the binary relation $\xrightarrow{\delta}$ over S as follows: $s \xrightarrow{\delta} s'$ with $s = (q, w_1, \ldots, w_d)$ and $s' = (q', w'_1, \ldots, w'_d)$ if and only if

- either op is a transmission $i!m$, in which case $w'_i = w_i \cdot m$ and $w'_j = w_j$ for all $j \neq i$,
- or op is a reception $i?m$, in which case $m \cdot w'_i = w_i$ and $w'_j = w_j$ for all $j \neq i$.

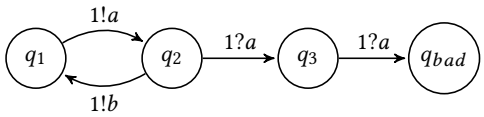

Figure 1: Simple example of a lossy channel system.

The relations $\xrightarrow{\delta}$ capture the perfect semantics of first-in first-out channels. To capture message losses, we also define the binary relation $\rightarrow_\lambda$ over S. Formally, $s \rightarrow_\lambda s'$ with $s = (q, w_1, \ldots, w_d)$ and $s' = (q', w'_1, \ldots, w'_d)$ if and only if the locations q and q' are the same and there exists a channel $i \in \{1, \ldots, d\}$ such that w'_i is obtained from w_i by deleting one message (i.e., $w_i = u \cdot m \cdot v$ and $w'_i = u \cdot v$, for some $m \in M$ and $u, v \in M^*$) and $w'_j = w_j$ for all $j \neq i$. For example, $(q_2, a \cdot b \cdot a) \rightarrow_\lambda (q_2, a \cdot a)$. We also introduce, for technical reasons, the identity relation $\{(s, s) \mid s \in S\}$, written $\rightarrow_\varepsilon$.

Now, the binary relation $\rightarrow$ over S is defined to be the union of the relation $\rightarrow_\varepsilon$, of the relation $\rightarrow_\lambda$ and of the relations $\xrightarrow{\delta}$ where δ ranges over Δ.

Example 5.2. Consider again the LCS depicted in Figure 1. The state (q_{bad}, ε) is reachable from the initial state because $(q_1, \varepsilon) \rightarrow (q_2, a) \rightarrow\!\rightarrow (q_2, aba) \rightarrow_\lambda (q_2, aa) \rightarrow\!\rightarrow (q_{bad}, \varepsilon)$. But a message loss (i.e., a $\rightarrow_\lambda$ step) is required to do so. □

5.2 LCSs Are Well-Structured

As was already observed in [3, 16], lossy channel systems are well-structured, and so coverability is decidable for them. The well-structure of LCSs allows us to use the algorithm presented in Section 4. But we need to show that the required effectivity conditions are fulfilled. Let us start by recalling how LCSs can be framed as well-structured transition systems.

For two words u and v in M^*, we say that u is a *subword* of v, written $u \preceq v$, if u can be obtained from v by erasing some letters. In other words, u is a (possibly non-contiguous) subsequence of v. The relation $\preceq$ is obviously a partial ordering[1] on M^*. Moreover, it is wqo on M^*. This last result is due to Higman [20].

Given an LCS $\mathcal{S} = (Q, q_{init}, M, d, \Delta)$, we define the relation $\leq$ over $S = Q \times (M^*)^d$ as follows. For two states $s = (q, w_1, \ldots, w_d)$ and $s' = (q', w'_1, \ldots, w'_d)$, let $s \leq s'$ if and only if $q = q'$ and w_i is a subword of w'_i for every $i \in \{1, \ldots, d\}$. It is readily seen that this relation is a partial ordering on S. The following lemma is an easy consequence of the definitions of $\rightarrow_\lambda$ and $\leq$.

Lemma 5.3. *The relations $\geq$ and $(\rightarrow_\lambda)^*$ are equal.*

It follows from the previous lemma that coverability and reachability coincide for lossy channel systems (equipped with $\leq$). Indeed, if there exists a state $s \in S$ such that $s_{init} \xrightarrow{*} s \geq s_{final}$ then $s_{init} \xrightarrow{*} s \xrightarrow{*} s_{final}$ hence $s_{init} \xrightarrow{*} s_{final}$. The converse always holds by reflexivity of $\leq$.

Lemma 5.4 ([3, 16]). *The semantics of a lossy channel system, equipped with the partial order $\leq$, is a WSTS.*

[1]Recall that a *partial ordering* on a set S is binary relation on S that is reflexive, transitive and antisymmetric.

$$cpre_!(a, abc) = abc \qquad cpre_?(a, \varepsilon) = a$$
$$cpre_!(a, abca) = abc \qquad cpre_?(c, abb) = cabb$$

Figure 2: Example of $cpre_!$ and $cpre_?$.

Proof. Since the set Q of locations is finite, equality is a wqo on Q. Recall that $\preceq$ is a wqo on M^*. It follows that $\leq$ is a wqo on S. It remains to prove that $\leq$ is compatible with the relation $\rightarrow$. Consider three states s, s' and t such that $s \rightarrow s'$ and $s \leq t$. By Lemma 5.3, it holds that $t \xrightarrow{*} s$. Therefore, $t \xrightarrow{*} s \rightarrow s' = t'$, which concludes the proof that $\leq$ is compatible with $\rightarrow$. □

As mentioned previously, we want to solve the coverability problem for lossy channel systems using the BCwP algorithm of Section 4. But several effectivity conditions need to be fulfilled in order to do so, even if we use the trivial invariant $I = S$.

Firstly, we observe that the partial order $\leq$ is computable because there are only finitely many channels and $\preceq$ itself is computable.

Secondly, we need to show LCSs admit an effective pre-basis, meaning that there exists an algorithm that, given a state s, computes a finite basis of $\uparrow pre_{\mathcal{S}}(\uparrow s)$. Toward this end, we introduce two functions $cpre_!$ and $cpre_?$ from $M \times M^*$ to M^* defined as follows:

$$cpre_!(m, w) = \begin{cases} w' & \text{if } w = w' \cdot m \\ w & \text{otherwise} \end{cases}$$

$$cpre_?(m, w) = m \cdot w$$

See examples in Figure 2. Now, for a state $s = (q, w_1, \ldots, w_d)$ and a rule $\delta = (p, i\#m, q) \in \Delta$, with $\# \in \{!, ?\}$, we let $cpre_\delta(s) = (p, w_1, \ldots, w_{i-1}, cpre_\#(m, w_i), w_{i+1}, \ldots w_d)$. It is readily seen that $cpre_\delta$ is computable. The following lemma shows that LCSs admit an effective pre-basis.

Lemma 5.5. *For every state $s = (q, w_1, \ldots, w_d)$ in S, the set*

$$\{s\} \cup \{cpre_\delta(s) \mid \delta = (p, op, q) \in \Delta\}$$

is a finite basis of $\uparrow pre_{\mathcal{S}}(\uparrow s)$.

Thirdly, line 10 of the BCwP algorithm simply amounts to the computation of the minimal elements of $B \cup P$.

We now have a way to solve the coverability problem for LCSs. To accelerate the computation we need good invariants. The next sections aim to provide that.

5.3 Simple Regular Expressions

The BCwP algorithm of Section 4 makes use of downward-closed invariants in order to accelerate the classical backward coverability analysis. Invariants are potentially infinite sets of states. So before attempting to generate good invariants, we need a finite representation for them. As there are only finitely many locations, the complexity comes from channel contents. These are d-tuples of words over the finite alphabet M of messages. A natural and simple approach consists in using recognizable subsets of $(M^*)^d$ to represent (potentially infinite) sets of channel contents. This is a reasonable limitation since the coverability set of an LCS, which is its most precise downward-closed invariant, is recognizable [4, 10]. By Mezei's theorem, recognizable subsets are finite unions of cartesian products of regular languages over the alphabet M. Since we are

interested in downward-closed invariants, we focus on downward-closed regular languages. Such languages can be represented by *simple regular expressions* [2].

Let us recall the definition of simple regular expressions. An *atom* is either $(\varepsilon + m)$ for a message m in M or $(m_1 + \cdots + m_n)^*$ where $m_1, \ldots, m_n$ are messages in M with $n \geq 1$. A *product* is a finite concatenation $a_1 \cdots a_n$ of atoms. A *simple regular expression* (SRE) is a finite sum $p_1 + \cdots + p_n$ of products.

The *language of an SRE* r is the regular language associated with it and will be denoted by $[\![r]\!]$. The following lemma gives us a reason to use SREs to represent downward-closed regular languages.

Lemma 5.6 ([2]). *A language $L \subseteq M^*$ is downward-closed if and only if it can be represented by an SRE.*

In practice, it is desirable for efficiency reasons to manipulate SREs that do not contain redundancies. We say that a product $p = e_1 \cdots e_n$ is *in normal form* [2] if and only if for all $1 \leq i \leq n$, we have $[\![e_i \cdot e_{i+1}]\!] \not\subseteq [\![e_{i+1}]\!]$ and $[\![e_i . e_{i+1}]\!] \not\subseteq [\![e_i]\!]$. An SRE $p_1 + \ldots + p_n$ is *in normal form* if all products p_i are in normal forms and there are no products p_i and p_j with $i \neq j$ such that $[\![p_i]\!] \subseteq [\![p_j]\!]$.

Lemma 5.7 ([2]). *For every SRE r, there is a unique (up to commutativity of +) SRE in normal form, which is denoted by* $\mathrm{nf}(r)$*, such that $[\![\mathrm{nf}(r)]\!] = [\![r]\!]$. Furthermore,* $\mathrm{nf}(r)$ *can be computed from r in quadratic time.*

We can see that there exist infinite increasing sequences of SREs. This means that we cannot directly use SREs to compute downward-closed invariants via Kleene iteration. Let us illustrate this issue on the LCS of Figure 1. Applying Kleene iteration to the loop $q_1 \xrightarrow{1!a} \xrightarrow{1!b} q_1$ leads to the sequence ε, $(a + \varepsilon) \cdot (b + \varepsilon)$, $(a+\varepsilon)\cdot(b+\varepsilon)\cdot(a+\varepsilon)\cdot(b+\varepsilon)$, etc. There exist acceleration techniques that can derive the effect of an arbitrary iteration of the loop [2], and come up with $(a+b)^*$. But, in general, the coverability set of an LCS cannot be computed [22] even though it is downward-closed and, hence, recognizable. Therefore, we settle for over-approximations of the coverability set. In the next section, we will introduce a subclass of SREs, that we call *compact simple regular expressions*, and show how to use it to compute downward-closed invariants.

6 INVARIANT USING COMPACT SRES

Despite the assumption that messages may be lost, all messages that remain in a channel are in the same order as the order in which they were sent. If a system sends some messages a and then some messages b in a channel, this channel can't contain a word that contains a message b after a message a. In this section, we propose an abstraction of channel contents that focuses on this kind of properties, and ignores the numbers of occurrences of messages. For instance, this abstraction provides properties of the type: it's not possible to have a message a after a message b.

We present an invariant generation technique that uses a subclass of SREs, that we call *compact simple regular expressions*. The idea is to abstract the contents of a channel with simple expressions of the form $(a^* \cdot b^*) + (a^* \cdot (c + d)^*)$.

Formally, a *compact simple regular expression* (CSRE) is an SRE $\sum p_i$ where every product p_i is a compact product. A *compact product* is of the form $a_1 \cdots a_n$ with every atom a_i of the form $(m_1 + \cdots + m_n)^*$ and such that every two atoms a_i and a_j with $i \neq j$ have distinct messages (i.e., $[\![a_i]\!] \cap [\![a_j]\!] = \{\varepsilon\}$). Some examples of CSREs are given in Figure 3. We see that every compact product is in normal form. However, not all CSREs are in normal forms. We note $CSRE_{nf}$ the set of CSREs in normal form. It is readily seen that this set is finite. The CSREs of Figure 3 are all in normal form.

$$csre_1 = (a^* \cdot b^*) + (a^* \cdot c^*) \qquad csre_3 = \varepsilon$$
$$csre_2 = ((a + b)^*) + ((c + d)^* \cdot e^*) \qquad csre_4 = \emptyset$$

Figure 3: Examples of compact simple regular expressions.

We define the binary relation $\sqsubseteq$ over $CSRE$ by $r_1 \sqsubseteq r_2$ if and only if $[\![r_1]\!] \subseteq [\![r_2]\!]$. This relation is reflexive and transitive hence it is a quasi-ordering. The relation $\sqsubseteq$ is also antisymmetric for CSREs in normal form, therefore $\sqsubseteq$ is a partial order over $CSRE_{nf}$. Note that $\sqsubseteq$ is not antisymmetric over the full class of CSREs (e.g., $[\![(a + b)^*]\!] = [\![(a + b)^* + a^*]\!]$).

We will exhibit a Galois connection between $(\mathcal{P}(M^*), \subseteq)$ and $(CSRE_{nf}, \sqsubseteq)$. The two following lemmas are needed to do so.

Lemma 6.1. *Consider two subsets of messages $A_1, A_2 \subseteq M$. For every languages $L_1 \subseteq (M \setminus A_1)^*$ and $L_2 \subseteq (M \setminus A_2)^*$, it holds that $(A_1^* \cdot L_1) \cap (A_2^* \cdot L_2) = (A^* \cdot (L_1 \cap B_2^* \cdot L_2)) \cup (A^* \cdot (L_2 \cap B_1^* \cdot L_1))$ where $A = A_1 \cap A_2$, $B_1 = A_1 \setminus A_2$ and $B_2 = A_2 \setminus A_1$.*

Proof. We only prove $\subseteq$ since the converse inclusion is immediate. Let $w \in (A_1^* \cdot L_1) \cap (A_2^* \cdot L_2)$. There exists $u_1 \in A_1^*$, $v_1 \in L_1$, $u_2 \in A_2^*$ and $v_2 \in L_2$ such that $w = u_1 \cdot v_1 = u_2 \cdot v_2$. Assume that the length of u_1 is smaller than or equal to the length of u_2. There exists u_2' such that $u_2 = u_1 \cdot u_2'$. It follows that $u_1 \in (A_1 \cap A_2)^* = A^*$. Observe also that $v_1 = u_2' \cdot v_2$. Since $v_1 \in L_1 \subseteq (M \setminus A_1)^*$, we get that $u_2' \in (A_2 \setminus A_1)^* = B_2^*$, and so $v_1 \in (B_2^* \cdot L_2)$. We have shown that $w \in (A^* \cdot (L_1 \cap B_2^* \cdot L_2))$. Symmetrically, if the length of u_1 is larger than or equal to the length of u_2, then $w \in (A^* \cdot (L_2 \cap B_1^* \cdot L_1))$. □

Lemma 6.2. *The set of languages that can be represented by CSREs is closed under intersection.*

Proof. We show that the intersection of the languages of two compact products can be represented by a CSRE. The lemma then follows by distributivity of intersection over union and by closure of CSREs under sum. For a compact product $p = a_1 \cdots a_n$, we write $|p|$ the number n of atoms composing p. We prove by induction on $k \geq 0$, that for all compacts products p_1 and p_2 such that $|p_1| + |p_2| \leq k$, there exists a CSRE r such that $[\![p_1]\!] \cap [\![p_2]\!] = [\![r]\!]$. For the basis step, we have $p_1 = p_2 = \varepsilon$ and $[\![p_1]\!] \cap [\![p_2]\!] = [\![\varepsilon]\!]$. For the induction step, let p_1 and p_2 two compact products such that $|p_1| + |p_2| = k + 1$. If one them is equal to ε then $[\![p_1]\!] \cap [\![p_2]\!] = [\![\varepsilon]\!]$. Otherwise, p_1 and p_2 may be written as the compact products $p_1 = A_1^* \cdot p_1'$ and $p_2 = A_2^* \cdot p_2'$ where A_1 and A_2 are two subsets of M. Note that $[\![p_1']\!] \subseteq (M \setminus A_1)^*$ and $[\![p_2']\!] \subseteq (M \setminus A_2)^*$. We get from Lemma 6.1 that $[\![p_1]\!] \cap [\![p_2]\!] = (A^* \cdot ([\![p_1']\!] \cap [\![B_2^* \cdot p_2']\!])) \cup (A^* \cdot ([\![p_2']\!] \cap [\![B_1^* \cdot p_1']\!]))$ where A, B_1 and B_2 are defined as in the lemma. Observe that $|p_1'| + |B_2^* \cdot p_2'| = |p_2'| + |B_1^* \cdot p_1'| = k$. It follows from the induction hypothesis that there exists two CSREs r_1 and r_2 such that $[\![r_1]\!] = ([\![p_1']\!] \cap [\![B_2^* \cdot p_2']\!])$ and $[\![r_2]\!] = ([\![p_2']\!] \cap [\![B_1^* \cdot p_1']\!])$. We obtain that $[\![p_1]\!] \cap [\![p_2]\!] = [\![A^* \cdot r_1 + A^* \cdot r_2]\!]$. It is routinely checked that $[\![r_1]\!]$

$$a^* \cdot b^* \cap b^* \cdot a^* \;=\; a^* \cup b^* \qquad a^* \cdot b^* \cap (a+b)^* \;=\; a^* \cdot b^*$$

Figure 4: Examples of intersections of compact products.

and $[\![r_2]\!]$ are contained in $(M \setminus A)^*$. Therefore, $A^* \cdot r_1$ and $A^* \cdot r_2$ can be rewritten as CSREs by distributivity of product over sum. □

See examples of the intersection of CSREs in Figure 4. Let us introduce the concretisation function $\gamma : CSRE_{nf} \to \mathcal{P}(M^*)$ defined by $\gamma = [\![\cdot]\!]$. The corresponding abstraction function $\alpha : \mathcal{P}(M^*) \to CSRE_{nf}$ is defined by $\alpha(L) = \mathrm{nf}(\bigcap\{r \in CSRE_{nf} \mid L \subseteq [\![r]\!]\})$. The function α is well-defined because $CSRE_{nf}$ is closed by intersection and is finite. Note that $\alpha(L)$ is computable when L is a regular language. We observe that (α, γ) is a Galois connection between $(\mathcal{P}(M^*), \subseteq)$ and $(CSRE_{nf}, \sqsubseteq)$. In order to apply the classical framework of abstract interpretation [13], it remains to show that we can perform abstractly the operations over the channels, namely transmissions and receptions.

For a language $L \subseteq M^*$, let $m^{-1}L = \{w \subseteq M^* \mid m \cdot w \in L\}$. Following the classical framework of abstract interpretation, an abstract transmission is defined by $(!m)^\#(r) = \alpha([\![r]\!] \cdot m)$ and an abstract reception is defined by $(?m)^\#(r) = \alpha(m^{-1} \cdot [\![r]\!])$. Let us show how to compute $(!m)^\#(r)$ and $(?m)^\#(r)$ given a CSRE r in normal form. We only need to consider the case where r is a compact product since $(!m)^\#(p_1 + \cdots + p_n) = \mathrm{nf}((!m)^\#(p_1) + \cdots + (!m)^\#(p_n))$, and similarly for $(?m)^\#$.

For a compact product $p = a_1 \cdots a_n$, let $merge(a_1 \cdots a_n)$ denote the atom that contains exactly the messages that appear in $a_1, \ldots, a_n$. Formally, $merge(a_1 \cdots a_n)$ is the atom A^* where $A = \{m \in M \mid m \in [\![a_1]\!] \cup \cdots [\![a_n]\!]\}$. We have for a compact product $p = a_1 \cdots a_n$ and a message $m \in M$:

$$(!m)^\#(p) \;=\; \begin{cases} a_1 \cdots a_{k-1} \cdot merge(a_k \cdots a_n) & \text{if } m \in [\![p]\!] \\ p \cdot m^* & \text{otherwise} \end{cases}$$

$$(?m)^\#(p) \;=\; \begin{cases} a_k \cdots a_n & \text{if } m \in [\![p]\!] \\ \emptyset & \text{otherwise} \end{cases}$$

where, in both cases, k the unique index such that $m \in [\![a_k]\!]$.

By a standard construction, we can extend the Galois connection (α, γ) into a Galois connection between $(\mathcal{P}(S), \subseteq)$ and the abstract domain consisting of the set of maps from Q to $CSRE^d_{nf}$, partially ordered by the component-wise extension of $\sqsubseteq$. So we can apply the classical framework of abstract interpretation [13] to generate an inductive downward-closed invariant by a standard fixpoint computation in this abstract domain. Since the set $CSRE_{nf}$ is finite, the fixpoint computation is guaranteed to converge. We may then use this downward-closed invariant to accelerate the backward coverability analysis, as is done in the BCwP algorithm of Section 4. However, this acceleration comes at a price: the generation of the invariant itself might be costly. We illustrate this issue in the following example.

Example 6.3. Consider an alphabet M containing the distinct messages $x_1, y_1, z_1, \ldots, x_n, y_n, z_n$. Let $L \subseteq M^*$ be the language

$$L \;=\; {x_1}^* \cdot ({y_1}^* + {z_1}^*) \cdot {x_2}^* \cdot ({y_2}^* + {z_2}^*) \cdots {x_n}^* \cdot ({y_n}^* + {z_n}^*)$$

This language can be represented by a CSRE. But the minimal CSRE r with $L = [\![r]\!]$, is the sum of all compact products of the form ${x_1}^* \cdot {m_1}^* \cdot {x_2}^* \cdot {m_2}^* \cdots {x_n}^* \cdot {m_n}^*$ where $m_i \in \{y_i, z_i\}$ for all $i \in \{1, \ldots, n\}$. There are exponentially many such products. Observe that r is in normal form. □

The potential exponential blow-up illustrated in Example 6.3 may limit, in practice, the usefulness of our invariant generation technique based on CSREs. See Section 9 for experimental results. The next section aims at solving this problem.

7 INVARIANT USING ORDERING FLOWS

In the last section, we saw how to generate downward-closed invariants by an abstraction of channel contents that ignores the numbers of occurrences of messages and only cares about their respective order. This abstraction relies on so-called compact simple regular expressions (CSREs). However, the generation of this invariant might be costly in practice. This comes from the fact that CSREs are, in fact, not compact enough (see Example 6.3).

We now present an invariant generation technique based on quasi-orderings (i.e. transitive and reflexive relations) over messages, that we call *message ordering flow* (MOFs). The goal is still the same: we want to keep track of which messages can be in which order in each channel. But we want an invariant that is faster to compute than the one based on CSREs.

Let M denote the set of messages of an LCS for which we want to compute a downward-closed invariant. A *message ordering flow* (MOF) is a pair (A, R) where $A \subseteq M$ and R is quasi-ordering over A. We let MOF denote the set of MOFs. We define the partial ordering $\sqsubseteq$ over MOF by $F_1 \sqsubseteq F_2$ with $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$ if and only if $A_1 \subseteq A_2$ and $R_1 \subseteq R_2$.

The least upper bound of two MOFs $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$ is $F_1 \sqcup F_2 = (A, R)$ where $A = A_1 \cup A_2$ and $R = (R_1 \cup R_2)^+$, i.e., the transitive closure of $R_1 \cup R_2$. The greatest lower bound of F_1 and F_2 is $F_1 \sqcap F_2 = (A, R)$ where $A = A_1 \cap A_2$ and $R = R_1 \cap R_2$.

As in Section 6, we want to exhibit a Galois Connection between $(\mathcal{P}(M^*), \subseteq)$ and $(MOF, \sqsubseteq)$. Let us start with the abstraction function $\alpha : \mathcal{P}(M^*) \to MOF$. For a word $w \in M^*$, let $\alpha(w) = (A, R) \in MOF$ where $A = \{m \in M \mid m \preceq w\}$ is the set of all messages occurring in w and $R = \{(a, b) \in A \times A \mid a = b \vee ab \preceq w\}$. Recall that $\preceq$ is the sub-word partial order on M^*. For a non-empty language $L \subseteq M^*$, we define $\alpha(L)$ to be the least upper bound of $\{\alpha(w) \mid w \in L\}$. The concretisation function $\gamma : MOF \to \mathcal{P}(M^*)$ is defined by $\gamma(A, R) = \{w \in A^* \mid \forall ab \preceq w, (a, b) \in R\}$.

It is readily seen that the concretisation of a MOF is a downward-closed language. But the set $\gamma(MOF)$ of languages that can be represented by a MOF misses an important language, namely the empty language. So we extend the set MOF with a new element, written $\bot$, whose concretisation is $\gamma(\bot) = \emptyset$. Conversely, the abstraction of the empty language is $\alpha(\emptyset) = \bot$. We also extend $\sqsubseteq$ in the obvious way. For every MOF F, we have $\bot \sqcup F = F$ and $\bot \sqcap F = \bot$.

Observe that, by construction, (α, γ) is a Galois connection between $(\mathcal{P}(M^*), \subseteq)$ and $(MOF, \sqsubseteq)$. As in Section 6, in order to apply the classical framework of abstract interpretation [13], it remains to show that we can perform abstractly the operations over the channels, namely transmissions and receptions.

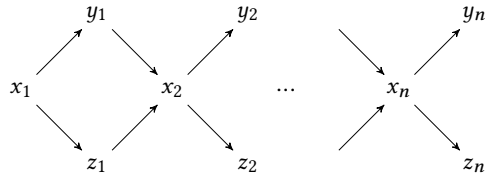

Figure 5: MOF for the language L of Example 6.3.

Following the classical framework of abstract interpretation, we define $(!m)^\# : MOF \to MOF$ and $(?m)^\# : MOF \to MOF$ by $(!m)^\#(F) = \alpha(\gamma(F) \cdot m)$ and $(?m)^\#(F) = \alpha(m^{-1} \cdot \gamma(F))$. Let us show that the functions $(!m)^\#$ and $(?m)^\#$ are computable.

- $(!m)^\#(\bot) = (?m)^\#(\bot) = \bot$.
- $(!m)^\#(A, R) = (A', R')$ where $A' = A \cup \{m\}$ and $R' = R \cup (A \times \{m\}) \cup (B \times B)$ where $B = \{b \in A \mid (m, b) \in R\}$ is the set of all messages that can appear after m.
- $(?m)^\#(A, R) = \bot$ if $m \notin A$, otherwise $(?m)^\#(A, R) = (A', R')$ where $A' = \{b \in A \mid (m, b) \in R\}$ is the set of all messages that can appear after m and $R' = R \cap (A' \times A')$.

As with CSREs, we can extend the Galois connection (α, γ) into a Galois connection between $(\mathcal{P}(S), \subseteq)$ and the abstract domain consisting of the set of maps from Q to MOF^d, partially ordered by the component-wise extension of $\sqsubseteq$. Again, we can generate an inductive downward-closed invariant by a standard fixpoint computation in this abstract domain. Since the set MOF is finite, the fixpoint computation is guaranteed to converge. We may then use this downward-closed invariant in the BCwP algorithm of Section 4.

On the contrary to CSREs that can suffer from an exponential blow-up, each MOF is of size at most quadratic in the cardinal of the alphabet M of messages of the LCS under analysis.

Example 7.1. Let us revisit Example 6.3. Recall that the language L defined there is

$$L = {x_1}^* \cdot ({y_1}^* + {z_1}^*) \cdot {x_2}^* \cdot ({y_2}^* + {z_2}^*) \cdots {x_n}^* \cdot ({y_n}^* + {z_n}^*)$$

This language can be represented by the MOF $F = (A, R)$ where A contains exactly the messages $x_1, y_1, z_1, \ldots, x_n, y_n, z_n$ and R respects the quasi-ordering shown in Figure 5, that is, R is the reflexive and transitive closure of the arrows depicted in that figure. On the contrary to the minimal CSRE representing L, which is exponential in n, the size of F is quadratic in n. □

8 INVARIANT WITH STATE INEQUATION

The last two invariants focused on the order of messages in the channels. Neither invariant takes into account the number of messages. We now present an invariant based on the Petri net state equation. It is a simple invariant that just counts the number of messages that can be in each channel. The state equation was recently used successfully for solving Petri net coverability problems [7, 15, 17].

Inspired by the state equation for Petri nets, we associate to an LCS and a state $s_{final} = (q_{final}, w_1, \ldots, w_d)$, a system of inequations that must be satisfiable when s_{final} is coverable. This system is defined over a vector $\mathbf{x}$ of free variables ranging over $\mathbb{Z}^\Delta$. Intuitively, $\mathbf{x}(\delta)$ denotes the number of times an execution uses the rule δ to cover the state s_{final}.

We first build a system of equations over $\mathbf{x}$, called the *location constraints* that takes into account the number of times an execution enters and leaves a location. To do so, we introduce the vector $\mathbf{e}_i$ in $\mathbb{Z}^Q$ defined as zero everywhere except for the index i where it is one. We denote by $LC_{s_{final}}(\mathbf{x})$ the following system of equations:

$$\mathbf{e}_{q_{init}} + \sum_{\delta=(p,op,q)\in\Delta} \mathbf{x}(\delta)(-\mathbf{e}_p + \mathbf{e}_q) = \mathbf{e}_{q_{final}} \tag{1}$$

Next, we build a system of inequalities over $\mathbf{x}$, called the *channel constraints* that counts the number of times each message m is sent and received in each channel i. To do so, we introduce the vector $\mathbf{e}_{i,m}$ defined as the matrix in $\mathbb{Z}^{d\times|M|}$ where the value is zero everywhere except for the index i, m where it is one. Given a word w of messages, we denote by $|w|_m$ the number of occurrences of m in w. We denote by $CC_{s_{final}}(\mathbf{x})$ the following system of inequations where the inequality $\geq$ over the matrices is defined component-wise:

$$\sum_{\delta=(p,i!m,q)\in\Delta} \mathbf{x}(\delta)\mathbf{e}_{i,m} - \sum_{\delta=(p,i?m,q)\in\Delta} \mathbf{x}(\delta)\mathbf{e}_{i,m} \geq \sum_{\substack{m\in M \\ i\in\{1,\ldots,d\}}} |w_i|_m \tag{2}$$

Finally, we introduce the system of equations $SI_{s_{final}}(\mathbf{x})$ defined as the conjunction $LC_{s_{final}}(\mathbf{x}) \wedge CC_{s_{final}}(\mathbf{x})$. The proof of the following lemma is immediate by observing that if s_{final} is coverable there exists an execution from s_{init} to a state $s \geq s_{final}$. Denoting by $\mathbf{x}(\delta)$ the number of times the rule δ is used by this execution, we get a vector $\mathbf{x}$ satisfying $SI_{s_{final}}(\mathbf{x})$.

Lemma 8.1. *The system $SI_{s_{final}}(\mathbf{x})$ is satisfiable for every coverable configuration s_{final}.*

Corollary 8.2. *The set $I_{LCS} = \{s_{final} \mid \exists \mathbf{x}\ SI_{s_{final}}(\mathbf{x})\}$ is an invariant.*

9 EXPERIMENTAL EVALUATION

We have presented a coverability checking algorithm, namely BCwP, that is parametrized by a downward-closed invariant. The algorithm uses this invariant in order to accelerate the classical backward coverability analysis. We have also presented three different invariant generation techniques, respectively based on compact simple regular expressions (CSREs), on message ordering flows (MOFs) and on the state inequation (SI). We now present our experimental results.

We want to evaluate two main effects of the invariants: the number of states handled and the time saved or added. On one hand, if the algorithm handles less states (computes less states with *cpre*, prunes more), this leads to a gain in time. On the other hand, an invariant can also slow down the computation in two ways: firstly for the generation of the invariant itself and secondly for checking membership of a given state in the invariant.

We have implemented the BCwP algorithm in the language *OCaml*. The prototype consists of a little more than three thousand lines of code. It can be found online [18]. It only works for lossy channel systems but, as it is modular, it can be extended to all well-structured transition systems with decidable order and effective pre-basis. New invariants can also be added easily. The state inequation invariant needs the help of an SMT-solver. We chose *Z3* [14]. Our experiments were run on a machine running Ubuntu Linux 14.04, with an Intel

Table 1: Time in seconds to solve coverability instances.

	∅	*SI*	*CSRE*	*MOF*	Best	*McScM*
Peterson3	947.6	1.9	21.5	22.1	1.9	9.3
Peterson4	>96h	2934.4	>96h	>96h	2934.4	error
pop3	157.9	0.3	>96h	0.9	0.3	0.6
BAwPC	0.4	4.3	0.0	0.0	0.0	0.1
BAwCC	1.9	11.1	0.1	0.1	0.1	0.1
BAwCC_enh	2.3	13.7	0.1	0.1	0.1	1.6
BAwPC_enh	0.7	3.5	0.0	0.0	0.0	0.2
BRP	0.2	0.1	0.3	0.3	0.1	1.2
server	0.1	1.3	0.6	0.4	0.1	5.0
server_err	0.0	0.0	0.0	0.0	0.0	0.0
ring2	0.2	0.2	0.4	0.4	0.2	2.6
tcp	0.0	0.0	0.0	0.0	0.0	0.1
tcp_err	0.0	0.1	0.0	0.0	0.0	0.0

i7-4770 CPU at 3.40 GHz and 16 GB of memory. The timeout for each run was set to 96 hours.

Our prototype takes as input a system of communicating finite-state automata, each with their own locations and rules, given in the *scm* file format. We build (on-the-fly) a global LCS by taking the cartesian product of these automata. While the *CSRE* and *MOF* invariants are computed on this global LCS, we optimize the state inequation invariant to avoid an exponential blow-up in Equation (1). Instead of applying Equation (1) to the global LCS, we create one equation for each automaton. The two systems are equivalent but the membership problem in the latter is easier.

The different examples come with the tool *McScM* [19]. We present the results of 13 examples. The examples *BAwCC*, *BAwPC*, *BAwCC_enh* and *BAwPC_enh* are business protocols and came from [23]. The first two are coverable examples and the last two are their enhanced uncoverable versions. Others examples include two versions of the Peterson protocol, one with three peers, the other with four, a version of the pop3 protocol, a simple server protocol and a tcp protocol with and without an error inserted, and finally a ring protocol. Many examples have tens of target states, and our prototype can handle that. Even though we presented our algorithm to solve the coverability problem for only one target state, it is easy to extend the algorithm for a set of targets, by taking $U_0 = \uparrow(S_{final} \cap I)$ where S_{final} is the set of all target states.

We were unable to compare our prototype with *TReX* [5] because the tool is not available anymore. So we compared with *McScM* [19], which uses the same *scm* format and can solve the coverability problem for LCS, even though this is not its primary goal. *McScM* has four *verification engines*. The best engine suited to our examples was *cegar*. It was able to solve 10 of 13 examples within three minutes but crashed with *Peterson4* and was not finished after two hours for *brp* and *BAwCC_enh*. In comparison, our prototype solved all cases with the invariant state inequation. Without invariant and with the invariant *MOF* it was able to solved all examples except two and with the invariant *CSRE* all except three.

Table 1 compares the time needed to solve the coverability problem without invariant, noted ∅, and with the invariants *SI*, *CSRE* and *MOF*. The least time is reported in the fifth column. In the last column, we put the time needed by *McScM* with the best engine

Table 2: Number of elements visited.

	∅	*SI*	*CSRE*	*MOF*
Peterson3	359131	2127	67	67
Peterson4	>5113486	852630	-	-
pop3	241595	813	-	17842
BAwPC	15266	15266	1587	1587
BAwCC	31693	31693	2480	2480
BAwCC_enh	38218	38218	59	59
BAwPC_enh	19193	12824	44	44
BRP	9343	470	9132	9343
server	8873	8873	7720	8873
server_err	105	1	1	1
ring2	11141	772	11141	11141
tcp	1519	37	1437	1519
tcp_err	1423	218	1290	1364

Table 3: Number of states tested and ratio of pruned states.

	SI		*CSRE*		*MOF*	
	#	%	#	%	#	%
Peterson3	1970	72.2	67	100.0	67	100
Peterson4	745009	72.0	-	-	-	-
pop3	324	61.8	-	-	5433	35.0
BAwPC	3645	0.0	1069	81.4	1069	81.4
BAwCC	7186	0.0	1718	80.6	1718	80.6
BAwCC_enh	7671	0.0	59	100.0	59	100.0
BAwPC_enh	2834	0.4	44	100.0	44	100.0
BRP	172	27.3	2692	3.8	2650	0.0
server	2037	0.0	1910	5.3	2037	0.0
server_err	1	100.0	1	100.0	1	100.0
ring2	490	71.2	2393	0.1	2393	0.0
tcp	37	81.1	751	3.1	768	0.0
tcp_err	173	47.4	671	8.0	686	3.8

for each example. We can see that *SI* dramatically accelerates the computation in the biggest examples. The same effect is also found for the invariant *MOF* except for the *Peterson4* protocol where *MOF* did not terminate within 96 hours.

Those numbers can be explained by the numbers of operations that the algorithm performed. Table 2 compares the numbers of nodes visited, i.e., the numbers of targets states added each step with all states created by the *cpre* in line 6 before removing those that are already in $\uparrow B$. Sometimes the number of visited states is very low compared to the version without invariant. For example for the invariants *CSRE* and *MOF* for the examples *BAwCC_enh* and *BAwPC_enh* it is 59 and 44 compared to 38218 and 19193. It is because the invariants were able to prune all targets states. Therefore the algorithm did not even enter the while loop.

To understand more specifically the pruning for each invariant and each example, we provide Table 3. It shows the number of times the algorithm tested if a state was or was not in the invariant, as well as the percentage of times the state was outside the invariant and hence was pruned.

We see that many times the algorithm was able to prune some states and therefore to decrease the number of states handled. This comes at a cost, since we need to compute a least fixpoint for the

Table 4: Cost in seconds to use invariants.

	SI		*CSRE*		*MOF*	
	pre	$\in$	pre	$\in$	pre	$\in$
Peterson3	0.0	1.9	21.5	0.0	22.1	0.0
Peterson4	0.0	1895.2	>96h	-	>96h	-
pop3	0.0	0.3	>96h	-	0.1	0.0
BAwPC	0.0	3.8	0.0	0.0	0.0	0.0
BAwCC	0.0	9.3	0.1	0.0	0.1	0.0
BAwCC_enh	0.0	11.4	0.1	0.0	0.1	0.0
BAwPC_enh	0.0	3.1	0.0	0.0	0.0	0.0
BRP	0.0	0.1	0.1	0.0	0.1	0.0
server	0.0	1.2	0.6	0.0	0.3	0.0
server_err	0.0	0.0	0.0	0.0	0.0	0.0
ring2	0.0	0.2	0.2	0.0	0.2	0.0
tcp	0.0	0.0	0.0	0.0	0.0	0.0
tcp_err	0.0	0.1	0.0	0.0	0.0	0.0

invariants *CSRE* and *MOF* and to check if state is in the invariant or not. Table 4 shows for each invariant the time spent before starting the algorithm and the total time spent checking if a state is in the invariant. We see that the state inequation is very fast to pre-compute. The reason is that it just needs to write the inequation for the SMT-solver *Z3*. Satisfiability checks are only accounted for in the times to check membership in the invariant. The invariant *CSRE* can take too much pre-computation time as illustrated with Example 6.3 in Section 6. The *MOF* invariant was able to do the computation in less time excepted for the Peterson protocols with four peers. Note that, unlike *Z3*, the least fixpoint module was not carefully coded with performance in mind.

The *MOF* and *CSRE* invariants have a similar pruning impact (see Table 3), but we see that *MOF* was better overall. It pruned less in a few examples but the biggest gap was between 5.3% against 0% for *server*. And in 6 examples out of 11 where they both finished they pruned exactly the same states. Because the pre-computation times are better for *MOF*, the invariant *MOF* is better than *CSRE*.

Regarding the *SI* invariant, we observe in Tables 3 and 4 that it did not prune states for the models *BAwCC* and *BAwPC* but it still cost time. Recall that the state inequation keeps track of the numbers of messages in each channel. This means that it was not helpful to have this information for these models.

Remark 9.1. We did not present results using combinations of invariants, but this is of course possible. Indeed, the intersection of downward-closed invariants is also a downward-closed invariant. In practice, we do not compute the intersection: if we have two invariants I_1 and I_2, we prune a configuration c if, and only, if $c \notin I_1$ or $c \notin I_2$. Our prototype is capable of using any combination of the invariants *SI*, *CSRE* and *MOF*. It turns out that such combinations are worse than single invariants for our examples. For instance, our hardest examples (*Peterson3* and *Peterson4*) do not benefit from *SI+MOF*, because the time needed for the pre-processing of *MOF* is greater than the time to solve the coverability question with *SI*. This does not mean that combinations of invariants have no interest, they could be useful for other models. □

Overall we see that, for most examples, at least one invariant was able to accelerate the computation.

10 CONCLUSION

We have presented in this paper a backward coverability algorithm for WSTSs, parametrized by downward-closed forward invariants. We have introduced three new invariants for lossy channel systems. One of these invariants counts messages and the other two keep track of the order of messages. We have implemented a prototype to assess the efficiency of these invariants. Our experimental evaluation shows an acceleration of the classical approach for two of the three invariants. As future work, we intend to apply these techniques to verify safety properties on weak-memory models because they are closely related to LCSs [6].

REFERENCES

[1] P-A. Abdulla, M-F. Atig, A. Bouajjani, and T-P. Ngo. 2016. The Benefits of Duality in Verifying Concurrent Programs under TSO. In *CONCUR (LIPIcs)*, Vol. 59. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 5:1–5:15.

[2] P-A. Abdulla, A. Bouajjani, and B. Jonsson. 1998. On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels. In *CAV (LNCS)*, Vol. 1427. Springer, 305–318.

[3] P-A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. 2000. Algorithmic Analysis of Programs with Well Quasi-ordered Domains. *Information and Computation* 160, 1-2 (2000), 109–127.

[4] P-A. Abdulla and B. Jonsson. 1996. Verifying Programs with Unreliable Channels. *Inf. Comput.* 127, 2 (1996), 91–101.

[5] A. Annichini, A. Bouajjani, and M. Sighireanu. 2001. TReX: A Tool for Reachability Analysis of Complex Systems. In *CAV (LNCS)*, Vol. 2102. Springer, 368–372.

[6] M-F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. 2010. On the verification problem for weak memory models. In *POPL*. ACM, 7–18.

[7] M. Blondin, A. Finkel, Ch. Haase, and S. Haddad. 2016. Approaching the Coverability Problem Continuously. In *TACAS (LNCS)*, Vol. 9636. Springer, 480–496.

[8] B. Boigelot and P. Godefroid. 1999. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. *Formal Methods in System Design* 14, 3 (1999), 237–255.

[9] A. Bouajjani, P. Habermehl, and T. Vojnar. 2004. Abstract Regular Model Checking. In *CAV (LNCS)*, Vol. 3114. Springer, 372–386.

[10] G. Cécé, A. Finkel, and S.P. Iyer. 1996. Unreliable Channels are Easier to Verify Than Perfect Channels. *Inf. Comput.* 124, 1 (1996), 20–31.

[11] P. Chambart and P. Schnoebelen. 2008. The Ordinal Recursive Complexity of Lossy Channel Systems. In *LICS*. IEEE Computer Society, 205–216.

[12] E-M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50, 5 (2003), 752–794.

[13] P. Cousot and R. Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*. ACM, 238–252.

[14] L. de Moura and N. Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (LNCS)*, Vol. 4963. Springer, 337–340.

[15] J. Esparza, R. Ledesma-Garza, R. Majumdar, P.J. Meyer, and F. Niksic. 2014. An SMT-Based Approach to Coverability Analysis. In *CAV (LNCS)*, Vol. 8559. Springer, 603–619.

[16] A. Finkel and P. Schnoebelen. 2001. Well-structured transition systems everywhere! *Theor. Comput. Sci.* 256, 1-2 (2001), 63–92.

[17] T. Geffroy, J. Leroux, and G. Sutre. 2016. Occam's Razor Applied to the Petri Net Coverability Problem. In *RP (LNCS)*, Vol. 9899. Springer, 77–89.

[18] T. Geffroy, J. Leroux, and G. Sutre. 2017. coverability checker for LCS. http://dept-info.labri.u-bordeaux.fr/~tgeffroy/lcs/. (2017).

[19] A. Heußner, T. Le Gall, and G. Sutre. 2012. McScM: A General Framework for the Verification of Communicating Machines. In *TACAS (LNCS)*, Vol. 7214. Springer, 478–484.

[20] G. Higman. 1952. Ordering by Divisibility in Abstract Algebras. *Proc. London Math. Soc.* s3-2, 1 (1952), 326–336.

[21] A. Kaiser, D. Kroening, and T. Wahl. 2014. A Widening Approach to Multithreaded Program Verification. *ACM* 36, 4 (2014), 14:1–14:29.

[22] R. Mayr. 2003. Undecidable problems in unreliable computations. *Theor. Comput. Sci.* 297, 1-3 (2003), 337–354.

[23] A.P. Ravn, J. Srba, and S. Vighio. 2011. Modelling and Verification of Web Services Business Activity Protocol. In *TACAS (LNCS)*, Vol. 6605. Springer, 357–371.