ON REACHABILITY FOR UNIDIRECTIONAL CHANNEL SYSTEMS EXTENDED WITH REGULAR TESTS

P. JANČAR, P. KARANDIKAR, AND PH. SCHNOEBELEN

FEI, Techn. Univ. Ostrava

Chennai Mathematical Institute and LSV, ENS Cachan

LSV, ENS Cachan, CNRS

ABSTRACT. "Unidirectional channel systems" (Chambart & Schnoebelen, CONCUR 2008) are finite-state systems where one-way communication from a Sender to a Receiver goes via one reliable and one unreliable unbounded fifo channel. While reachability is decidable for these systems, equipping them with the possibility of testing regular properties on the contents of channels makes it undecidable. Decidability is preserved when only emptiness and nonemptiness tests are considered: the proof relies on an elaborate reduction to a generalized version of Post's Embedding Problem.

1. INTRODUCTION

Channel systems are a family of computational models where concurrent agents communicate via (usually unbounded) fifo communication channels [13]. They are sometimes called *queue automata* when there is only one finite-state agent using the channels as fifo memory buffers. These models are well-suited to the formal specification and algorithmic analysis of communication protocols and concurrent programs [9, 10, 35].

A particularly interesting class of channel systems are the *lossy channel systems*, "LCSes" for short, popularized by Abdulla, Bouajjani, Jonsson, Finkel, *et al.* [14, 4, 2]. Lossy channels are unreliable and can lose messages nondeterministically and without any notification. This weaker model is easier to analyse: safety, inevitability and several more properties are decidable for LCSes [14, 4, 1, 6] while they are undecidable when channels are reliable.

Let us stress that LCSes also are an important and fundamental computation model *per se*. During the last decade, they have been used as an automaton model to prove the decidability (or the hardness) of problems on Timed Automata, Metric Temporal Logic, modal logics, etc. [3, 36, 31, 30, 33, 11, 34, 7]. They also are a very natural low-level computational model that captures some important complexity classes in the ordinal-recursive hierarchy [18, 39, 28, 40, 38].

Unidirectional channel systems, "UCSes" for short, are channel systems where a Sender process communicates to a Receiver process via *one reliable* and *one lossy* channel, see Fig. 1. They were

1st author: Supported by the project GAČR:P202/11/0340.
2nd author: Partially funded by Tata Consultancy Services.
3rd author: Supported by Grant ANR-11-BS02-001.

LOGICAL METHODS
IN COMPUTER SCIENCE
DOI:10.2168/LMCS-???

introduced by Chambart and Schnoebelen who identified them as a minimal setting to which one can reduce reachability problems for more complex combinations of lossy and reliable channels [16]. UCSes are limited to one-way communication: there are no channels going from Receiver to Sender.

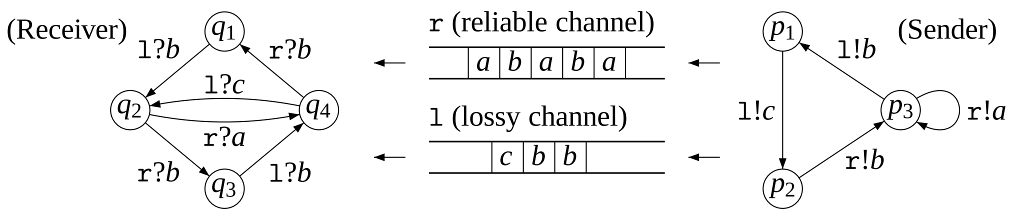

Figure 1: UCS = buffered one-way communication via one reliable and one lossy channels

One-way communication appears, e.g., in half-duplex protocols [26] or in the acyclic networks of [32, 5].

The reachability problem for UCSes is quite challenging: it was proved decidable by reformulating it more abstractly as the *(Regular) Post Embedding Problem* (PEP), which is easier to analyze [15, 17, 19]. We want to stress that, while PEP is a natural variant of Post's Correspondence Problem, it was first identified through questions on UCSes. Recently, PEP has proved useful in other areas: graph logics for databases [7] and fast-growing complexity [28].

Testing channel contents. In basic channel systems, the agents are not allowed to inspect the contents of the channels. However, it is sometimes useful to enrich the basic setup with tests. For example, a multiplexer process will check each of its input channels in turn and will rely on emptiness and/or non-emptiness tests to ensure that this round robin policy does not block when one input channel is empty [37]. In other settings, channel systems with insertion errors becomes more expressive when emptiness tests are allowed [11].

In this article we consider such emptiness and non-emptiness tests, as well as more general tests given by arbitrary regular predicates on channel contents. A simple example is given below in Fig. 2 (see page 5) where some of Sender's actions depend on the parity of the number of messages currently in r. When verifying plain UCSes, one can reorder steps and assume a two-phase behaviour where all Sender steps occur before all Receiver steps. When one has tests, one can no longer assume this.

Our contribution. We extend UCSes with the possibility of testing channel contents with regular predicates (Section 2). This makes reachability undecidable even with restricted sets of simple tests (Section 3). Our main result (Theorem 4.1) is that reachability is decidable for UCSes extended with emptiness and non-emptiness tests. The proof goes through a series of reductions, some of them nontrivial, that leave us with UCSes extended by only emptiness tests on a single side of a single channel, called "$Z_1^{\mathtt{l}}$ tests" (sections 5 and 6). This minimal extension is then reduced (Section 7) to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, or "PEP with partial codirectness", a nontrivial extension of PEP that was recently proved decidable [29]. This last reduction extends the reduction from UCS to PEP in [17]. Finally, Section 8 proves that emptiness and/or non-emptiness tests strictly enrich the basic UCS model.

Related work. Emptiness and non-emptiness tests have been considered already in [37], while Promela (SPIN's input language) offers head tests (that test the first available message without consuming it) [25]. Beyond such *specific* tests, we are not aware of results that consider models with a general notion of tests on channel contents (except in the case of LCSes where very general tests can be allowed without compromising the main decidability results, see [8, sect. 6]).

Regarding unidirectional channels, the decidability results in [5, 32, 24, 23, 20] apply to systems where communication between two agents is limited to a *single* one-way channel (sometimes complemented with a finite shared memory, real-time clock, integer-valued counter, or local push-down stack). Finally let us mention the recent work by Clemente *et al*. where fifo and "bag" channels can be mixed: one can see bag channels as unreliable channels where the temporal ordering of messages is not preserved [21].

2. Unidirectional Channel Systems

2.1. **Unidirectional Channel System with Tests.** A *UCST* is a tuple $S = (\mathtt{Ch}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, where $\mathtt{M}$ is the finite alphabet of *messages*, Q_1, Q_2 are the disjoint finite sets of *states* of Sender and Receiver, respectively, and Δ_1, Δ_2 are the finite sets of *rules* of Sender and Receiver, respectively. $\mathtt{Ch} = \{\mathtt{r}, \mathtt{l}\}$ is a fixed set of channel names, just *channels* for short, where $\mathtt{r}$ is *reliable* and $\mathtt{l}$ is *lossy* (since messages in $\mathtt{l}$ can spontaneously disappear).

A rule $\delta \in \Delta_i$ is a tuple $(q, \mathtt{c}, \alpha, q') \in Q_i \times \mathtt{Ch} \times Act \times Q_i$ where the set of actions Act contains *tests*, checking whether the contents of $\mathtt{c} \in \mathtt{Ch}$ belongs to some regular language $R \in \mathsf{Reg}(\mathtt{M})$, and *communications* (sending a message $a \in \mathtt{M}$ to $\mathtt{c}$ in the case of Sender's actions, reading it for Receiver's). Allowed actions also include the *empty action* (no test, no communication) that will be treated as "sending/reading the empty word ε"; formally we put $Act \stackrel{\text{def}}{=} \mathsf{Reg}(\mathtt{M}) \cup \mathtt{M} \cup \{\varepsilon\}$.

We also write a rule $(q, \mathtt{c}, \alpha, q')$ as $q \xrightarrow{\mathtt{c},\alpha} q'$, or specifically $q \xrightarrow{\mathtt{c}:R} q'$ for a rule where the action is a test on $\mathtt{c}$, and $q \xrightarrow{\mathtt{c}!a} q'$ or $q \xrightarrow{\mathtt{c}?a} q'$ when the action is a communication by Sender or by Receiver, respectively. We also write just $q \to q'$ or $q \xrightarrow{\top} q'$ when the action is empty.

In graphical representations like Fig. 1, Sender and Receiver are depicted as two disjoint directed graphs, where states appear as nodes and where rules $q \xrightarrow{\mathtt{c},\alpha} q'$ appear as edges from q to q' with the corresponding labellings.

2.2. **Operational Semantics.** The behaviour of a UCST is defined via an operational semantics along standard lines. A *configuration* of $S = (\mathtt{Ch}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$ is a tuple $C \in Conf_S \stackrel{\text{def}}{=} Q_1 \times Q_2 \times \mathtt{M}^* \times \mathtt{M}^*$. In $C = (q_1, q_2, u, v)$, q_1 and q_2 are the current states of Sender and Receiver, respectively, while u and v are the current contents of $\mathtt{r}$ and $\mathtt{l}$, respectively.

The rules in $\Delta_1 \cup \Delta_2$ give rise to transitions in the expected way. We use two notions of transitions, or "steps", between configurations. We start with so-called "reliable" steps: given two configurations $C = (q_1, q_2, u, v)$, $C' = (q'_1, q'_2, u', v')$ and a rule $\delta = (q, \mathtt{c}, \alpha, q')$, there is a reliable step denoted $C \xrightarrow{\delta} C'$ if, and only if, the following four conditions are satisfied:

- **states:** $q = q_1$ and $q' = q'_1$ and $q_2 = q'_2$ (for Sender rules), or $q = q_2$ and $q' = q'_2$ and $q_1 = q'_1$ (for Receiver rules);
- **tests:** if δ is a test rule $q \xrightarrow{\mathtt{c}:R} q'$, then $\mathtt{c} = \mathtt{r}$ and $u \in R$, or $\mathtt{c} = \mathtt{l}$ and $v \in R$, and furthermore $u' = u$ and $v' = v$;

writes: if δ is a writing rule $q \xrightarrow{\mathtt{c}!x} q'$ with $x \in \mathtt{M} \cup \{\varepsilon\}$, then $\mathtt{c} = \mathtt{r}$ and $u' = ux$ and $v' = v$, or $\mathtt{c} = \mathtt{l}$ and $u' = u$ and $v' = vx$;

reads: if δ is a reading rule $q \xrightarrow{\mathtt{c}?x} q'$, then $\mathtt{c} = \mathtt{r}$ and $u = xu'$ and $v' = v$, or $\mathtt{c} = \mathtt{l}$ and $u' = u$ and $v = xv'$.

This reliable behaviour is completed with message losses. For $v, v' \in \mathtt{M}^*$, we write $v' \sqsubseteq_1 v$ when v' is obtained by deleting a single (occurrence of a) symbol from v, and we let $\sqsubseteq$ denote the reflexive-transitive closure of $\sqsubseteq_1$. Thus $v' \sqsubseteq v$ when v' is a scattered subword, i.e., a subsequence, of v. (E.g., *aba* $\sqsubseteq_1$ *abba* and *aa* $\sqsubseteq$ *abba*.) This is extended to configurations and we write $C' \sqsubseteq_1 C$ or $C' \sqsubseteq C$ when $C' = (q_1, q_2, u, v')$ and $C = (q_1, q_2, u, v)$ with $v' \sqsubseteq_1 v$ or $v' \sqsubseteq v$, respectively. Now, whenever $C' \sqsubseteq_1 C$, the operational semantics of S includes a step from C to C', called a *message loss* step, and denoted $C \xrightarrow{\text{los}} C'$, considering that "los" is an extra, implicit rule that is always allowed.

Thus a step $C \xrightarrow{\delta} C'$ of S is either a reliable step, when $\delta \in \Delta_1 \cup \Delta_2$, or a (single) message loss, when $\delta = \text{los}$.

Remark 2.1 (On reliable steps)**.** As is usual with unreliable channel systems, the reliable semantics plays a key role even though the object of our study is reachability via not necessarily reliable steps. First it is a normative yardstick from which one defines the unreliable semantics by extension. Then many hardness results on lossy systems are proved via reductions where a lossy system simulates in some way the reliable (and Turing-powerful) behaviour: proving the correctness of such reductions requires having the concept of reliable steps. □

Remark 2.2 (UCSTs and well-structured systems)**.** It is well-known that $(\mathtt{M}^*, \sqsubseteq)$ is a well-quasi-order (a wqo): any infinite sequence $v_0, v_1, v_2, \ldots$ of words over $\mathtt{M}$ contains an infinite increasing subsequence $v_{i_0} \sqsubseteq v_{i_1} \sqsubseteq v_{i_2} \sqsubseteq \cdots$ This classic result, called Higman's Lemma, plays a fundamental role in the algorithmic verification of lossy channel systems and other well-structured systems [14, 22]. Here we note that $(Conf, \sqsubseteq)$ is *not* a wqo since $C \sqsubseteq D$ requires equality on channel $\mathtt{r}$, so that UCSTs are not well-structured systems despite the presence of a lossy channel. □

2.3. **Reachability.** A *run* from C_0 to C_n is a sequence of chained steps $C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \cdots \xrightarrow{\delta_n} C_n$, abbreviated as $C_0 \xrightarrow{*} C_n$ (or $C_0 \xrightarrow{+} C_n$ when we rule out zero-length runs).

The *(Generalized) Reachability Problem*, or just "G-G-Reach" for short, is the question, given a UCST $S = (\mathtt{Ch}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, some states $p_{\text{in}}, p_{\text{fi}} \in Q_1$, $q_{\text{in}}, q_{\text{fi}} \in Q_2$, some regular languages $U, V, U', V' \in \mathsf{Reg}(\mathtt{M})$, whether there are some $u \in U$, $v \in V$, $u' \in U'$ and $v' \in V'$ such that S has a run $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, u, v) \xrightarrow{*} C_{\text{fi}} = (p_{\text{fi}}, q_{\text{fi}}, u', v')$.

Since U, V, U', V' can be taken as singleton sets, the G-G-Reach problem is more general than asking whether S has a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ for some given initial and final configurations. We shall need the added generality in Section 6 in particular. However, sometimes we will also need to put restrictions on U, V, U', V'. We use E-G-Reach to denote the reachability problem where $U = V = \{\varepsilon\}$, i.e., where C_{in} has empty channels (E is for "Empty"), while $U', V' \in \mathsf{Reg}(\mathtt{M})$ are not constrained. We will also consider the E-E-Reach restriction where $U = V = U' = V' = \{\varepsilon\}$. It is known —see [16, Theo 3.1]— that E-E-Reach is decidable for UCSes, i.e., UCSTs that do not use tests.

3. Testing channels and the undecidability of reachability

Despite their similarities, UCSes and LCSes (lossy channel systems) behave differently. The algorithms deciding reachability for LCSes can easily accommodate regular (or even more expressive) tests [8, Sect. 6]. By contrast, UCSes become Turing-powerful when equipped with regular tests. The main result of this section is the undecidability of reachability for UCSTs. To state the respective theorem in a stronger version, we first introduce a notation for restricting the (regular) tests.

3.1. **Restricted sets of tests.**

When $\mathcal{T} \subseteq \mathsf{Reg}(\mathtt{M})$, we write UCST[$\mathcal{T}$] to denote the class of UCSTs where only tests, i.e. languages, belonging to $\mathcal{T}$ are allowed. Thus UCSTs and UCSes coincide with UCST[$\mathsf{Reg}(\mathtt{M})$] and UCST[$\varnothing$], respectively. We single out some simple tests (i.e., languages) defined via regular expressions:

$$Even \stackrel{\text{def}}{=} (\mathtt{M}.\mathtt{M})^*, \qquad Odd \stackrel{\text{def}}{=} \mathtt{M}.Even, \qquad Z \stackrel{\text{def}}{=} \varepsilon, \qquad N \stackrel{\text{def}}{=} \mathtt{M}^+, \qquad H_a \stackrel{\text{def}}{=} a.\mathtt{M}^*.$$

Thus $\mathcal{P} = \{Even, Odd\}$ is the set of *parity* tests, Z is the *emptiness* (or “zero”) test, N is the *non-emptiness* test and $\mathcal{H} = \{H_a \mid a \in \mathtt{M}\}$ is the set of *head* tests (that allows checking what is the first message in a channel *without consuming it*). Note that the non-emptiness test can be simulated with head tests.

Before proving (in later sections) the decidability of G-G-Reach for UCST[$\{Z, N\}$], we start by showing that E-E-Reach is undecidable for both UCST[$\mathcal{P}$] and UCST[$\mathcal{H}$]: this demonstrates that we get undecidability not only with simple “global” tests (parity tests) whose outcome depends on the entire contents of a channel, but also with simple “local” tests (head tests).

In fact, we even show the stronger statement that E-E-Reach is undecidable for UCST[$\mathcal{P}_1^{\mathtt{r}}$] and UCST[$\mathcal{H}_1^{\mathtt{r}}$], where the use of subscripts and/or superscripts means that we consider restricted systems where only Sender (for subscript 1, only Receiver for subscript 2) may use the tests, and that the tests may only apply on channel $\mathtt{r}$ or $\mathtt{l}$ (depending on the superscript). E.g., in UCST[$\mathcal{P}_1^{\mathtt{r}}$] the only allowed tests are parity tests performed by Sender on channel $\mathtt{r}$.

Theorem 3.1. *Reachability (E-E-Reach) is undecidable for both UCST[$\mathcal{P}_1^{\mathtt{r}}$] and UCST[$\mathcal{H}_1^{\mathtt{r}}$].*

We now proceed to prove Theorem 3.1 by simulating queue automata with UCSTs.

3.2. **Simulating queue automata.**

Like queue automata, UCSes have a reliable channel but, unlike them, Sender (or Receiver) cannot both read *and* write from/to it. If Sender could somehow read from the head of $\mathtt{r}$, it would be as powerful as a queue automaton, i.e., Turing-powerful. Now we show that parity tests used by Sender on $\mathtt{r}$ allow us to construct a simple protocol making Receiver act as a proxy for Sender and implement read actions on its behalf. See Fig. 2 for an illustrating example of how Sender simulates a rule $p_1 \xrightarrow{\mathtt{r}?a} p_2$.

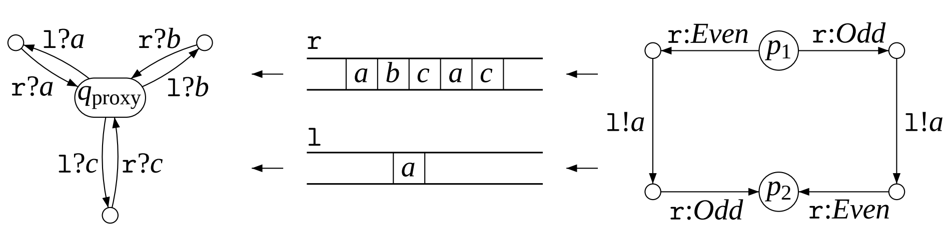

Figure 2: Sender simulates “$p_1 \xrightarrow{\mathtt{r}?a} p_2$” with parity tests and proxy Receiver

Described informally, the protocol is the following:

(1) Channel `l` is initially empty.
(2) In order to "read" from `r`, Sender checks and records whether the length of the current contents of `r` is odd or even, using a parity test on `r`.
(3) It then writes on `l` the message that it wants to read (a in the example).
(4) During this time Receiver waits in its initial q_{proxy} state and tries to read from `l`. When it reads a message a from `l`, it understands it as a request telling it to read a from `r` on behalf of Sender. Once it has performed this read on `r` (when a really was there), it returns to q_{proxy} and waits for the next instruction.
(5) Meanwhile, Sender checks that (equivalently, waits until) the parity of the contents of `r` has changed, and on detecting this change, concludes that the read was successful.
(6) Channel `l` is now empty and the simulation of a read by Sender is concluded.

If no messages are lost on `l`, the protocol allows Sender to read on `r`; if a message is lost on `l`, the protocol deadlocks. Also, Sender deadlocks if it attempts to read a message that is not at the head of `r`, in particular when `r` is empty; i.e., Sender has to guess correctly.

Our simulation of a queue automaton thus introduces many possible deadlocks, but it still suffices for proving undecidability of reachability, namely of E-E-Reach for UCST[$\mathcal{P}_1^{\mathtt{r}}$].

To prove undecidability for UCST[$\mathcal{H}_1^{\mathtt{r}}$] we just modify the previous protocol. We use two copies of the message alphabet, e.g., using two "colours". When writing on `r`, Sender strictly alternates between the two colours. If now Sender wants to read a given letter, say a, from `r`, it checks that an a (of the right colour) is present at the head of `r` by using $\mathcal{H}_1^{\mathtt{r}}$ tests. It then asks Receiver to read a by sending a message via `l`. Since colours alternate in `r`, Sender can check (i.e., wait until), again using head tests, that the reading of a occurred.

4. Main theorem and a roadmap for its proof

We will omit set-brackets in the expressions like UCST[$\{Z,N\}$], UCST[$\{Z_1,N_1\}$], UCST[$\{Z_1^{\mathtt{1}}\}$]; we thus write UCST[Z,N], UCST[Z_1,N_1], UCST[$Z_1^{\mathtt{1}}$], etc. We now state our main theorem:

Theorem 4.1. *Reachability (G-G-Reach) is decidable for UCST[Z,N].*

Hence adding emptiness and nonemptiness tests to UCSes does not compromise the decidability of reachability (unlike what happens with parity or head tests).

Our proof of Theorem 4.1 is quite long, being composed of several consecutive reductions, some of which are nontrivial. A scheme of the proof is depicted in Fig. 3, and we give a brief outline in the rest of this section.

We first recall that the reachability problem for UCSes (i.e., for UCST[$\varnothing$]) was shown decidable via a reduction to PEP (Post's Embedding Problem) in [17]. Relying on this earlier result (by reducing UCST[Z,N] to UCST[$\varnothing$]) or extending its proof (by reducing UCST[Z,N] to PEP directly) does not seem at all trivial. At some point $\mathsf{PEP}_{\text{codir}}^{\text{partial}}$, a non-trivial generalization of the basic PEP problem, was introduced as a certain intermediate step and shown decidable in [29].

Once it is known that $\mathsf{PEP}_{\text{codir}}^{\text{partial}}$ is decidable, our proof for Theorem 4.1 is composed of two main parts:

(1) One part, given in Section 7, is a reduction of E-E-Reach for UCST[$Z_1^{\mathtt{1}}$] to $\mathsf{PEP}_{\text{codir}}^{\text{partial}}$. It is relatively compact, since we have found a suitable intermediate notion between runs of UCST[$Z_1^{\mathtt{1}}$] and solutions of $\mathsf{PEP}_{\text{codir}}^{\text{partial}}$.
(2) The other part of the proof, given in sections 5 and 6, reduces G-G-Reach for UCST[Z,N] to E-E-Reach for UCST[$Z_1^{\mathtt{1}}$]. It has turned out necessary to decompose this reduction in a

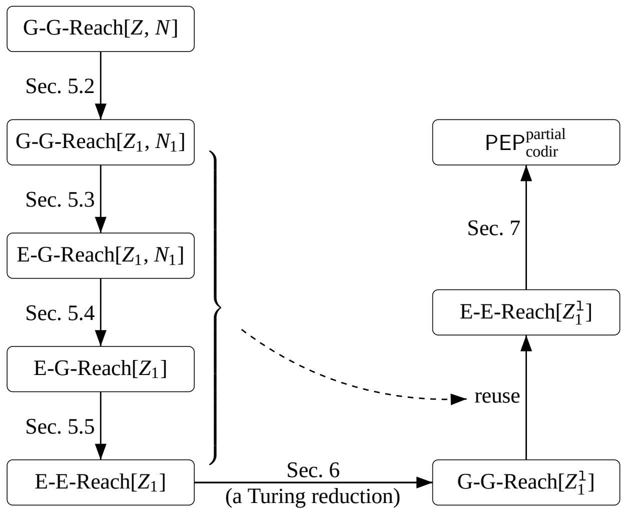

Figure 3: Roadmap of the reductions from G-G-Reach[Z, N] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$

series of smaller steps (as depicted in Fig. 3) where features such as certain kinds of tests, or general initial and final conditions, are eliminated step by step. The particular way in which these features are eliminated is important. For example, we eliminate Z_2 and N_2 tests by one simulation reducing G-G-Reach[Z, N] to G-G-Reach[Z_1, N_1] (Sec. 5.2); the simulation would not work if we wanted to eliminate Z_2 and N_2 separately, one after the other.

One of the crucial steps in our series is the reduction from E-E-Reach[Z_1] to G-G-Reach[Z_1^1]. This is a Turing reduction, while we otherwise use many-one reductions. Even though we start with a problem instance where the initial and final configurations have empty channel contents, we need oracle calls to a problem where the initial and final conditions are more general. This alone naturally leads to considering the G-G-Reach instances.

We note that, when UCSes are equipped with tests, reducing from G-G-Reach to E-E-Reach is a problem in itself, for which the simple "solution" that we sketched in our earlier extended abstract [27] does not work.

It seems also worth noting that all reductions in Section 5 treat the two channels in the same way; no special arrangements are needed to handle the lossiness of `l`. The proofs of correctness, of course, do need to take the lossiness into account.

5. Reducing G-G-Reach for UCST[Z,N] to E-E-Reach for UCST[Z_1]

This section describes four simulations that, put together, entail Point 1 in Theorem 5.1 below. Moreover, the last three simulations also yield Point 2. We note that the simulations are tailored to the reachability problem: they may not preserve other behavioural aspects like, e.g., termination or deadlock-freedom.

Theorem 5.1.
(1) G-G-Reach[Z,N] many-one reduces to E-E-Reach[Z_1].
(2) G-G-Reach[Z_1^1] many-one reduces to E-E-Reach[Z_1^1].

Before proceeding with the four reductions, we present a simple Commutation Lemma that lets us reorder runs and assume that they follow a specific pattern.

5.1. **Commuting steps in UCST[Z,N] systems.** We say that two consecutive steps $C \xrightarrow{\delta_1} C' \xrightarrow{\delta_2} C''$ (of some S) *commute* if $C \xrightarrow{\delta_2} D \xrightarrow{\delta_1} C''$ for some configuration D of S. The next lemma lists some conditions that are sufficient for commuting steps in an arbitrary UCST[Z,N] system S:

Lemma 5.2 (Commutation)**.** *Two consecutive steps $C \xrightarrow{\delta_1} C' \xrightarrow{\delta_2} C''$ commute in any of the following cases:*

(1) *No contact: δ_1 is a read/write/test by Sender or Receiver acting on one channel* $\mathtt{c}$ *(or a message loss on $\mathtt{c} = \mathtt{l}$), while δ_2 is a rule of the* other agent *acting on the* other channel *(or is a loss).*
(2) *Postponable loss: δ_1 is a message loss that does not occur at the head of (the current content of)* $\mathtt{l}$.
(3) *Advanceable Sender: δ_1 is a Receiver's rule or a loss, and δ_2 is a Sender's rule but not a Z_1-test.*
(4) *Advanceable loss: δ_2 is a loss and δ_1 is not an "$\mathtt{l}$:N" test or a Sender's write on* $\mathtt{l}$.

Proof. By a simple case analysis. For example, for (2) we observe that if δ_1 loses a symbol behind the head of $\mathtt{l}$, then there is another message at the head of $\mathtt{l}$, and thus commuting is possible even if δ_2 is an "$\mathtt{l}?a$" read or an "$\mathtt{l}$:Z" test. □

We will use Lemma 5.2 several times and in different ways. For the time being, we consider in particular the convenient restriction to "head-lossy" runs. Formally, a message loss $C \xrightarrow{\text{los}} C'$ is *head-lossy* if it is of the form $(p,q,u,av) \xrightarrow{\text{los}} (p,q,u,v)$ where $a \in \mathtt{M}$ (i.e., the lost message was the head of $\mathtt{l}$). A *run* $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ is *head-lossy* if all its message loss steps are head-lossy, or occur after all the reliable steps in the run (it is convenient to allow unconstrained losses at the end of the run). Repeated use of Point (2) in Lemma 5.2 easily yields the next corollary:

Corollary 5.3. *If there is a run from C_{in} to C_{fi} then there is a head-lossy run from C_{in} to C_{fi}.*

5.2. **Reducing G-G-Reach[Z,N] to G-G-Reach[Z_1,N_1].** Our first reduction eliminates Z and N tests by Receiver. These tests are replaced by reading two special new messages, "$\mathtt{z}$" and "$\mathtt{n}$", that Sender previously put in the channels.

Formally, we consider an instance of G-G-Reach[Z,N], made of a given UCST $S = (\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$, given states $p_{\text{in}}, p_{\text{fi}} \in Q_1$, $q_{\text{in}}, q_{\text{fi}} \in Q_2$, and given languages $U,V,U',V' \in \text{Reg}(\mathtt{M})$. We construct a new UCST S' from S as follows (see Fig. 4):

(1) We add two special new messages $\mathtt{z},\mathtt{n}$ to $\mathtt{M}$, thus creating the alphabet $\mathtt{M}' \stackrel{\text{def}}{=} \mathtt{M} \uplus \{\mathtt{z},\mathtt{n}\}$.
(2) For each channel $\mathtt{c} \in \{\mathtt{r},\mathtt{l}\}$ and each Sender's state $p \in Q_1$ we add new states $p^1_{\mathtt{c}}$, $p^2_{\mathtt{c}}$ and an "*(emptiness) testing loop*" $p \xrightarrow{\mathtt{c}:Z} p^1_{\mathtt{c}} \xrightarrow{\mathtt{c}!\mathtt{z}} p^2_{\mathtt{c}} \xrightarrow{\mathtt{c}:Z} p$ (i.e., three new rules).
(3) For every Sender's writing rule θ of the form $p \xrightarrow{\mathtt{c}!x} p'$ we add a new state p_θ and the following three rules: $p \xrightarrow{\top} p_\theta$, $p_\theta \xrightarrow{\mathtt{c}!\mathtt{n}} p_\theta$ (a "*padding loop*"), and $p_\theta \xrightarrow{\mathtt{c}!x} p'$.
(4) For every Receiver's rule $q \xrightarrow{\mathtt{c}:Z} q'$ (testing emptiness of $\mathtt{c}$) we add the rule $q \xrightarrow{\mathtt{c}?\mathtt{z}} q'$.
(5) For every Receiver's rule $q \xrightarrow{\mathtt{c}:N} q''$ (testing non-emptiness of $\mathtt{c}$) we add the rule $q \xrightarrow{\mathtt{c}?\mathtt{n}} q''$.
(6) At this stage, the resulting system is called S_{aux}.

(7) Finally we remove all Receiver's tests, i.e., the rules $q \xrightarrow{\mathtt{c}:Z} q'$ and $q \xrightarrow{\mathtt{c}:N} q''$. We now have S'.

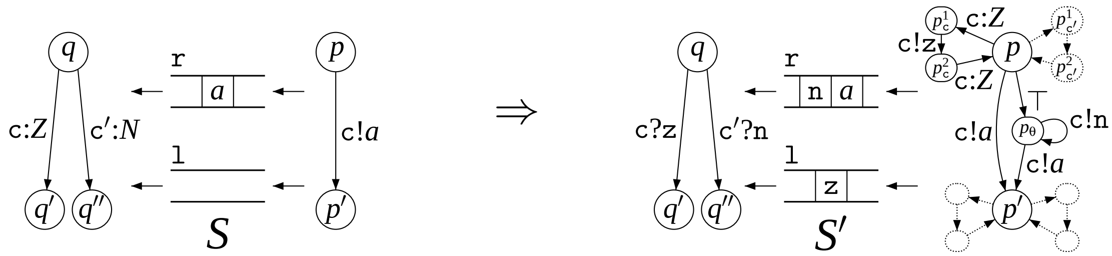

Figure 4: Reducing G-G-Reach$[Z,N]$ to G-G-Reach$[Z_1,N_1]$: eliminating Receiver's tests

The intuition behind S' is that Sender runs a small protocol signaling to Receiver what the status of the channels is. When a channel is empty, Sender may write a z to it that Receiver can read in place of testing for emptiness. For correctness, it is important that Sender does not proceed any further until this z has disappeared from the channel. For non-emptiness tests, Sender can always write several extraneous n messages before writing an original message. Receiver can then read these n's in place of testing for nonemptiness.

For $w = a_1 a_2 \dots a_\ell \in \mathtt{M}^*$, we let $pad(w) \stackrel{\text{def}}{=} \mathtt{n}^* a_1 \mathtt{n}^* a_2 \dots \mathtt{n}^* a_\ell$ denote the set (a regular language) of all *paddings* of w, i.e., words obtained by inserting any number of n's in front of the original messages. Note that $pad(\varepsilon) = \{\varepsilon\}$. This is extended to arbitrary languages in the usual way: for $L \subseteq \mathtt{M}^*$, $pad(L) = \bigcup_{w \in L} pad(w)$ and we note that, when L is regular, $pad(L)$ is regular too. Furthermore, one easily derives an FSA (a finite-state automaton) or a regular expression for $pad(L)$ from an FSA or a regular expression for L.

By replacing S, U, V with $S', pad(U), pad(V)$ (and keeping p_{in}, p_{fi}, q_{in}, q_{fi}, U', V' unchanged), the initial G-G-Reach$[Z,N]$ instance is transformed into a G-G-Reach$[Z_1,N_1]$ instance. The correctness of this reduction is captured by the next lemma, that we immediately proceed to prove in the rest of section 5.2:

Lemma 5.4. *For any $u, v, u', v' \in \mathtt{M}^*$, S has a run $(p_{\text{in}}, q_{\text{in}}, u, v) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ if, and only if, S' has a run $(p_{\text{in}}, q_{\text{in}}, \hat{u}, \hat{v}) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ for some padded words $\hat{u} \in pad(u)$ and $\hat{v} \in pad(v)$.*

Though we are ultimately interested in S and S', it is convenient to consider special runs of S_{aux} since S_{aux} "contains" both S and S'. We rely on Corollary 5.3 and tacitly assume that all runs are head-lossy. We say that a (head-lossy) *run* $C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} \cdots \xrightarrow{\delta_n} C_n$ of S_{aux} is *faithful* if $C_0 = (p_0, q_0, u_0, v_0)$ with $u_0, v_0 \in pad(\mathtt{M}^*)$, $C_n = (p_n, q_n, u_n, v_n)$ with $u_n, v_n \in \mathtt{M}^*$, $p_0, p_n \in Q_1$, $q_0, q_n \in Q_2$, and the following two properties are satisfied (for all $i = 1, 2, \dots, n$):

– if δ_i is some $p \xrightarrow{\mathtt{c}:Z} p^1_{\mathtt{c}}$ then δ_{i+1}, δ_{i+2}, and δ_{i+3} are $p^1_{\mathtt{c}} \xrightarrow{\mathtt{c!z}} p^2_{\mathtt{c}}$, $q \xrightarrow{\mathtt{c?z}} q'$, $p^2_{\mathtt{c}} \xrightarrow{\mathtt{c}:Z} p$ (for some $q, q' \in Q_2$). In this case, the subrun $C_{i-1} \xrightarrow{*} C_{i+3}$ is called a *P1-segment* of the run. (P1)

– if δ_i is some $p \xrightarrow{\top} p_\theta$ then there is some $j > i$ such that $\delta_{i+1}, \delta_{i+2}, \dots, \delta_j$ are $p_\theta \xrightarrow{\mathtt{c!n}} p_\theta \xrightarrow{\mathtt{c!n}} \cdots \xrightarrow{\mathtt{c!n}} p_\theta \xrightarrow{\mathtt{c!}a} p'$ for some $a \in \mathtt{M}$ and $p' \in Q_1$. The subrun $C_{i-1} \xrightarrow{*} C_j$ is called a *P2-segment*. (P2)

Informally, a run is faithful if it uses the new rules (introduced in S_{aux}) in the "intended" way: e.g., P1 enforces that each z written by Sender (necessarily via a rule $p^{\mathtt{c}}_1 \xrightarrow{\mathtt{c!z}} p^{\mathtt{c}}_2$) is immediately read after being written in the empty channel. We note that any run of S is trivially faithful since it does not use the new rules.

We now exhibit two reversible transformations of runs of S_{aux}, one for Z tests in §5.2.1, the other for N tests in §5.2.2, that preserve faithfulness. This will allow us to translate runs of S, witnessing the original instance, to faithful runs of S', witnessing the created instance, and vice versa. Finally we show in §5.2.3 that if there is a run of S' witnessing the created instance, then there is a faithful one as well.

When describing the two transformations we shall assume, in order to fix notations, that we transform a test on channel $\mathtt{l}$; the case for the channel $\mathtt{r}$ is completely analogous. For both transformations we assume a faithful (head-lossy) run π of S_{aux} in the following form:

$$(p_{\text{in}}, q_{\text{in}}, u_0, v_0) = C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \cdots \xrightarrow{\delta_n} C_n = (p_{\text{fi}}, q_{\text{fi}}, u_n, v_n) \tag{π}$$

where $\delta_1, \ldots, \delta_n$ can be rules of S_{aux} or the "los" symbol for steps where a message is lost. For $i = 0, 1, \ldots, n$, we let $C_i = (p_i, q_i, u_i, v_i)$.

5.2.1. *Trading Z_2 tests for P1-segments.* Assume that the step $C_m \xrightarrow{\delta_{m+1}} C_{m+1}$ in π is a Z_2-test (an emptiness test by Receiver), hence has the form $(p, q, w, \varepsilon) \xrightarrow{\mathtt{l}:Z} (p, q', w, \varepsilon)$ if we assume $\mathtt{c} = \mathtt{l}$. We may replace this step with the following steps

$$(p, q, w, \varepsilon) \xrightarrow{\mathtt{l}:Z} (p_1^1, q, w, \varepsilon) \xrightarrow{\mathtt{l}!\mathtt{z}} (p_1^2, q, w, \mathtt{z}) \xrightarrow{\mathtt{l}?\mathtt{z}} (p_1^2, q', w, \varepsilon) \xrightarrow{\mathtt{l}:Z} (p, q', w, \varepsilon) \tag{5.1}$$

using the rules introduced in S_{aux}. This transforms (the faithful run) π into another faithful run π', decreasing the number of Receiver's tests (by one occurrence of a Z_2-test). In the other direction, if π contains a P1-segment $C_{m-1} \xrightarrow{*} C_{m+3}$, it must be of the form (5.1), when the involved channel is $\mathtt{c} = \mathtt{l}$, and we can replace it with one step $C_{m-1} \xrightarrow{\mathtt{c}:Z} C_{m+3}$, preserving faithfulness.

5.2.2. *Trading N_2 tests for occurrences of* $\mathtt{n}$. Now assume that the step $C_m \xrightarrow{\delta_{m+1}} C_{m+1}$ is an $N_2^{\mathtt{l}}$-test, hence has the form $(p, q, u, xv) \xrightarrow{\mathtt{l}:N} (p, q', u, xv)$ for some message $x \in \mathtt{M}'$. Now $x \neq \mathtt{z}$ since there was no $\mathtt{z}$'s in v_0 and, as noted above, any $\mathtt{z}$ written by Sender in a faithful run is immediately read. Hence $x \in \mathtt{M} \cup \{\mathtt{n}\}$. We want to replace the $q \xrightarrow{\mathtt{l}:N} q'$ test (by Receiver) with a $q \xrightarrow{\mathtt{l}?\mathtt{n}} q'$ but this requires inserting one $\mathtt{n}$ in $\mathtt{l}$, i.e., using a new rule $p_\theta \xrightarrow{\mathtt{l}!\mathtt{n}} p_\theta$ at the right moment.

We now follow the (occurrence of) x singled out in C_m and find the first configuration, say C_k, where this x appears already; we can thus write $v_i = w_i x w'_i$, i.e., $C_i = (p_i, q_i, u_i, w_i x w'_i)$, for $i = k, k+1, \ldots, m$. Here x always depicts the same occurrence, and e.g., $w_m x w'_m = xv$ entails $w_m = \varepsilon$ and $w'_m = v$. By adding $\mathtt{n}$ in front of x in each C_i for $i = k, k+1, \ldots, m$, we obtain new configurations $C'_k, C'_{k+1}, \ldots, C'_m$ given by $C'_i = (p_i, q_i, u_i, w_i\, \mathtt{n}\, x w'_i)$. Now $C'_k \xrightarrow{\delta_{k+1}} C'_{k+1} \xrightarrow{\delta_{k+2}} \cdots \xrightarrow{\delta_m} C'_m$ is a valid run of S_{aux} since x is not read during $C_k \xrightarrow{*} C_m$ and since, thanks to the presence of x, adding one $\mathtt{n}$ does not change the (non)emptiness status of $\mathtt{l}$ in this subrun. Moreover, since $q \xrightarrow{\mathtt{l}:N} q'$ is a rule of S, there is a rule $q \xrightarrow{\mathtt{l}?\mathtt{n}} q'$ in S_{aux}, where $C'_m = (p, q, u, \mathtt{n}\, xv) \xrightarrow{\mathtt{l}?\mathtt{n}} (p, q', u, xv) = C_{m+1}$ is a valid step.

If $k = 0$ (i.e., if x is present at the beginning of π), we have exhibited a faithful run $C'_0 \xrightarrow{*} C'_m \xrightarrow{\mathtt{l}?\mathtt{n}} C_{m+1} \xrightarrow{*} C_n$, starting from $C'_0 = (p_{\text{in}}, q_{\text{in}}, u_0, w_0\, \mathtt{n}\, x w'_0)$, where $w_0 \mathtt{n} x w'_0 \in pad(v_0)$ since $v_0 = w_0\, x\, w'_0$. If $k > 0$, the highlighted occurrence of x necessarily appears in C_k via $\delta_k = p_{k-1} \xrightarrow{\mathtt{l}!x} p_k$ and we have $v_k = v_{k-1} x$. If δ_k is a rule of S, we may exhibit a sequence $C_{k-1} \xrightarrow{*} C'_k$ using the new rules

$$C_{k-1} \xrightarrow{\top} (p_{\delta_k}, q_{k-1}, u_{k-1}, v_{k-1}) \xrightarrow{\mathtt{l}!\mathtt{n}} (p_{\delta_k}, q_{k-1}, u_{k-1}, v_{k-1}\, \mathtt{n}) \xrightarrow{\mathtt{l}!x} (p_k, q_{k-1}, u_{k-1}, v_{k-1}\, \mathtt{n}\, x) = C'_k\,,$$

while if δ_k is a new rule $p_\theta \xrightarrow{\texttt{1}!x} p_k$, we can use $C_{k-1} \xrightarrow{\texttt{1}!\texttt{n}}\xrightarrow{\texttt{1}!x} C'_k$. In both cases we can use $C_{k-1} \xrightarrow{*} C'_k$ to construct a new faithful run $C_0 \xrightarrow{*} C_{k-1} \xrightarrow{*} C'_k \xrightarrow{*} C'_m \to C_{m+1} \xrightarrow{*} C_n$. We have again decreased the number of Receiver's tests, now by one occurrence of an N_2-test.

For the backward transformation we assume that $\texttt{n}$ occurs in a configuration of π. We select one such occurrence and let $C_k, C_{k+1}, \ldots, C_m$ ($0 \leq k \leq m < n$) be the part of π where this occurrence of $\texttt{n}$ appears. For $i = k, k+1, \ldots, m$, we highlight this occurrence of $\texttt{n}$ by writing v_i in the form $w_i \,\texttt{n}\, w'_i$ (assuming w.l.o.g. that the $\texttt{n}$ occurs in $\texttt{1}$), i.e., we write $C_i = (p_i, q_i, u_i, w_i \,\texttt{n}\, w'_i)$. Removing the $\texttt{n}$ yields new configurations $C'_k, C'_{k+1}, \ldots, C'_m$ given by $C'_i = (p_i, q_i, u_i, w_i\, w'_i)$.

We claim that $C'_k \xrightarrow{\delta_{k+1}} C'_{k+1} \cdots \xrightarrow{\delta_m} C'_m$ is a valid run of S_{aux}. For this, we only need to check that removing $\texttt{n}$ does not make channel $\texttt{1}$ empty in some C'_i where δ_{i+1} is an $N^{\texttt{1}}$-test. If $k = 0$ then $\texttt{n}$ in $v_0 = w_0 \texttt{n} w'_0$ is followed by a letter $x \in \texttt{M} \cup \{\texttt{n}\}$ since $v_0 \in \mathit{pad}(\texttt{M}^*)$. This x remains in $\texttt{1}$ until at least C_{m+1} since it cannot be read while $\texttt{n}$ remains, nor can it be lost before the $C_i \to C_{i+1}$ step since the run is head-lossy. If $k > 0$, then our $\texttt{n}$ appeared in a step of the form $C_{k-1} = (p_\theta, q_{k-1}, u_{k-1}, v_{k-1}) \xrightarrow{\texttt{1}!\texttt{n}} C_k = (p_\theta, q_{k-1}, u_{k-1}, v_{k-1}\texttt{n})$ (for some write rule θ of S, inducing $p_\theta \xrightarrow{\texttt{1}!\texttt{n}} p_\theta$ in S_{aux}). Since $p_0 = p_{\text{in}}$ is not p_θ, a rule $p_\ell \xrightarrow{\top} p_\theta$ was used before step k, and π has a P2-segment $C_\ell \xrightarrow{\top} \cdots C_{k-1} \xrightarrow{\texttt{1}!\texttt{n}} C_k \xrightarrow{\texttt{1}!x} \cdots C_{\ell'}$ where $\ell' \leq m$ and $x \in \texttt{M} \cup \{\texttt{n}\}$ is present in all $C_{k+1}, \ldots, C_m$. As before, this x guarantees that $C_{k-1} = C'_k \xrightarrow{\delta_{k+1}} C'_{k+1} \cdots \xrightarrow{\delta_m} C'_m$ is a valid run of S_{aux}.

We now recall that $m < n$ and that δ_{m+1} is either $q_m \xrightarrow{\texttt{1}?\texttt{n}} q_{m+1}$ or the loss of $\texttt{n}$. In the first case, S_{aux} has a step $C'_m \xrightarrow{\texttt{1}:N} C_{m+1}$, while in the second case $C'_m = C_{m+1}$.

The corresponding run $C'_0 \xrightarrow{*} C'_m \xrightarrow{*} C_{m+1} \xrightarrow{*} C_n$ in the case $k = 0$, or $C_0 \xrightarrow{*} C_{k-1} \to C'_{k+1} \xrightarrow{*} C'_m \xrightarrow{*} C_{m+1} \xrightarrow{*} C_n$ in the case $k > 0$, is a faithful run; we have thus removed an occurrence of $\texttt{n}$, possibly at a cost of introducing one N_2 test.

5.2.3. *Handling S' runs and faithfulness.* Since a witness run of S is (trivially) faithful, the above transformations allow us to remove one by one all occurrences of Receiver's Z and N tests, creating a (faithful) witness run for S' (with a possibly padded C_0). We have thus proved the "only-if" part of Lemma 5.4. The "if" part is shown analogously, now using the two transformations in the other direction and removing occurrences of the new $\texttt{z}$ and $\texttt{n}$ messages, *with one proviso*: we only transform faithful runs. We thus need to show that if S' has a (head-lossy) run $(p_{\text{in}}, q_{\text{in}}, \hat{u}, \hat{v}) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ then it also has a faithful one.

Let us assume that π above, of the form $C_0 \xrightarrow{*} C_n$, is a witness run of S', not necessarily faithful, having minimal length. We show how to modify it locally so that the resulting run is faithful.

Assume that some rule $\delta_i = p \xrightarrow{\top} p_\theta$ is used in π, and that P2 fails on this occurrence of δ_i. Since π does not end in state p_θ, Sender necessarily continues with some (possibly zero) $p_\theta \xrightarrow{\texttt{c}!\texttt{n}} p_\theta$ steps, followed by some $\delta_j = p_\theta \xrightarrow{\texttt{c}!x} p'$. Now all Receiver or message loss steps between δ_i and δ_j can be swapped and postponed after δ_j since Receiver has no tests and Sender does not test between δ_i and δ_j (recall Lemma 5.2(3)). After the transformation, δ_i and the rules after it form a P2-segment. Also, since message losses have been postponed, the run remains head-lossy.

Consider now a rule δ_i of the form $p \xrightarrow{\texttt{c}:Z} p^1_{\texttt{c}}$ in π and assume that P1 fails on this occurrence. Sender necessarily continues with some $\delta_j = p^1_{\texttt{c}} \xrightarrow{\texttt{c}!\texttt{z}} p^2_{\texttt{c}}$ and $\delta_k = p^2_{\texttt{c}} \xrightarrow{\texttt{c}:Z} p$, interleaved with Receiver's steps and/or losses. It is clear that the $\texttt{z}$ written on $\texttt{c}$ by δ_j must be lost, or read by a Receiver's $\delta_\ell = q \xrightarrow{\texttt{c}?\texttt{z}} q'$ before δ_k can be used. The read or loss occurs at some step ℓ with $j < \ell < k$.

Note that Receiver does not read from $\mathtt{c}$ between steps i and k, except perhaps at step ℓ. Since Sender only tests for emptiness of $\mathtt{c}$ between steps i and k, all Receiver's steps and losses between steps i and ℓ can be swapped and put before δ_i. The run remains head-lossy since the swapped losses do not occur on $\mathtt{c}$, which is empty at step i. Similarly, all non-Sender steps between steps ℓ and k can be swapped after δ_k, preserving head-lossiness. The obtained run has a segment of the form $C \xrightarrow{\mathtt{c}:Z} \xrightarrow{\mathtt{c!z}} \xrightarrow{\mathtt{c?z}} \xrightarrow{\mathtt{c}:Z} C'$ that is now a P1-segment, or of the form $C \xrightarrow{\mathtt{c}:Z} \xrightarrow{\mathtt{c!z}} \xrightarrow{\mathrm{los}} \xrightarrow{\mathtt{c}:Z} C' = C$, i.e., a dummy loop $C \xrightarrow{+} C$ that contradicts minimality of π.

5.3. **Reducing G-G-Reach[Z_1, N_1] to E-G-Reach[Z_1, N_1].** A G-G-Reach[Z_1, N_1] instance where the initial contents of $\mathtt{r}$ and $\mathtt{l}$ are restricted to (regular languages) U and V respectively can be transformed into an equivalent instance where U and V are both replaced with $\{\varepsilon\}$. For this, one adds a new (fresh) initial state p_{new} to Sender, from which Sender first nondeterministically generates some word $u \in U$, writing it on $\mathtt{r}$, then generates some word $v \in V$, writing it on $\mathtt{l}$, and then enters p_{in}, the original initial state. The resulting S' is just S with extra states and rules between p_{new} and p_{in} that mimic FSAs for U and V.

Stating the correctness of this reduction has the form

$$S \text{ has a run } (p_{\mathrm{in}}, q_{\mathrm{in}}, u, v) \xrightarrow{*} C \text{ for some } u \in U \text{ and } v \in V \text{ iff } S' \text{ has a run } (p_{\mathrm{new}}, q_{\mathrm{in}}, \varepsilon, \varepsilon) \xrightarrow{*} C\,. \quad (\star)$$

Now, since S' can do $(p_{\mathrm{new}}, q_{\mathrm{in}}, \varepsilon, \varepsilon) \xrightarrow{*} (p_{\mathrm{in}}, q_{\mathrm{in}}, u, v)$ for any $u \in U$ and $v \in V$, the left-to-right implication in $(\star)$ is clear. Note that, in the right-to-left direction, *it is essential that Receiver has no tests* and this is what we missed in [27]. Indeed, it is the absence of Receiver tests that allows us to reorder any S' run from $(p_{\mathrm{new}}, q, \varepsilon, \varepsilon)$ so that all steps that use the new "generating" rules (from p_{new} to p_{in}) happen before any Receiver steps.

5.4. **Reducing E-G-Reach[Z_1, N_1] to E-G-Reach[Z_1].** When there are no Receiver tests and a run starts with the empty channels, then N_1 tests can be easily eliminated by a buffering technique on Sender's side. Each channel $\mathtt{c} \in \{\mathtt{r}, \mathtt{l}\}$ gets its one-letter buffer $\mathrm{B}_{\mathtt{c}}$, which can be emptied at any time by moving its content to $\mathtt{c}$. Sender can only write to an empty buffer; it passes a $Z_1^{\mathtt{c}}$ test if both channel $\mathtt{c}$ and $\mathrm{B}_{\mathtt{c}}$ are empty, while any $N_1^{\mathtt{c}}$ test is replaced with the (weaker) "test" if $\mathrm{B}_{\mathtt{c}}$ is nonempty.

Formally, we start with an instance $(S, p_{\mathrm{in}}, p_{\mathrm{fi}}, q_{\mathrm{in}}, q_{\mathrm{fi}}, \{\varepsilon\}, \{\varepsilon\}, U', V')$ of E-G-Reach[Z_1, N_1], where $S = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, and we create $S' = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1', \Delta_1', Q_2, \Delta_2)$ arising from S as follows (see Fig. 5). We put $Q_1' = Q_1 \times (\mathtt{M} \cup \{\varepsilon\}) \times (\mathtt{M} \cup \{\varepsilon\})$; the components x, y in a state $\langle q, x, y \rangle$

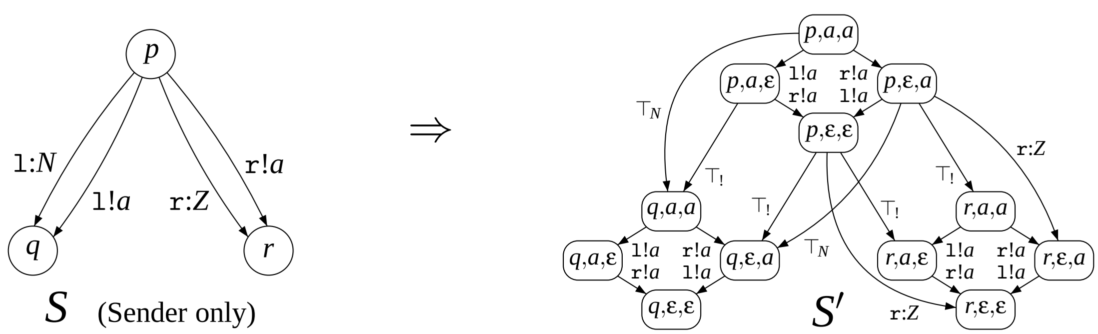

Figure 5: Reducing E-G-Reach[Z_1, N_1] to E-G-Reach[Z_1]

denote the contents of the buffers for $\mathtt{r}$ and $\mathtt{l}$, respectively. We now replace each rule $q \xrightarrow{\mathtt{r}!x} q'$ with

$\langle q,\varepsilon,y\rangle \xrightarrow{\top} \langle q',x,y\rangle$ for all $y \in \mathtt{M}\cup\{\varepsilon\}$ (Fig. 5 uses "$\top_!$" to highlight these transformed rules). Each $q \xrightarrow{\mathtt{r}:N} q'$ is replaced with $\langle q,x,y\rangle \xrightarrow{\top} \langle q',x,y\rangle$ for all x,y where $x \neq \varepsilon$ (Fig. 5 uses "$\top_N$"). Each $q \xrightarrow{\mathtt{r}:Z} q'$ is replaced with $\langle q,\varepsilon,y\rangle \xrightarrow{\mathtt{r}:Z} \langle q',\varepsilon,y\rangle$ (for all y). Analogously we replace all $q \xrightarrow{\mathtt{l}!x} q'$, $q \xrightarrow{\mathtt{l}:N} q'$, and $q \xrightarrow{\mathtt{l}:Z} q'$. Moreover, we add the rules $\langle q,x,y\rangle \xrightarrow{\mathtt{r}!x} \langle q,\varepsilon,y\rangle$ (for $x \neq \varepsilon$) and $\langle q,x,y\rangle \xrightarrow{\mathtt{l}!y} \langle q,x,\varepsilon\rangle$ (for $y \neq \varepsilon$). Our desired reduction is completed, by the next lemma:

Lemma 5.5. *S has a run $C_{\text{in}} = (p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (p_{\text{fi}},q_{\text{fi}},u',v') = C_{\text{fi}}$ if, and only if, S' has a run $C'_{\text{in}} = (\langle p_{\text{in}},\varepsilon,\varepsilon\rangle,\langle q_{\text{in}},\varepsilon,\varepsilon\rangle,\varepsilon,\varepsilon) \xrightarrow{*} (\langle p_{\text{fi}},\varepsilon,\varepsilon\rangle,\langle q_{\text{fi}},\varepsilon,\varepsilon\rangle,u',v') = C'_{\text{fi}}$.*

Proof. $\Leftarrow$: A run $C'_{\text{in}} = C'_0 \xrightarrow{\delta'_1} C'_1 \xrightarrow{\delta'_2} C'_2 \cdots \xrightarrow{\delta'_n} C'_n = C'_{\text{fi}}$ of S' can be simply translated to a run of S by the following transformation: each $C'_i = (\langle p_i,x,y\rangle,q_i,u_i,v_i)$ is translated to $C_i = (p_i,q_i,u_ix,v_iy)$, each step $C'_{i-1} \xrightarrow{\delta'_i} C'_i$ where δ'_i is $\langle q,\varepsilon,y\rangle \xrightarrow{\top} \langle q',x,y\rangle$ is replaced with $C_{i-1} \xrightarrow{\delta} C_i$ where δ is $q \xrightarrow{\mathtt{r}!x} q'$, etc. It can be easily checked that the arising run $C_0 \xrightarrow{*} C_n$ is indeed a valid run of S (that can be shorter because it "erases" the steps by the rules $\langle q,x,y\rangle \xrightarrow{\mathtt{r}!x} \langle q,\varepsilon,y\rangle$ and $\langle q,x,y\rangle \xrightarrow{\mathtt{l}!y} \langle q,x,\varepsilon\rangle$).

$\Rightarrow$: A run $C_{\text{in}} = C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \cdots \xrightarrow{\delta_n} C_n = C_{\text{fi}}$ of S can be translated into a run of S' by a suitable transformation, starting with $C'_0 = (\langle p_{\text{in}},\varepsilon,\varepsilon\rangle,\langle q_{\text{in}},\varepsilon,\varepsilon\rangle,\varepsilon,\varepsilon)$. Suppose that $C_0 \xrightarrow{*} C_i = (p,q,ux,vy)$ has been translated to $C'_0 \xrightarrow{*} C'_i = (\langle p,x,y\rangle,q,u,v)$ (for some $x,y \in \mathtt{M}\cup\{\varepsilon\}$). If δ_{i+1} is $p \xrightarrow{\mathtt{r}!a} p'$, then we translate $C_i \xrightarrow{\delta_i} C_{i+1}$ in the case $x=\varepsilon$ to $C'_i \to C'_{i+1} = (\langle p',a,y\rangle,q,u,v)$ (using the rule $\langle p,\varepsilon,y\rangle \xrightarrow{\top} \langle p',a,y\rangle$), and in the case $x \neq \varepsilon$ to $C'_i \to (\langle p,\varepsilon,y\rangle,q,ux,v) \to (\langle p',a,y\rangle,q,ux,v) = C'_{i+1}$ (using the rules $\langle p,x,y\rangle \xrightarrow{\mathtt{r}!x} \langle p,\varepsilon,y\rangle$ and $\langle p,\varepsilon,y\rangle \xrightarrow{\top} \langle p',a,y\rangle$). We handle the other forms of δ_{i+1} in the obvious way; e.g., if δ_{i+1} is a loss at (the head of) $\mathtt{l}$ while $C'_i = (\langle p,x,y\rangle,q,u,\varepsilon)$, then we also use two steps: $C'_i \to (\langle p,x,\varepsilon\rangle,q,u,y) \xrightarrow{\text{los}} (\langle p,x,\varepsilon\rangle,q,u,\varepsilon) = C'_{i+1}$. This process obviously results in a valid run of S'. ☐

5.5. **Reducing E-G-Reach[Z_1] to E-E-Reach[Z_1].** The idea of the reduction is similar to what was done in section 5.3. The regular final conditions "$u' \in U'$" and "$v' \in V'$" are checked by Receiver consuming the final channel contents. When Sender (guesses that it) is about to write the first message that will be part of the final u' in $\mathtt{r}$ (respectively, the final v' in $\mathtt{l}$), it signals this by inserting a special symbol # just before. After it has written # to a channel, Sender is not allowed to test that channel anymore.

Formally we start with an instance $(S,p_{\text{in}},p_{\text{fi}},q_{\text{in}},q_{\text{fi}},\{\varepsilon\},\{\varepsilon\},U',V')$ of E-G-Reach[Z_1], where $S = (\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$. With S we associate S' where $\mathtt{M}' = \mathtt{M} \uplus \{\#\}$, as sketched in Fig. 6. This yields the instance $(S',p'_{\text{in}},p'_{\text{fi}},q_{\text{in}},q_f,\{\varepsilon\},\{\varepsilon\},\{\varepsilon\},\{\varepsilon\})$ of E-E-Reach[Z_1], for the new final Receiver state q_f.

We define $S' = (\{\mathtt{r},\mathtt{l}\},\mathtt{M}',Q'_1,\Delta'_1,Q'_2,\Delta'_2)$ with the Receiver part Q'_2,Δ'_2 obtained from Q_2,Δ_2 by adding q_f and other necessary states and so called *cleaning rules* so that q_f is reachable from q_{fi} precisely by sequences of read-steps $\mathtt{r}?\#$, $\mathtt{l}?\#$, $\mathtt{r}?a_1$, $\mathtt{r}?a_2$, $\ldots$, $\mathtt{r}?a_{m_1}$, $\mathtt{l}?b_1$, $\mathtt{l}?b_2$, $\ldots$, $\mathtt{l}?b_{m_2}$, where $u' = a_1a_2\ldots a_{m_1} \in U'$ and $v' = b_1b_2\ldots b_{m_2} \in V'$. (The new states and cleaning rules mimic finite-state automata accepting $\{\#\}\cdot U'$ and $\{\#\}\cdot V'$.)

The Sender part Q'_1, Δ'_1 of S' is obtained from Q_1,Δ_1 as follows. We put $Q'_1 \stackrel{\text{def}}{=} Q_1 \times \{\top,\#\} \times \{\top,\#\}$, and $p'_{\text{in}} = \langle p_{\text{in}},\top,\top\rangle$, $p'_{\text{fi}} = \langle p_{\text{fi}},\#,\#\rangle$. A state $\langle p,x,y\rangle$ "remembers" if # has been already written to $\mathtt{r}$ ($x=\#$) or not ($x=\top$); similarly for $\mathtt{l}$ (by $y=\#$ or $y=\top$). For changing the status (just

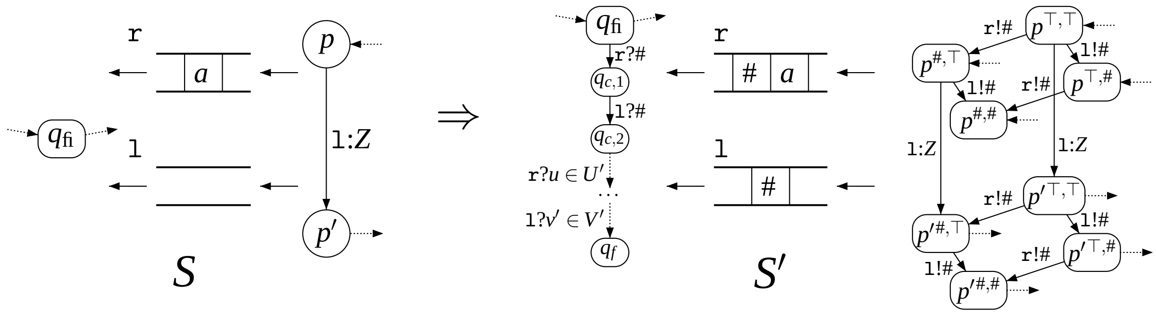

Figure 6: Reducing E-G-Reach[Z_1] to E-E-Reach[Z_1]

once for each channel), Δ_1' contains the rules $\langle p,\top,y\rangle \xrightarrow{\texttt{r}!\#} \langle p,\#,y\rangle$ and $\langle p,x,\top\rangle \xrightarrow{\texttt{l}!\#} \langle p,x,\#\rangle$ for each $p \in Q_1$ and $x,y \in \{\top,\#\}$. Moreover, any rule $p \xrightarrow{\texttt{c},\alpha} p'$ in Δ_1 induces the rules $\langle p,x,y\rangle \xrightarrow{\texttt{c},\alpha} \langle p',x,y\rangle$, *except for the rules* $\langle p,\#,y\rangle \xrightarrow{\texttt{r}:Z} \ldots$ and $\langle p,x,\#\rangle \xrightarrow{\texttt{l}:Z} \ldots$ (i.e., $Z_1^{\texttt{c}}$ tests are forbidden after # has been written to c). The next lemma shows that the above reduction is correct.

Lemma 5.6. *S has a run $(p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (p_{\text{fi}},q_{\text{fi}},u',v')$ for some $u' \in U'$ and $v' \in V'$ if, and only if, S' has a run $(\langle p_{\text{in}},\top,\top\rangle,q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (\langle p_{\text{fi}},\#,\#\rangle,q_f,\varepsilon,\varepsilon)$.*

Proof. "⇒": Suppose $C_0 = (p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{\delta_1} C_1 \cdots \xrightarrow{\delta_n} C_n = (p_{\text{fi}},q_{\text{fi}},u',v')$, where $u' \in U'$, $v' \in V'$, is a run of S. We first transform it into a mimicking run $C_0' = (\langle p_{\text{in}},\top,\top\rangle,q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} C_n' = (\langle p_{\text{fi}},\#,\#\rangle,q_{\text{fi}},\#u',\#v')$. This amounts to find some right points for inserting two steps of the forms $(\langle p,\top,y\rangle,q,u,v) \xrightarrow{\texttt{r}!\#} (\langle p,\#,y\rangle,q,u\#,v)$ and $(\langle p,x,\top\rangle,q,u,v) \xrightarrow{\texttt{l}!\#} (\langle p,x,\#\rangle,q,u,v\#)$ (in some order). For the first one, if $u' \neq \varepsilon$ then we find the least index i_1 such that δ_{i_1+1} is some $\texttt{r}!a$ and the written occurrence of a is *permanent*, i.e., $C_{i_1} \xrightarrow{\texttt{r}!a} C_{i_1+1}$ is the step that actually writes the symbol occurring at the head of u' in $C_n = (p_{\text{fi}},q_{\text{fi}},u',v')$; if $u' = \varepsilon$ then we find the least i_1 such that no $\texttt{r}!a$ and no $\texttt{r}:Z$ are performed in $C_j \xrightarrow{\delta_{j+1}} C_{j+1}$ with $j \geq i_1$. For l (and v') we find i_2 analogously. In either case, after i_1 (respectively, i_2) the channel r (respectively, l) is not tested for $\texttt{r}:Z$.

Having $C_0' \xrightarrow{*} C_n' = (\langle p_{\text{fi}},\#,\#\rangle,q_{\text{fi}},\#u',\#v')$, the "cleaning rules" are used to continue with $C_n' \xrightarrow{*} (\langle p_{\text{fi}},\#,\#\rangle,q_f,\varepsilon,\varepsilon)$.

"⇐": Consider a run $C_0 = (\langle p_{\text{in}},\top,\top\rangle,q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (\langle p_{\text{fi}},\#,\#\rangle,q_f,\varepsilon,\varepsilon) = C_n$ of S'. Since Receiver is in state q_{in} at the beginning and in q_f at the end, the Receiver step sequence must be composed of two parts: the first from q_{in} to q_{fi}, and the second from q_{fi} to q_f; the latter corresponds to a sequence of cleaning (reading) rules. The cleaning steps can be commuted after message losses (recall Lemma 5.2(4)), and after Sender's rules (Lemma 5.2(3)) since the first cleaning steps are r?# and l?# and Sender does not test the channels after having written # on them.

Hence we can assume that the run $C_0 \xrightarrow{*} C_n$ of S' has the form

$$C_0 = (\langle p_{\text{in}},\top,\top\rangle,q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} C_m = (\langle p_{\text{fi}},\#,\#\rangle,q_{\text{fi}},\#u',\#v') \xrightarrow{*} C_n = (((\langle p_{\text{fi}},\#,\#\rangle,q_{\text{fi}},\varepsilon,\varepsilon)$$

with only Receiver steps in $C_m \xrightarrow{*} C_n$, which entails $u' \in U'$ and $v' \in V'$. If we now just ignore the two mode-changing steps in the subrun $C_0 \xrightarrow{*} C_m$ (relying on the fact that S' has no N tests) we obtain a new run $C_0 \xrightarrow{*} C_m'$ with $C_m' = (\langle p_{\text{fi}},\top,\top\rangle,q_{\text{fi}},u',v')$. This new run can be directly translated into a run $(p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (p_{\text{fi}},q_{\text{fi}},u',v')$ in S. $\square$

6. Reducing E-E-Reach[Z_1] to G-G-Reach[$Z_1^{\mathtt{l}}$]

We now describe an algorithm deciding E-E-Reach[Z_1] instances, assuming a procedure deciding instances of G-G-Reach[$Z_1^{\mathtt{l}}$]. This is a Turing reduction. The main idea is to partition a run of a UCST[Z_1] system into subruns that do not use the $Z_1^{\mathtt{r}}$ tests (i.e., that only use the $Z_1^{\mathtt{l}}$ tests) and connect them at configurations where $\mathtt{r}$ is known to be empty.

For a UCST $S = (\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$, we let $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ be the subset of configurations in which $\mathtt{r}$ is empty; they are thus of the form (p,q,ε,v). We have put $C = (p,q,u,v) \sqsubseteq C' = (p',q',u',v')$ iff $p = p'$, $q = q'$, $u = u'$, and $v \sqsubseteq v'$. Hence $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ is a well-quasi-ordered by $\sqsubseteq$, unlike Conf.

Slightly abusing terminology, we say that a subset $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is *regular* if there are some state-indexed regular languages $(V_{p,q})_{p\in Q_1,q\in Q_2}$ in $\mathsf{Reg}(\mathtt{M})$ such that $W = \{(p,q,\varepsilon,v) \mid v \in V_{p,q}\}$. Such regular subsets of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ can be finitely represented using, e.g., regular expressions or finite-state automata.

$W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is *upward-closed* (in $\mathit{Conf}_{\mathtt{r}=\varepsilon}$) if $C \in W$, $C \sqsubseteq C'$ and $C' \in \mathit{Conf}_{\mathtt{r}=\varepsilon}$ imply $C' \in W$. It is *downward-closed* if $\mathit{Conf}_{\mathtt{r}=\varepsilon} \smallsetminus W$ is upward-closed. The upward-closure $\uparrow W$ of $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ is the smallest upward-closed set that contains W. A well-known consequence of Higman's Lemma (see Remark 2.2) is that upward-closed and downward-closed subsets of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$ are regular, and that upward-closed subsets can be canonically represented by their finitely many minimal elements.

For $W \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$, we let $\mathrm{Pre}^*(W) \stackrel{\text{def}}{=} \{C \in \mathit{Conf}_{\mathtt{r}=\varepsilon} \mid \exists D \in W : C \xrightarrow{*} D\}$: note that $\mathrm{Pre}^*(W) \subseteq \mathit{Conf}_{\mathtt{r}=\varepsilon}$ by our definition.

Lemma 6.1. *If S is a UCST[$Z_1^{\mathtt{l}}$] system and W is a regular subset of $\mathit{Conf}_{\mathtt{r}=\varepsilon}$, then* $\mathrm{Pre}^*(W)$ *is upward-closed; moreover, given an oracle for G-G-Reach[$Z_1^{\mathtt{l}}$],* $\mathrm{Pre}^*(W)$ *is computable from S and W.*

Proof. We note that $\mathrm{Pre}^*(W)$ is upward-closed since $C \sqsubseteq D$ is equivalent to $D(\xrightarrow{\text{los}})^*C$, hence $D \in \mathrm{Pre}^*(C)$.

We now assume that an oracle for G-G-Reach[$Z_1^{\mathtt{l}}$] is available, and we construct a finite set $F \subseteq \mathrm{Pre}^*(W)$ whose upward-closure $\uparrow F$ is $\mathrm{Pre}^*(W)$. We build up F in steps, starting with $F_0 = \varnothing$; clearly $\uparrow F_0 = \varnothing \subseteq \mathrm{Pre}^*(W)$. The $(i+1)$th iteration, starting with F_i, proceeds as follows.

We put $W' \stackrel{\text{def}}{=} \mathit{Conf}_{\mathtt{r}=\varepsilon} \smallsetminus \uparrow F_i$; note that W' is regular. We check whether there exist some $C \in W'$ and $D \in W$ such that $C \xrightarrow{*} D$; this can be decided using the oracle (it is a finite disjunction of G-G-Reach[$Z_1^{\mathtt{l}}$] instances, obtained by considering all possibilities for Sender and Receiver states). If the answer is "no", then $\uparrow F_i = \mathrm{Pre}^*(W)$; we then put $F = F_i$ and we are done.

Otherwise, the answer is "yes" and we look for some concrete $C \in W'$ s.t. $C \xrightarrow{*} D$ for some $D \in W$. This can be done by enumerating all $C \in W'$ and by using the oracle for G-G-Reach[$Z_1^{\mathtt{l}}$] again. We are bound to eventually find such a C since $W' \cap \mathrm{Pre}^*(W)$ is not empty.

Once some C is found, we set $F_{i+1} \stackrel{\text{def}}{=} F_i \cup \{C\}$. Clearly F_{i+1}, and so $\uparrow F_{i+1}$, is a subset of $\mathrm{Pre}^*(W)$. By construction, $\uparrow F_0 \subsetneq \uparrow F_1 \subsetneq \uparrow F_2 \subsetneq \cdots$ is a strictly increasing sequence of upward-closed sets. By the well-quasi-ordering property, this sequence cannot be extended indefinitely: eventually we will have $\uparrow F_i = \mathrm{Pre}^*(W)$, signalled by the answer "no". □

Lemma 6.2. *E-E-Reach[Z_1] is Turing reducible to G-G-Reach[$Z_1^{\mathtt{l}}$].*

Proof. Assume $S = (\{\mathtt{r},\mathtt{l}\},\mathtt{M},Q_1,\Delta_1,Q_2,\Delta_2)$ is a UCST[Z_1], and we ask if there is a run $C_{\text{in}} = (p_{\text{in}},q_{\text{in}},\varepsilon,\varepsilon) \xrightarrow{*} (p_{\text{fi}},q_{\text{fi}},\varepsilon,\varepsilon) = C_{\text{fi}}$. By S' we denote the UCST[$Z_1^{\mathtt{l}}$] system arising from S by removing all $Z_1^{\mathtt{r}}$ rules. Hence Lemma 6.1 applies to S'. The set of configurations of S and S' is the same, so there is no ambiguity in using the notation Conf and $\mathit{Conf}_{\mathtt{r}=\varepsilon}$.

We aim at computing $\text{Pre}^*(\{C_{\text{fi}}\})$ for S. For $k \geq 0$, let $T_k \subseteq Conf_{\mathtt{r}=\varepsilon}$ be the set of $C \in Conf_{\mathtt{r}=\varepsilon}$ for which there is a run $C \xrightarrow{*} C_{\text{fi}}$ of S with at most k steps that are $Z_1^{\mathtt{r}}$ tests; hence $\uparrow\{C_{\text{fi}}\} \subseteq T_0$ (by message losses). For each k, T_k is upward-closed and $T_k \subseteq T_{k+1}$. Defining $T = \bigcup_{k\in\mathbb{N}} T_k$, we note that $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ iff $C_{\text{in}} \in T$. Since $Conf_{\mathtt{r}=\varepsilon}$ is well quasi-ordered, the sequence $T_0 \subseteq T_1 \subseteq T_2 \subseteq \cdots$ eventually stabilizes; hence there is n such that $T_n = T_{n+1}$, which implies that $T_n = T$.

By Lemma 6.1, and using an oracle for G-G-Reach[Z_1^1], we can compute $\text{Pre}^*_{S'}(\{C_{\text{fi}}\})$, where the "$S'$" subscript indicates that we consider runs in S', not using $Z_1^{\mathtt{r}}$ tests. Hence $T_0 = \text{Pre}^*_{S'}(\{C_{\text{fi}}\})$ is computable. Given T_k, we compute T_{k+1} as follows. We put

$$\begin{aligned} T'_k &= \{C \in Conf_{\mathtt{r}=\varepsilon} \mid \exists D \in T_k : C \xrightarrow{\mathtt{r}:Z} D\} \\ &= \{(p,q,\varepsilon,w) \mid \exists p' \in Q_1 : p \xrightarrow{\mathtt{r}:Z} p' \in \Delta_1 \text{ and } (p',q,\varepsilon,w) \in T_k\}\,. \end{aligned}$$

Thus $T'_k \subseteq Conf_{\mathtt{r}=\varepsilon}$ is the set of configurations from which one can reach T_k with one $Z_1^{\mathtt{r}}$ step. Clearly T'_k is upward-closed (since T_k is) and can be computed from a finite representation of T_k, e.g., its minimal elements. Then $T_{k+1} = T_k \cup \text{Pre}^*_{S'}(T'_k)$, and we use Lemma 6.1 again to compute it.

Iterating the above process, we compute the sequence $T_0, T_1, \ldots$, until the first n such that $T_n = T_{n+1}$ (recall that $T_n = T$ then). Finally we check if $C_{\text{in}} \in T_n$. ☐

7. Reducing E-E-Reach[Z_1^1] to a Post Embedding Problem

As stated in Theorem 5.1 (see also Fig. 3), our series of reductions from G-G-Reach[Z_1, N_1] to E-E-Reach[Z_1] also reduces G-G-Reach[Z_1^1] to E-E-Reach[Z_1^1]; this can be easily checked by recalling that the respective reductions do not introduce new tests. In Subsection 7.1 we show a (polynomial) many-one reduction from E-E-Reach[Z_1^1] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, a generalization of Post's Embedding Problem. Since $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ was shown decidable in [29], our proof of Theorem 4.1 will be thus completed. We also add Subsection 7.2 that shows a simple reduction in the opposite direction, from $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ to E-E-Reach[Z_1^1].

7.1. E-E-Reach[Z_1^1] reduces to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$.

Definition 7.1 (Post embedding with partial codirectness [29]). $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the question, given two finite alphabets Σ, Γ, two morphisms $u, v : \Sigma^* \to \Gamma^*$, and two regular languages $R, R' \in \mathsf{Reg}(\Sigma)$, whether there is $\sigma \in R$ (called a *solution*) such that $u(\sigma) \sqsubseteq v(\sigma)$, and such that furthermore $u(\sigma') \sqsubseteq v(\sigma')$ for all suffixes σ' of σ that belong to R'.

The above definition uses the same subword relation, denoted $\sqsubseteq$, that captures message losses. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and PEP (which is the special case where $R' = \varnothing$) are a variant of Post's Correspondence Problem, where the question is whether there exists $\sigma \in \Sigma^+$ such that $u(\sigma) = v(\sigma)$; see also [7] for applications in graph logics.

Lemma 7.2. *E-E-Reach[Z_1^1] reduces to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ (via a polynomial reduction).*

We now prove the lemma. The reduction from E-E-Reach[Z_1^1] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ extends an earlier reduction from UCS to PEP [17]. In our case the presence of Z_1^1 tests creates new difficulties.

We fix an instance $S = (\{\mathtt{r},\mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, \varepsilon, \varepsilon)$, $C_{\text{fi}} = (p_{\text{fi}}, q_{\text{fi}}, \varepsilon, \varepsilon)$ of E-E-Reach[Z_1^1], and we construct a $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $\mathcal{P} = (\Sigma, \Gamma, u, v, R, R')$ intended to express the existence of a run from C_{in} to C_{fi}.

We first put $\Sigma \stackrel{\text{def}}{=} \Delta_1 \cup \Delta_2$ and $\Gamma \stackrel{\text{def}}{=} \mathtt{M}$ so that words $\sigma \in \Sigma^*$ are sequences of rules of S, and their images $u(\sigma), v(\sigma) \in \Gamma^*$ are sequences of messages. With any $\delta \in \Sigma$, we associate $\mathit{write\_}\mathtt{r}(\delta)$ defined by $\mathit{write\_}\mathtt{r}(\delta) = x$ if δ is a Sender rule of the form $p \xrightarrow{\mathtt{r}!x} p'$, and $\mathit{write\_}\mathtt{r}(\delta) = \varepsilon$ in all other cases. This is extended to sequences with $\mathit{write\_}\mathtt{r}(\delta_1 \cdots \delta_n) = \mathit{write\_}\mathtt{r}(\delta_1) \cdots \mathit{write\_}\mathtt{r}(\delta_n)$. In a similar way we define $\mathit{write\_}\mathtt{l}(\sigma) \in \mathtt{M}^*$, the message sequence written to $\mathtt{l}$ by the rule sequence σ, and $\mathit{read\_}\mathtt{r}(\sigma)$ and $\mathit{read\_}\mathtt{l}(\sigma)$, the sequences read by σ from $\mathtt{r}$ and $\mathtt{l}$, respectively. We define $E_{\mathtt{r}} \in \mathsf{Reg}(\Sigma)$ as $E_{\mathtt{r}} \stackrel{\text{def}}{=} E_1 \cup E_2$ where

$$E_1 \stackrel{\text{def}}{=} \{\delta \in \Sigma \mid \mathit{write\_}\mathtt{r}(\delta) = \mathit{read\_}\mathtt{r}(\delta) = \varepsilon\}\,,$$
$$E_2 \stackrel{\text{def}}{=} \{\delta_1\delta_2 \in \Sigma^2 \mid \mathit{write\_}\mathtt{r}(\delta_1) = \mathit{read\_}\mathtt{r}(\delta_2) \neq \varepsilon\}\,.$$

In other words, E_1 gathers the rules that do not write to or read from $\mathtt{r}$, and E_2 contains all pairs of Sender/Receiver rules that write/read the same letter to/from $\mathtt{r}$.

Let now $P_1 \subseteq \Delta_1^*$ be the set of all sequences of Sender rules of the form $p_{\text{in}} = p_0 \dashrightarrow p_1 \dashrightarrow p_2 \cdots \dashrightarrow p_n = p_{\text{fi}}$, i.e., the sequences corresponding to paths from p_{in} to p_{fi} in the graph defined by Q_1 and Δ_1. Similarly, let $P_2 \subseteq \Delta_2^*$ be the set of all sequences of Receiver rules that correspond to paths from q_{in} to q_{fi}. Since P_1 and P_2 are defined by finite-state systems, they are regular languages. We write $P_1 \| P_2$ to denote the set of all interleavings (shuffles) of a word in P_1 with a word in P_2. This operation is regularity-preserving, so $P_1 \| P_2 \in \mathsf{Reg}(\Sigma)$. Let $T_1 \subseteq \Delta_1$ be the set of all Sender rules that test the emptiness of $\mathtt{l}$ (which are the only test rules in S). We define R and R' as the following regular languages:

$$R = E_{\mathtt{r}}^* \cap (P_1 \| P_2), \qquad\qquad R' = T_1 \cdot \big(\Delta_1 \cup \Delta_2\big)^*.$$

Finally, the morphisms $u, v : \Sigma^* \to \Gamma^*$ are given by $u \stackrel{\text{def}}{=} \mathit{read\_}\mathtt{l}$ and $v \stackrel{\text{def}}{=} \mathit{write\_}\mathtt{l}$. This finishes the construction of the $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $\mathcal{P} = (\Sigma, \Gamma, u, v, R, R')$.

We will now prove the correctness of this reduction, i.e., show that S has a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ if, and only if, $\mathcal{P}$ has a solution. Before starting with the proof itself, let us illustrate some aspects of the reduction by considering a schematic example (see Fig. 7).

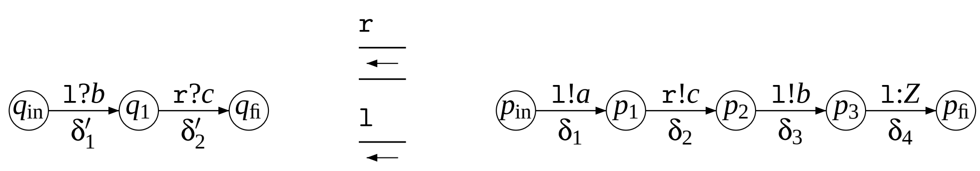

Figure 7: A schematic UCST[Z_1^1] instance

Let us consider $\sigma_{\text{sol}} = \delta_1\delta'_1\delta_2\delta'_2\delta_3\delta_4$ and check whether it is a solution of the $\mathcal{P}$ instance obtained by our reduction. For this, one first checks that $\sigma_{\text{sol}} \in R$, computes $u(\sigma_{\text{sol}}) = \mathit{read\_}\mathtt{l}(\sigma_{\text{sol}}) = b$ and check that $b \sqsubseteq v(\sigma_{\text{sol}}) = \mathit{write\_}\mathtt{l}(\sigma_{\text{sol}}) = ab$. There remains to check the suffixes of σ_{sol} that belong to R', i.e., that start with a $\mathtt{l}{:}Z$ rule. Here, only $\sigma' = \delta_4$ is in R', and indeed $u(\sigma') = \varepsilon \sqsubseteq v(\sigma')$. Thus σ_{sol} is a solution.

However, a solution like σ_{sol} does not directly correspond to a run of S. For instance, any run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ in the system from Fig. 7 must use δ_3 (write $\mathtt{b}$ on $\mathtt{l}$) before δ'_1 (read it).

Reciprocally, a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ does not directly lead to a solution. For example, on the same system the following run

$$C_{\text{in}} \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \xrightarrow{\delta_3} C_3 = (p_3, q_{\text{in}}, c, ab) \xrightarrow{\text{los}} C_4 = (p_3, q_{\text{in}}, c, b) \xrightarrow{\delta_1'} C_5 \xrightarrow{\delta_4} C_6 \xrightarrow{\delta_2'} C_{\text{fi}} \qquad (\pi)$$

has an action in "$C_3 \xrightarrow{\text{los}} C_4$" that is not accounted for in Σ and cannot appear in solutions of $\mathcal{P}$. Also, the Σ-word $\sigma_\pi = \delta_1\delta_2\delta_3\delta_1'\delta_4\delta_2'$ obtained from π is not a solution. It belongs to $P_1 \| P_2$ but not to $E_{\mathtt{r}}^*$ (which requires that each occurrence of δ_2 is immediately followed by some $. \xrightarrow{\mathtt{r}?c} .$ rule). Note that σ_{sol} had δ_2 followed by δ_2', but it is impossible in a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ to have δ_2 immediately followed by δ_2'.

With these issues in mind, we introduce a notion bridging the difference between runs of S and solutions of $\mathcal{P}$. We call $\sigma \in (\Delta_1 \cup \Delta_2)^*$ a *pre-solution* if the following five conditions hold:

(c1) $\sigma \in P_1 \| P_2$;
(c2) $\textit{read\_}\mathtt{r}(\sigma) = \textit{write\_}\mathtt{r}(\sigma)$;
(c3) $\textit{read\_}\mathtt{r}(\sigma_1)$ is a prefix of $\textit{write\_}\mathtt{r}(\sigma_1)$ for each prefix σ_1 of σ;
(c4) $\textit{read\_}\mathtt{l}(\sigma) \sqsubseteq \textit{write\_}\mathtt{l}(\sigma)$;
(c5) $\textit{read\_}\mathtt{l}(\sigma_2) \sqsubseteq \textit{write\_}\mathtt{l}(\sigma_2)$ for each factorization $\sigma = \sigma_1\delta\sigma_2$ where $\delta \in T_{\mathtt{l}}$ (i.e., δ is a $\mathtt{l}$:Z rule).

A pre-solution σ has a *Receiver-advancing switch* if $\sigma = \sigma_1\delta\delta'\sigma_2$ where δ is a Sender rule, δ' is a Receiver rule, and $\sigma' = \sigma_1\delta'\delta\sigma_2$ is again a pre-solution. A *Receiver-postponing switch* is defined analogously, for δ being a Receiver rule and δ' being a Sender rule. For example, the sequence σ_π above is a pre-solution. It has a Receiver-advancing switch on δ_3 and δ_1', and one on δ_4 and δ_2'. Note that when σ is a pre-solution, checking whether a potential Receiver-advancing or Receiver-postponing switch leads again to a pre-solution only requires checking (c3) or, respectively, (c5). Considering another example, σ_{sol}, being a solution is a pre-solution. It has two Receiver-postponing switches but only one Receiver-advancing switch since switching δ_2 and δ_2' does not maintain (c3).

It is obvious that if there is a pre-solution σ then there is an *advance-stable pre-solution* σ', which means that σ' has no Receiver-advancing switch; there is also a *postpone-stable pre-solution* σ'' which has no Receiver-postponing switch.

Claim 7.3. Any advance-stable pre-solution σ is in $E_{\mathtt{r}}^*$, and it is thus a solution of $\mathcal{P}$.

Proof. Let us write an advance-stable pre-solution σ as $\sigma_1\sigma_2$ where σ_1 is the longest prefix such that $\sigma_1 \in E_{\mathtt{r}}^*$; hence $\textit{read\_}\mathtt{r}(\sigma_1) = \textit{write\_}\mathtt{r}(\sigma_1)$ by the definition of $E_r = E_1 \cup E_2$. Now suppose $\sigma_2 \neq \varepsilon$. Then $\sigma_2 = \delta_1\delta_2\cdots\delta_k$ where $\delta_1 \notin E_1$. Since $\textit{read\_}\mathtt{r}(\sigma_1) = \textit{write\_}\mathtt{r}(\sigma_1)$, δ_1 must be of the form $. \xrightarrow{\mathtt{r}!x} .$ to guarantee (c3). Let us pick the smallest ℓ such that $\delta_\ell = . \xrightarrow{\mathtt{r}?x} .$ —which must exist by (c2)— and note that $\ell > 2$ since $\delta_1\delta_2 \notin E_2$ by maximality of σ_1. If we now pick the least j in $\{1, \ldots, \ell-1\}$ such that δ_j is a Sender rule and δ_{j+1} is a Receiver rule, then switching δ_j and δ_{j+1} leads again to a pre-solution as can be checked by inspecting (c1–c5). This contradicts the assumption that σ is an advance-stable pre-solution. $\square$

Claim 7.4. If $\sigma = \delta_1 \ldots \delta_n$ is a postpone-stable pre-solution, S has a run of the form $C_{\text{in}} \xrightarrow{\delta_1} \xrightarrow{\text{los}^*} \cdots \xrightarrow{\delta_n} \xrightarrow{\text{los}^*} C_{\text{fi}}$.

Proof. Assume that we try to fire $\delta_1, \ldots, \delta_n$ in that order, starting from C_{in}, and sometimes inserting message losses. Since σ belongs to $P_1 \| P_2$, we can only fail because at some point the current channel contents does not allow the test or the read action carried by the next rule to be fired, i.e., not because we end up in a control state that does not carry the next rule.

So let us consider channel contents, starting with $\mathtt{r}$. For $i = 0,\ldots,n$, let $x_i = \mathit{read}\_\mathtt{r}(\delta_1 \ldots \delta_i)$ and $y_i = \mathit{write}\_\mathtt{r}(\delta_1 \ldots \delta_i)$. Since σ satisfies (c3), y_i is some $x_i x'_i$ (and $x'_0 = \varepsilon$). One can easily verify by induction on i that after firing $\sigma_1 \ldots \sigma_i$ from C_{in}, $\mathtt{r}$ contains exactly x'_i. In fact (c3) implies that if δ_{i+1} reads on $\mathtt{r}$, it must read the first letter of x'_i (and δ_{i+1} cannot be a read on $\mathtt{r}$ when $x'_i = \varepsilon$).

Now, regarding the contents of $\mathtt{l}$, we can rely on (c4) and conclude that the actions in σ write on $\mathtt{l}$ everything that they (attempt to) read, but we do not know that messages are written *before* they are needed for reading, i.e., we do not have an equivalent of (c3) for $\mathtt{l}$. For this, we rely on the assumption that σ is postpone-stable. Write σ under the form $\sigma_0 z_1 \sigma_1 z_2 \sigma_2 \ldots z_k \sigma_k$ where the z_i's are the test rules from T_1, and where the σ_i's factors contain no test rules. Note that, inside a σ_i, all Sender rules occur before all Receiver rules thanks to postpone-stability.

We claim that $\mathit{read}\_\mathtt{l}(\sigma_i) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_i)$ for all $i = 0,\ldots,k$: assume, by way of contradiction, that $\mathit{read}\_\mathtt{l}(\sigma_i) \not\sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_i)$ for some $i \in \{0,\ldots,k\}$ and let δ be the last rule in σ_i. Necessarily δ is a reading rule. Now (c4) and (c5) entail $i < k$ and $\mathit{read}\_\mathtt{l}(\sigma_i z_{i+1} \sigma_{i+1} \ldots \sigma_k) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_i z_{i+1} \sigma_{i+1} \ldots \sigma_k)$. Then $\mathit{read}\_\mathtt{l}(\sigma_i) \not\sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_i)$ entails

$$\mathit{read}\_\mathtt{l}(\delta z_{i+1} \sigma_{i+1} \ldots z_k \sigma_k) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_{i+1} \ldots z_k \sigma_k)\,. \tag{$\star\star$}$$

There is now a Receiver-postponing switch since ($\star\star$) ensures that (c5) holds after switching δ and z_{i+1}, which contradicts the assumption that σ is postpone-stable.

Now, with $\mathit{read}\_\mathtt{l}(\sigma_i) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_i)$, it is easy to build a run $C_{\text{in}} \xrightarrow{\delta_1} \xrightarrow{\text{los}^*} \cdots \xrightarrow{\delta_n} \xrightarrow{\text{los}^*} C_{\text{fi}}$ and guarantee that $\mathtt{l}$ is empty before firing any z_i rule. □

We now see that our reduction is correct. Indeed, if $C_{\text{in}} \xrightarrow{\sigma} C_{\text{fi}}$ is a run of S then σ with all occurrences of los removed is a pre-solution; and there is also an advance-stable pre-solution, i.e., a solution of $\mathcal{P}$. On the other hand, if σ is a solution of $\mathcal{P}$ then σ is a pre-solution, and there is also a postpone-stable pre-solution, which corresponds to a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ of S. This finishes the proof of Lemma 7.2, and of Theorem 4.1.

7.2. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ reduces to E-E-Reach[$Z_1^{\mathtt{l}}$].

We now prove a converse of Lemma 7.2, thus showing that $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and E-E-Reach[$Z_1^{\mathtt{l}}$] are equivalent problems. Actually, $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ can be easily reduced to E-E-Reach[$Z_i^{\mathtt{c}}$] for any $i \in \{1,2\}$ and $\mathtt{c} \in \mathtt{Ch}$, but we only show a reduction for $i = 1$ and $\mathtt{c} = \mathtt{l}$ explicitly. (The other reductions would be analogous.)

Lemma 7.5. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ *reduces to E-E-Reach[$Z_1^{\mathtt{l}}$] (via a polynomial reduction).*

Proof. Given a $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$-instance $(\Sigma, \Gamma, u, v, R, R')$, we construct a UCST[$Z_1^{\mathtt{l}}$] system (denoted S) with distinguished states $p_{\text{in}}, p_{\text{fi}}, q_{\text{loop}}$, such that

$$\text{the instance has a solution iff } S \text{ has a run } (p_{\text{in}}, q_{\text{loop}}, \varepsilon, \varepsilon) \xrightarrow{*} (p_{\text{fi}}, q_{\text{loop}}, \varepsilon, \varepsilon)\,. \tag{$\star\star\star$}$$

The idea is simple: Sender nondeterministically guesses a solution σ, writing $u(\sigma)$ on $\mathtt{r}$ and $v(\sigma)$ on $\mathtt{l}$, and Receiver validates it, by reading identical sequences from $\mathtt{r}$ and $\mathtt{l}$ (some messages from $\mathtt{l}$ might be lost). We now make this idea more precise.

Let M and M' be deterministic FSAs recognizing R and the *complement of* R', respectively. Sender stepwise nondeterministically generates $\sigma = a_1 a_2 \ldots, a_m$, while taking the "commitment" that σ belongs to R; concretely, after generating $a_1 a_2 \ldots a_i$ Sender also remembers the state reached by M via $a_1 a_2 \ldots a_i$, and Sender cannot enter p_{fi} when the current state of M is non-accepting. Moreover, for each $i \in \{1,2,\ldots,m\}$, i.e., at every step, Sender might decide to take a further commitment, namely that $a_i a_{i+1} \ldots, a_m \notin R'$; for each such commitment Sender starts a new copy of M', remembering the states visited by M' via $a_i a_{i+1} \ldots a_m$, and it cannot enter p_{fi} if a copy of M' is in a non-accepting state. Though we do not bound the number of copies of M', it suffices to remember just a bounded information, namely the set of current states of all these copies.

When generating a_i, Sender writes $u(a_i)$ on $\mathtt{r}$ and $v(a_i)$ on $\mathtt{l}$. To check that $\mathtt{r}$ contains a subword of $\mathtt{l}$, Receiver behaves as in Fig. 8 (that illustrates another reduction). So far we have guaranteed that there is a run $(p_{\text{in}}, q_{\text{loop}}, \varepsilon, \varepsilon) \xrightarrow{*} (p_{\text{fi}}, q_{\text{loop}}, \varepsilon, \varepsilon)$ iff there is $\sigma = a_1 a_2 \ldots, a_m \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ (using the lossiness of $\mathtt{l}$ where $v(\sigma)$ has been written).

We finish by adding a modification guaranteeing $u(a_i a_{i+1} \ldots, a_m) \sqsubseteq v(a_i a_{i+1} \ldots, a_m)$ for each $i \in \{1, 2, \ldots, m\}$ where Sender does not commit to $a_i a_{i+1} \ldots, a_m \notin R'$. For such steps, and before writing $u(a_i)$ and $v(a_i)$, Sender must simply wait until $\mathtt{l}$ is empty, i.e., Sender initiates step i by (nondeterministically) either committing to $a_i a_{i+1} \ldots, a_m \notin R'$ or by taking a $Z_1^{\mathtt{l}}$-step.

It is now a routine exercise to verify that $(\star\star\star)$ holds. □

Remark 7.6 (On complexity)**.** Based on known results on the complexity of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ (see [39, 29, 28]), our reductions prove that reachability for UCST$[Z,N]$ is $\mathbf{F}_{\omega^\omega}$-complete, using the ordinal-recursive complexity classes introduced in [38]. □

8. Two undecidable problems for UCST$[Z,N]$

The main result of this article is Theorem 4.1, showing the decidability of the reachability problem for UCST$[Z,N]$. In this section we argue that the emptiness and non-emptiness tests ("Z" and "N") strictly increase the expressive power of UCSes. We do this by computational arguments, namely by exhibiting two variants of the reachability problem that are undecidable for UCST$[Z,N]$. Since these variants are known to be decidable for plain UCSes (with no tests), we conclude that there is no effective procedure to transform a UCST$[Z,N]$ into an equivalent UCS in general. Subsection 8.1 deals with the problem of *recurrent reachability* of a control state. In Subsection 8.2 we consider the usual reachability problem but we assume that *messages can be lost only during writing* to $\mathtt{l}$ (i.e., we assume that channel $\mathtt{l}$ is reliable and that the unreliability is limited to the writing operation).

8.1. **Recurrent reachability.**

The *Recurrent Reachability Problem* asks, when given S and its states $p_{\text{in}}, q_{\text{in}}, p, q$, whether S has an *infinite* run $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, \varepsilon, \varepsilon) \xrightarrow{*} (p, q, u_1, v_1) \xrightarrow{+} (p, q, u_2, v_2) \xrightarrow{+} (p, q, \ldots) \cdots$ visiting the pair (p, q) infinitely often (NB: with no constraints on channel contents), called a "*pq^∞-run*" for short.

The next theorem separates UCSes from UCSTs, even from UCST$[Z_1^{\mathtt{r}}]$, i.e., UCSTs where the only tests are emptiness tests on $\mathtt{r}$ by Sender. It implies that $Z_1^{\mathtt{r}}$ tests cannot be simulated by UCSes.

Theorem 8.1. *Recurrent reachability is decidable for UCSes, and is Σ_1^0-complete (hence undecidable) for UCST$[Z_1^{\mathtt{r}}]$.*

We start with the upper bounds. Consider a UCST$[Z_1^{\mathtt{r}}]$ system S and assume it admits a pq^∞-run π. There are three cases:

case 1: If π uses infinitely many Z tests, it can be written under the form

$$C_{\text{in}} \xrightarrow{*} D_1 \xrightarrow{\mathtt{r}:Z} \xrightarrow{*} (p,q,\ldots) \xrightarrow{*} D_2 \xrightarrow{\mathtt{r}:Z} \xrightarrow{*} (p,q,\ldots) \cdots \xrightarrow{*} D_n \xrightarrow{\mathtt{r}:Z} \xrightarrow{*} (p,q,\ldots) \cdots$$

Observe that $D_1, D_2, \ldots$ belong to $Conf_{\mathtt{r}=\varepsilon}$ since they allow a $\mathtt{r}$:Z test. By Higman's Lemma, there exists two indexes $i < j$ such that $D_i \sqsubseteq D_j$. Then $D_j (\xrightarrow{\text{los}})^* D_i \xrightarrow{*} (p,q,\ldots) \xrightarrow{*} D_j$ and we conclude that S also has a "looping" pq^∞-run, witnessed by a finite run of the form $C_{\text{in}} \xrightarrow{*} (p,q,u,v) \xrightarrow{+} (p,q,u,v)$.

case 2: Otherwise, if π only uses finitely many Z tests, it can be written under the form $C_{\text{in}} \xrightarrow{*} C = (p,q,u,v) \to \cdots$ such that no test occur after C. After C, any step by Sender can be advanced before Receiver steps and message losses, according to Lemma 5.2(3). Assuming that π uses infinitely many Sender steps, we conclude that S has a pq^∞ run that eventually only uses Sender rules (but no Z tests). At this point, we can forget about the contents of the channels (they are not read or tested anymore). Hence a finite witness for such pq^∞-runs is obtained by the combination of a finite run $C_{\text{in}} \xrightarrow{*} (p,q,u,v)$ and a loop $p = p_1 \xrightarrow{\delta_1} p_2 \xrightarrow{\delta_2} \cdots p_n \xrightarrow{\delta_n} p_1$ in Sender's rules that does not use any testing rule.

case 3: The last possibility is that π uses only finitely many Sender rules. In that case, the contents of the channels is eventually fixed hence there is a looping pq^∞-run of the form $C_{\text{in}} \xrightarrow{*} C = (p,q,u,v) \xrightarrow{+} C$ such that the loop from C to C only uses Receiver rules. A finite witness for such cases is a finite run $C_{\text{in}} \xrightarrow{*} (p,q,u,v)$ combined with a loop $q = q_1 \xrightarrow{\delta_1} q_2 \xrightarrow{\delta_2} \cdots q_n \xrightarrow{\delta_n} q_1$ in Receiver's rules that only uses rules reading ε.

Only the last two cases are possible for UCSes: for these systems, deciding Recurrent reachability reduces to deciding whether some $(p,q,...)$ is reachable and looking for a loop (necessarily with no tests) starting from p in Sender's graph, or a loop with no reads starting from q in Receiver's graph.

For UCST[$Z_1^{\mathtt{r}}$], one must also consider the general looping "case 1", i.e., $\exists u,v : C_{\text{in}} \xrightarrow{*} (p,q,u,v) \xrightarrow{+} (p,q,u,v)$. Since reachability is decidable, this case is in Σ_1^0, as is Recurrent reachability for UCST[$Z_1^{\mathtt{r}}$].

Now for the lower bound. We prove Σ_1^0-hardness by a reduction from the looping problem for semi-Thue systems.

A *semi-Thue system* $T = (\Gamma, R)$ consists of a finite alphabet Γ and a finite set $R \subseteq \Gamma^* \times \Gamma^*$ of *rewrite rules*; we write $\alpha \to \beta$ instead of $(\alpha,\beta) \in R$. The system gives rise to a *one-step rewrite relation* $\to_R \subseteq \Gamma^* \times \Gamma^*$ as expected: $x \to_R y \overset{\text{def}}{\Leftrightarrow} x$ and y can be factored as $x = z\alpha z'$ and $y = z\beta z'$ for some rule $\alpha \to \beta$ and some strings $z, z' \in \Gamma^*$. As usual, we write $x \xrightarrow{+}_R y$ if x can be rewritten into y by a nonempty sequence of steps.

We say that $T = (\Gamma, R)$ is *length-preserving* if $|\alpha| = |\beta|$ for each rule in R, and that it *has a loop* if there is some $x \in \Gamma^*$ such that $x \xrightarrow{+}_R x$. The following is standard (since the one-step relation between Turing machine configurations can be captured by finitely many length-preserving rewrite rules).

Fact 8.2. The question whether a given length-preserving semi-Thue system has a loop is Σ_1^0-complete.

We now reduce the existence of a loop for length-preserving semi-Thue systems to the recurrent reachability problem for UCST[$Z_1^{\mathtt{r}}$].

Let $T = (\Gamma, R)$ be a given length-preserving semi-Thue system. We construct a UCST S, with message alphabet $\mathtt{M} \overset{\text{def}}{=} \Gamma \uplus \{\#\}$. The reduction is illustrated in Fig. 8, assuming $\Gamma = \{a, b\}$. The resulting S behaves as follows:

(a) Sender starts in state p_{in}, begins by nondeterministically sending some $y_0 \in \Gamma^*$ on $\mathtt{l}$, then moves to state p_{loop}. In state p_{loop}, Sender performs the following steps in succession:

(1) check that (equivalently, wait until) $\mathtt{r}$ is empty;
(2) send # on $\mathtt{l}$;
(3) nondeterministically send a string $z \in \Gamma^*$ on both $\mathtt{l}$ and $\mathtt{r}$;

(4) nondeterministically choose a rewrite rule $\alpha \to \beta$ (from R) and send α on `r` and β on `l`;
(5) nondeterministically send a string $z' \in \Gamma^*$ on both `l` and `r`;
(6) send # on `r`;
(7) go back to p_{loop} (and repeat 1–7).

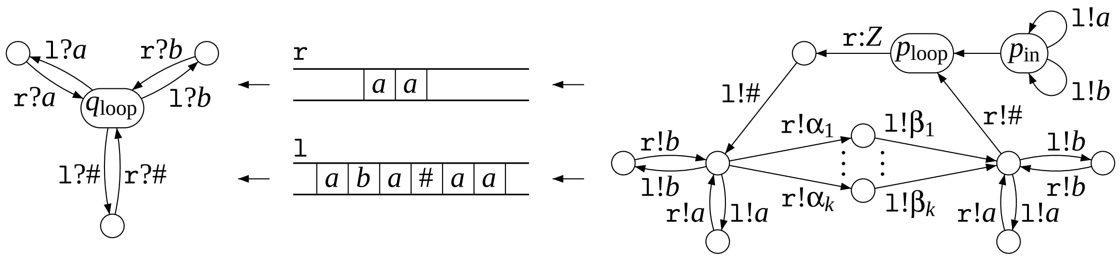

Figure 8: Solving the looping problem for semi-Thue systems

The above loop 1–7 can be also summarized as: check that `r` is empty, nondeterministically guess two strings x and y such that $x \to_R y$, writing x# on `r` and #y on `l`.
(b) Receiver starts in state q_{loop} from where it reads any pair of identical symbols from `r` and `l`, returns to q_{loop}, and repeats this indefinitely.

Claim 8.3 (Correctness of the reduction)**.** S has an infinite run starting from $C_{\text{in}} = (p_{\text{in}}, q_{\text{loop}}, \varepsilon, \varepsilon)$ and visiting the control pair $(p_{\text{loop}}, q_{\text{loop}})$ infinitely often if, and only if, $x \xrightarrow{+}_R x$ for some $x \in \Gamma^*$.

Proof. For the "$\Leftarrow$" direction we assume that T has a loop $x = x_0 \to_R x_1 \to_R \ldots \to_R x_n = x$ with $n > 0$. Let $C_i \stackrel{\text{def}}{=} (p_{\text{loop}}, q_{\text{loop}}, \varepsilon, x_i)$. S obviously has a run $C_{\text{in}} \xrightarrow{*} C_0$, sending x_0 on `l`. For each $i \geq 0$, S has a run $C_i \xrightarrow{+} C_{i+1}$: it starts with appending the pair $x_i \to_R x_{i+1}$ on the channels, hence visiting $(.,., x_i\#, x_i\#x_{i+1})$, from which Receiver can read the x_i# prefix on both channels, thus reaching C_{i+1}. Note that no messages are lost in these runs. Chaining them gives an infinite run that visits $(p_{\text{loop}}, q_{\text{loop}})$ infinitely many times.

For the "$\Rightarrow$" direction, we assume that S has an infinite run starting from C_{in} that visits $(p_{\text{loop}}, q_{\text{loop}})$ infinitely often. Since Sender checks the emptiness of `r` before running through its loop, we conclude that no # character written to `l` is lost during the run. Let y_0 be written on `l` before the first visit of p_{loop}; for $i \geq 1$, let (x_i, y_i) be the pair of strings guessed by Sender during the ith iteration of its loop 1–7 (x_i written on `r` and y_i on `l`). Receiver can only empty the reliable channel `r` if $x_i \sqsubseteq y_{i-1}$ for all $i \geq 1$. This implies $|x_i| \leq |y_{i-1}|$. We also have $|x_i| = |y_i|$ since T is length-preserving. Therefore eventually, say for all $i \geq n$, all x_i and y_i have the same length. Then $x_i = y_{i-1}$ for $i > n$ (since $x_i \sqsubseteq y_{i-1}$ and $|x_i| = |y_{i-1}|$). Hence T admits an infinite derivation of the form

$$x_n \to_R y_n = x_{n+1} \to_R y_{n+1} = x_{n+2} \to_R \cdots$$

Since there are only finitely many strings of a given length, there are two positions $m' > m \geq n$ such that $x_m = x_{m'}$; hence T has a loop $x_m \xrightarrow{+}_R x_m$. $\square$

8.2. **Write-lossy semantics.** As another illustration of the power of tests, we consider UCSTs with *write-lossy semantics*, that is, UCSTs with the assumption that messages are only lost during steps that write them to `l`. Once messages are in `l`, they are never lost. If we start with the empty channel `l` and we only allow the emptiness tests on `l`, then any computation in normal lossy semantics can be mimicked by a computation in write-lossy semantics: any occurrence of a message that gets finally lost will simply not be written. Adding the non-emptiness test makes a difference, since the reachability problem becomes undecidable.

We now make this reasoning more formal, using the new transition relation $C \to_{\text{wrlo}} C'$ that is intermediary between the reliable and the lossy semantics.

Each `l`-writing rule δ of the form $p \xrightarrow{1!x} p'$ in a UCST S will give rise to *write-lossy steps* of the form $(p,q,u,v) \xrightarrow{\text{wrlo}} (p',q,u,v)$, where δ is performed but nothing is actually written. We write $C \to_{\text{wrlo}} C'$ when there is a reliable or a write-lossy step from C to C', and use $C \to_{\text{rel}} C'$ and $C \to_{\text{los}} C'$ to denote the existence of a reliable step, and respectively, of a reliable or a lossy step. Then $\to_{\text{rel}} \subseteq \to_{\text{wrlo}} \subseteq \xrightarrow{*}_{\text{los}}$.

Now we make precise the equivalence of the two semantics when we start with the empty `l` and only use the emptiness tests:

Lemma 8.4. *Assume S is a UCST[Z] system. Let $C_{\text{in}} = (p,q,u,\varepsilon)$ be a configuration (where* `l` *is empty). Then, for any C_{fi} configuration, $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ iff $C_{\text{in}} \xrightarrow{*}_{\text{wrlo}} C_{\text{fi}}$.*

Proof. The "$\Leftarrow$" direction is trivial. For the "$\Rightarrow$" direction we claim that

$$\text{if } C \to_{\text{wrlo}} C' \sqsupseteq_1 C'', \text{ then also } C \sqsupseteq D \to_{\text{wrlo}} C'' \text{ for some } D. \tag{†}$$

Indeed, if (the occurrence of) the message in C' that is missing in C'' occurs in C, then it is possible to first lose this message, leading to D, before mimicking the step that went from C to C' (we rely here on the fact that S only uses Z tests). Otherwise, C'' is obtained by losing the message that has just been (reliably) written when moving from C to C', and taking $D = C$ is possible.

Now, since $\xrightarrow{*}_{\text{los}}$ is $\left(\to_{\text{wrlo}} \cup \sqsupseteq_1\right)^*$ and since $\left(\sqsupseteq_1\right)^*$ is $\sqsupseteq$, we can use (†) and conclude that $C \xrightarrow{*}_{\text{los}} D$ implies that $C \sqsupseteq C' \xrightarrow{*}_{\text{wrlo}} D$ for some C'. Finally, in the case where $C = C_{\text{in}}$ and `l` is empty, only $C' = C_{\text{in}}$ is possible. □

Corollary 8.5. *E-G-Reachability is decidable for UCST[Z] with write-lossy semantics.*

The write-lossy semantics is meaningful when modeling unreliability of the writing actions as opposed to unreliability of the channels. In the literature, write-lossy semantics is mostly used as a way of restricting the nondeterminism of message losses without losing any essential generality, relying on equivalences like Lemma 8.4 (see, e.g., [18, section 5.1]).

However, for our UCST systems, the write-lossy and the standard lossy semantics do not coincide when N tests are allowed. In fact, Theorem 4.1 does not extend to write-lossy systems.

Theorem 8.6. *E-E-Reach is undecidable for UCST[Z_1^1, N_1^1] with write-lossy semantics.*

Proof Idea. As in Section 3.2, Sender simulates a queue automaton using tests and the help of Receiver. See Fig. 9. Channel `l` is initially empty. To read, say, a from `r`, Sender does the following: (1) write a on `l`; (2) check that `l` is nonempty (hence the write was not lost); (3) check that, i.e., wait until, `l` is empty. Meanwhile, Receiver reads identical letters from `r` and `l`. □

Thus, at least in the write-lossy setting, we can separate UCST[Z] and UCST[Z_1^1, N_1^1] w.r.t. decidability of reachability.

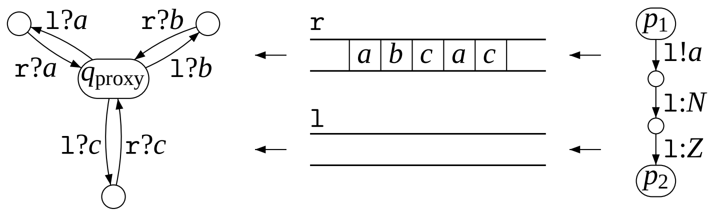

Figure 9: Write-lossy Sender simulates "$p_1 \xrightarrow{\mathtt{r}?a} p_2$" with N and Z tests and proxy Receiver

9. Conclusion

UCSes are communicating systems where a Sender can send messages to a Receiver via one reliable and one unreliable, lossy, channel, but where no direct communication is possible in the other direction. We introduced UCSTs, an extension of UCSes where steps can be guarded by tests, i.e., regular predicates on channel contents. This extension introduces limited but real possibilities for synchronization between Sender and Receiver. For example, Sender (or Receiver) may use tests to detect whether the other agent has read (or written) some message. As a consequence, adding tests leads to undecidable reachability problems in general. Our main result is that reachability remains decidable when only emptiness and non-emptiness tests are allowed. The proof goes through a series of reductions from UCST$[Z,N]$ to UCST$[Z_1^{\mathtt{l}}]$ and finally to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, an extension of Post's Embedding Problem that was motivated by the present article and whose decidability was recently proved by the last two authors [29].

These partial results do not yet provide a clear picture of what tests on channel contents make reachability undecidable for UCSTs. At the time of this writing, the two most pressing questions we would like to see answered are:

(1) what about occurrence and non-occurrence tests, defined as $\{O_a, NO_a \mid a \in \mathtt{M}\}$ with $O_a = \mathtt{M}^*.a.\mathtt{M}^*$ and $NO_a = (\mathtt{M} \smallsetminus \{a\})^*$? Such tests generalize N and Z tests and have been considered for channel systems used as a tool for questions on Metric Temporal Logic [12].
(2) what about UCSTs with tests restricted to the lossy $\mathtt{l}$ channel? The undecidable reachability questions in Theorem 3.1 all rely on tests on the reliable $\mathtt{r}$ channel.

References

[1] P. A. Abdulla, N. Bertrand, A. Rabinovich, and Ph Schnoebelen. Verification of probabilistic systems with faulty communication. *Information and Computation*, 202(2):141–165, 2005.
[2] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.
[3] P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.
[4] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
[5] M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *Proc. CONCUR 2008*, volume 5201 of *Lecture Notes in Computer Science*, pages 356–371. Springer, 2008.
[6] C. Baier, N. Bertrand, and Ph. Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Trans. Computational Logic*, 9(1), 2007.
[7] P. Barceló, D. Figueira, and L. Libkin. Graph logics with rational relations. *Logical Methods in Comp. Science*, 9(3), 2013.
[8] N. Bertrand and Ph. Schnoebelen. Computable fixpoints in well-structured symbolic model checking. *Formal Methods in System Design*, 43(2):233–267, 2013.

[9] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Formal Methods in System Design*, 14(3):237–255, 1999.

[10] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theor. Comp. Sci.*, 221(1–2):211–250, 1999.

[11] P. Bouyer, N. Markey, J. Ouaknine, Ph. Schnoebelen, and J. Worrell. On termination and invariance for faulty channel machines. *Formal Aspects of Computing*, 24(4–6):595–607, 2012.

[12] P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. The cost of punctuality. In *Proc. LICS 2007*, pages 109–120. IEEE Comp. Soc. Press, 2007.

[13] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[14] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[15] P. Chambart and Ph. Schnoebelen. Post Embedding Problem is not primitive recursive, with applications to channel systems. In *Proc. FST&TCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

[16] P. Chambart and Ph. Schnoebelen. Mixing lossy and perfect fifo channels. In *Proc. CONCUR 2008*, volume 5201 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

[17] P. Chambart and Ph. Schnoebelen. The ω-Regular Post Embedding Problem. In *Proc. FOSSACS 2008*, volume 4962 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2008.

[18] P. Chambart and Ph. Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216. IEEE Comp. Soc. Press, 2008.

[19] P. Chambart and Ph. Schnoebelen. Pumping and counting on the regular Post embedding problem. In *Proc. ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2010.

[20] L. Clemente, F. Herbreteau, A. Stainer, and G. Sutre. Reachability of communicating timed processes. In *Proc. FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013.

[21] L. Clemente, F. Herbreteau, and G. Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *Proc. CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 281–296. Springer, 2014.

[22] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comp. Sci.*, 256(1–2):63–92, 2001.

[23] A. Heußner, T. Le Gall, and G. Sutre. Safety verification of communicating one-counter machines. In *Proc. FST&TCS 2012*, Leibniz International Proceedings in Informatics, pages 224–235. Leibniz-Zentrum für Informatik, 2012.

[24] A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. *Logical Methods in Comp. Science*, 8(3), 2012.

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

[26] O. H. Ibarra, Zhe Dang, and P. San Pietro. Verification in loosely synchronous queue-connected discrete timed automata. *Theoretical Computer Science*, 290(3):1713–1735, 2003.

[27] P. Jančar, P. Karandikar, and Ph. Schnoebelen. Unidirectional channel systems can be tested. In *Proc. IFIP TCS 2012*, volume 7604 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2012.

[28] P. Karandikar and S. Schmitz. The parametric ordinal-recursive complexity of Post embedding problems. In *Proc. FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2013.

[29] P. Karandikar and Ph. Schnoebelen. Cutting through regular Post embedding problems. In *Proc. CSR 2012*, volume 7353 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2012.

[30] B. Konev, R. Kontchakov, F. Wolter, and M. Zakharyaschev. Dynamic topological logics over spaces with continuous functions. In *Advances in Modal Logic, vol.6*, pages 299–318. College Publications, 2006.

[31] A. Kurucz. Combining modal logics. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logics*, volume 3, chapter 15, pages 869–926. Elsevier Science, 2006.

[32] S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proc. TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2008.

[33] S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Computational Logic*, 9(2), 2008.

[34] R. Lazić, J. Ouaknine, and J. Worrell. Zeno, Hercules and the Hydra: Downward rational termination is Ackermannian. In *Proc. MFCS 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2013.

[35] A. Muscholl. Analysis of communicating automata. In *Proc. LATA 2010*, volume 6031 of *Lecture Notes in Computer Science*, pages 50–57. Springer, 2010.

[36] J. Ouaknine and J. Worrell. On metric temporal logic and faulty Turing machines. In *Proc. FOSSACS 2006*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[37] L. E. Rosier and Hsu-Chun Yen. Boundedness, empty channel detection, and synchronization for communicating finite automata. *Theoretical Computer Science*, 44(1):69–105, 1986.

[38] S. Schmitz. Complexity hierarchies beyond elementary. Preprint arXiv:1312.5686 [cs.CC], December 2013.

[39] S. Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *Proc. ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2011.

[40] S. Schmitz and Ph. Schnoebelen. The power of well-structured systems. In *Proc. CONCUR 2013*, volume 8052 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2013.