On Post's embedding problem and the complexity of lossy channels

Pierre Chambart

EDSP
École Doctorale
Sciences Pratiques

ENSC-2011 n°303

ENS CACHAN

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Monsieur Pierre Chambart

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine :
Informatique

Sujet de la thèse :

Du problème de sous mot de Post et de la complexité des canaux non fiables

Thèse présentée et soutenue à Cachan le 29/09/2011 devant le jury composé de :

Ahmed Bouajjani	Professeur	Président, rapporteur
Joel Ouaknine	Professeur	Rapporteur
Olivier Serre	Chargé de recherches	Examinateur
Grégoire Sutre	Chargé de recherches	Examinatrice
Philippe Schnoebelen	Directeur de recherches	Directrice de thèse

Nom du Laboratoire LSV
ENS CACHAN/CNRS/UMR 8643
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

On Post's Embedding Problem and the complexity of lossy channels

Pierre Chambart

29/09/2011

Chapter 1

Introduction

1.1 The paradox of lossy communications

Channel Systems (CS), also called *Finite-State communicating Machines*, are systems of finite-state automata that communicate via unbounded FIFO channels. *Lossy Channel Systems* (LCS) are a variant model permitting more behaviours than Channel Systems. They also allow to directly model protocols assuming communication unreliability.

CS are long known to be equivalent to Turing Machines. It was a real surprise when Abdulla and Jonsson [AJ93] showed that LCS analysis were easier, by proving that reachability, safety and inevitability problems were decidable. Simultaneously Finkel [Fin94] showed the decidability of termination. The right propreties allowing decidability were summarised in Well Structured Transition Systems. They were introduced independently by both Abdulla and Finkel. Meanwhile these algorithms also started to be implemented in tools [ABJ98]. Destipe the accessibility of reachability and termination, LCS are not a trivial model : indeed some problems on LCS are undecidable, such as liveness, finiteness and bisimulation [AJ96a],[AK95]. Once the most natural decidability questions were answered, studies started to focus on probabilistic versions and efficient algorithms.

1.2 The mystery of complexity

In the 90s, many decidability questions on LCS were solved, but no progress was made on the question of assessing the complexity of the decidability problems. It was argued that the non-constructiveness of termination proofs, based on well quasi ordering theory, could not bring any bounds. As stated by Abdulla and Jonsson in [AJ93] "The analysis is difficult since in general there is no bound on the length of sequences in Higman's theorem". But this was not a hindrance to development of tools that could even implement non-terminating algorithms like [ACBJ04]. Nobody knew the order of magnitude of the complexity before the first step by Schnoebelen [Sch02] on the nonprimitive recursive lower bounds. Indeed, using the right encoding, LCS seemed to be able to subsume every classical complexity class. It seems that the problem was more with guessing the right complexity class than proving LCS would live there. The right class came from the world of proof theory.

More precisely, we show below that verifying LCS is exactly at level $\mathfrak{F}_{\omega^\omega}$ of the Extended Grzegorczyk Hierarchy. This hierarchy is very rarely visited in the verification community. We can still mention a few examples : Petri net equivalences [MM81, McA84, Clo86, Jan01] and upper bounds on the size of Karp-Miller trees [Mül85].

Mayr and Meyer [MM81] used some unusual technique, relating their problem to a bounded version of Hilbert's 10th problem. Jancar gave a simpler version of this result, based on a direct simulation of Turing Machines

in space bounded by the Ackermann function in [Jan01], and applied it to different Petri net problems. The proof on LCS was inspired by this work. Accessibility on LCS was the second problem in verification known to be decidable but nonprimitive recursive.

These more classical reductions made those results more accessible to the verification community, which found in it the base for more lower bound results on many other models such as metric temporal logic [OW07], alternating one-clock timed automata [ADO+08, LW08], leftist grammars [Jur08, CS10], products of modal logics [GKWZ06], data nets [LNO+08], weak memory models [ABBM10]. We suppose that those results are reductions from LCS rather than from Petri nets because there is a broader choice of source problem on LCS. Indeed almost any non trivial problem is nonprimitive recursive on LCS whereas only some problems are on Petri nets. This, de facto, led to the reachability problem on LCS (ReachLcs) becoming a central problem of its own complexity class.

This was the state of the art when we started to work on this problem. Our results improve the knowledge on the complexity of ReachLcs in two directions.

Firstly, we explored proof theory literature searching for results indicating how far we could push the lower bound. We found an article from Cichon and Tahhan Bittar[CT98] giving limits on length of sequences obtained by Higman's lemma. With this result, we could show that the algorithm solving ReachLcs is in a class of functions called $\mathfrak{F}_{\omega^\omega}$. Then, we showed that this could not be solved in any smaller class by building sequences following LCS behaviours that could reach the upper bound. Here, we define a classical complexity class $\mathsf{F}_{\omega^\omega}$, closely linked to $\mathfrak{F}_{\omega^\omega}$, such that ReachLcs is $\mathsf{F}_{\omega^\omega}$-complete.

The second evolution was to develop the regular Post Embedding Problem ($\mathsf{PEP}^{\mathrm{reg}}$) as another base $\mathsf{F}_{\omega^\omega}$-complete problem. Its definition is much simpler and its manipulation requires less coding artifacts than ReachLcs. We think that $\mathsf{PEP}^{\mathrm{reg}}$ is, in many cases, better suited as a base problem for the class $\mathsf{F}_{\omega^\omega}$. It could be used the same way the Post Correspondence Problem (PCP) is for undecidable problems.

1.3 Complementary notes

1.3.1 The way it happened

$\mathsf{PEP}^{\mathrm{reg}}$ did not came out of the blue, we were not looking for those results when I started this thesis. I first studied the limit between decidability and undecidability on systems allowing both reliable and unreliable communications, as presented in chapter 5. We first looked at the Post's Correspondence Problem with equality replaced by embedding, hoping that the PCP community already proved its undecidability, to show that some base case

was undecidable. It appeared that nobody ever looked at it. We still chose to continue and defined $\mathsf{PEP}^{\mathrm{reg}}$ since its manipulation was easier than our channel systems. It later appeared to be decidable and equivalent to ReachLcs. We realized that $\mathsf{PEP}^{\mathrm{reg}}$ was a promising candidate, as a base problem, and decided to investigate it further.

1.3.2 More related work

The study of the complexity of Higman's Lemma was initiated by de Jongh and Parikh [dJP77] who measured the maximum order-type compatible with the subword ordering. Constructive proofs of Higman's Lemma provide recursive upper bounds that are inherited from the computational power of the underlying logical framework, and are thus exaggeratedly high. Using clever combinatorial reasoning, Cichon and Tahhan Bittar [CT98] were the firsts to provide tight upper bounds for the length of bad sequences with relation to the subword ordering. An earlier $\mathfrak{F}_\omega$ upper bound for bad sequences in $\mathbb{N}^k$ (Dickson's Lemma) was provided by McAloon [McA84]. From a proof theory point of view, the part of our results on lower bound of ReachLcs, where it is shown that LCS can compute the F_{ω^ω} function, can be seen as a characterization of multiply recursive functions with Higman's lemma. Such a work was already done by Touzet [Tou02] using different rewriting systems.

On the practical side of LCS, it quickly appeared that the backwards algorithms were not ideal. In practical cases, the forward algorithms, even if they have no termination guaranty [FG09a], seem to be far more efficient [ABJ98] [ACBJ04]. In fact, given such a high complexity, the practical difference between terminating and non terminating algorithm does not really matter. But in the case of verification of human made protocols, the forward analysis seams to better match the way they were designed. Moreover it also gives liveness information. The TReX tool was designed on this principle [ABS01].

Another approach to tackle the limitations of the liveness analysis taken by Bertrand and Schnoebelen et al. is to consider probabilistic loss rather than non-deterministic [BS03],[ABRS05],[BBS07].

Atig and Bouajjani also studied systems connected by lossy channels. They looked at more powerful systems, pushdown ones, but no reliable connections [AB09].

The first definitions of Well Structured Transition System (WSTS) comes from Finkel [Fin87], it was first inspired by Petri nets. Then it grew well and what seems to be now a stable definition and set of classical results can be found in [FS01]. In recent years the studies followed the same path as LCS and the results on forward analysis were generalized to WSTS on [FG09a] and [FG09b].

1.4 Short summary

In chapter 3 we prove the aforementioned lower and upper bounds on ReachLcs. Chapter 4 is dedicated to the equivalence between $\mathsf{PEP}^{\mathrm{reg}}$ and ReachLcs. Chapter 5 describes our results on mixing lossy and reliable channels.

The second part focuses on $\mathsf{PEP}^{\mathrm{reg}}$ and its variants. Chapter 6 gives variants justifying our definition of $\mathsf{PEP}^{\mathrm{reg}}$ and some infinitary extensions. Chapter 7 gives a direct decidability proof of $\mathsf{PEP}^{\mathrm{reg}}$. Chapter 8 studies what we call blockers PEP languages, technical elements introduced to prove the decidability of $\mathsf{PEP}^{\mathrm{reg}}$, which turned out to have interesting composability properties and $\mathsf{F}_{\omega^\omega}$-complete problems. Chapter 9 will show results on languages of PEP-solutions.

Contents

Chapter 2

Preliminary

2.1 Words, languages and subword ordering

We write $x, y, w, t, \sigma, \rho, \alpha, \beta, \ldots$ for words, i.e., finite sequences of letters such as $a, b, i, j, \ldots$ from alphabets $\Sigma, \Gamma, \ldots$. With $x.y$, or xy, we denote the concatenation of x and y. With ϵ we denote the empty word. The *length* of x is written $|x|$.. A language $L \subset \Sigma^*$ is a set of words. The mirror image of a word x is denoted $\widetilde{x}$, e.g., $\widetilde{abc} = bca$. The mirror image of a language L is $\widetilde{L} \stackrel{\text{def}}{=} \{\widetilde{x} \mid x \in L\}$.

2.1.1 Word morphisms

A *morphism* from Σ^* to Γ^* is a map $u : \Sigma^* \to \Gamma^*$ that respects the monoidal structure, i.e., with $u(\epsilon) = \epsilon$ and $u(x.y) = u(x).u(y)$. A morphism u is completely defined by its image $u(a)$, $u(b)$, ..., on $\Sigma = \{a, b, \ldots\}$. Most of the time, we shall write $u_a, u_b, \ldots$, and u_x, instead of $u(a), u(b), \ldots$, and $u(x)$.

2.1.2 Syntactic congruence.

For a language L, we let $\sim_L$ denote the syntactic congruence induced by L: $x \sim_L y \stackrel{\text{def}}{\Leftrightarrow} \forall w, w'(wxw' \in L \Leftrightarrow wyw' \in L)$. The Myhill-Nerode Theorem states that $\sim_L$ has finite index iff L is a regular language. For a regular L, we let n_L denote the number of equivalence classes w.r.t. $\sim_L$.[1]

2.1.3 Subword ordering

Given two words x and y, we write $x \sqsubseteq y$ when x is a *subword* of y, i.e., when x can be obtained by erasing some letters (possibly none) from y. For example, $abba \sqsubseteq \underline{ab}racada\underline{b}r\underline{a}$. The subword relation, aka *embedding*, is a partial ordering on words. It is compatible with the monoidal structure:

$$\epsilon \sqsubseteq x, \qquad (x \sqsubseteq y \wedge x' \sqsubseteq y') \Rightarrow xx \sqsubseteq yy'.$$

2.1.4 Well quasi ordering.

A *well quasi ordering* (wqo) is a quasi ordering $(S, \leq)$ such that for any infinite sequence $s_0 s_1 s_2 \cdots$ of S^ω there exist $i < j$ in $\mathbb{N}$ such that $s_i \leq s_j$. Equivalently, there does not exist any strictly descending chain $s_0 > s_1 > \cdots > s_i > \cdots$, and any *antichain*, i.e. set of pairwise incomparable elements, is finite. A *well partial order* (wpo) is an antisymmetric wqo

Remark 2.1.1 *If X is finite set $(X, =)$ is a wpo. $(\mathbb{N}, \leq)$ is a wpo. $(\mathbb{N}^k, \leq)$ the set of vectors of k natural numbers with component-wise ordering is a*

[1] If the minimal complete DFA that accepts L has q states, then n_L can be bounded by q^q.

wpo (Dickson's lemma), and more generaly, if $(X_1, \leq_1), \ldots, (X_k, \leq_k)$ *are wqos then* $(X_1 \times X_2 \times \ldots \times X_k, \leq_{1,\ldots,k})$ *the tuples of elements from* $X_1, \ldots, X_k$ *with component-wise ordering is a wqo.*

Bad sequences.

We say that a sequence $x_1, \ldots, x_l, \ldots$ of words in Σ^* is n-good if there exists indexes $i_1 < i_2 < \ldots < i_n$ such that $x_{i_1} \sqsubseteq x_{i_2} \sqsubseteq \ldots \sqsubseteq x_{i_n}$, i.e., if the sequence contains a subsequence of length n that is increasing w.r.t. embedding. It is n-bad otherwise. On wqo's every infinite sequence is 2-good, and even n-good for any $n \in \mathbb{N}$. Hence n-bad sequences are finite.

2.1.5 Higman's Lemma.

Lemma 2.1.2 (Higman's Lemma [Hig52]) *The subword ordering* $(\Sigma^*, \sqsubseteq)$ *is a well partial order if* Σ *is finite.*

Upward-closed and downward-closed languages.

A language $L \subseteq \Gamma^*$ is *upward-closed* if $x \in L$ and $x \sqsubseteq y$ imply $y \in L$. It is *downward-closed* if $x \in L$ and $y \sqsubseteq x$ imply $y \in L$ (equivalently, if its complement is upward-closed). Higman's Lemma entails that any antichain is finite, thus that any upward-closed set has a finite set of minimal elements. This directly implies that upward-closed and downward-closed languages are regular (See also [Hai69]). Upward-closed languages can naturally be denoted by very simple regular expressions. Downward-closed languages also have a convenient respresentation called simple regular expression [FG09a, ACBJ04]. We write $\uparrow w$ ($\uparrow L$) for the smallest upward-closed language containing a word w (language L) and $\downarrow w$ ($\downarrow L$) for downward-closed languages.

Simple regular expressions. $*$*-products* are concatenations of *atoms* that are either of the form $a + \epsilon$ for some $a \in \Gamma$, or of the form A^* for some sub-alphabet $A \subseteq \Gamma$. A *simple regular expressions* (SRE) is a finite union of $*$-products. For example, with $\Gamma = \{a, b, c\}$, the set of subwords of *abac* is $(a + \epsilon).(b + \epsilon).(a + \epsilon).(c + \epsilon)$ and the set of words that do not have *ab* as a subword is $\{b, c\}^*.\{a, c\}^*$.

Theorem 2.1.3 (Abdulla, Collomb-Annichini, Bouajjani, Jonsson) *The downward closed languages are the languages recognizable by SRE's*

Remark 2.1.4 *Another equivalent definition of well quasi ordering, is an ordering such that every increasing sequence of upward closed set* $(U_i)_{i\in\mathbb{N}}, U_i \subset U_{i+1}$ *eventually stabilize, i.e. there exists* j *such that* $\forall k \geq j, U_k = U_j$ *(See [Kru72]).*

Higman's Lemma on tuples of words. Higman's Lemma also holds on the component-wise extension $((\Sigma^*)^p, \sqsubseteq_p)$ of $\sqsubseteq$ to p-tuples of words. i.e. $(x_1, \ldots, x_p) \sqsubseteq_p (y_1, \ldots, y_p)$ if $x_1 \sqsubseteq y_1, \ldots x_p \sqsubseteq y_p$. Indeed, if we add a new letter $\#$ to the alphabet Σ, with $x_1, \ldots, x_p, y_1, \ldots, y_p \in \Sigma^*$, $x_1\#x_2\#\ldots\#x_p \sqsubseteq y_1\#y_2\#\ldots\#y_p$ iff $(x_1, \ldots, x_p) \sqsubseteq_p (y_1, \ldots, y_p)$.

From Now on, we will only use the notation $\sqsubseteq$ to denote both orders.

Miniaturisation

Higman's Lemma is often described as being "non-effective" in that it does not give any information on the length of bad sequences. Indeed, arbitrarily long bad sequences exist. However, upper bounds on the length of bad sequences can certainly be given when one restricts to "simple" sequences. Such finitary versions of well-quasi-ordering properties are called "miniaturisations" in proof-theoretical circles.

We will consider a very simple miniaturisation that applies to "controlled" sequences [CT98]. Formally, given $n \in \mathbb{N}$ and an increasing function $g : \mathbb{N} \longrightarrow \mathbb{N}$, we say that a sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ is *controlled by* (g, n) when for each i, $|\boldsymbol{x}_i| \leq g^i(n)$. For a p-tuples $\boldsymbol{x} = (x^1, \ldots, x^p)$, the size $|\boldsymbol{x}|$ is $\max_{1 \leq i \leq p}(x^i)$. In our setting, we will only use linear control functions g. We say that a sequence is *k-controlled* when it is controled by $(Succ^k, 0)$, i.e. $\boldsymbol{x}_i \leq i \times k$.

Lemma 2.1.5 *There exists a* bounding function $H : \mathbb{N}^4 \to \mathbb{N}$ *such that, for any $n, k, p \in \mathbb{N}$ and $l \geq H(n, k, p, |\Sigma|)$, any k-controlled sequence of p-tuples of words of length l in Σ^* is n-good.*

The lemma states that if a k-controlled sequence is long enough, it is n-good. Equivalently, n-bad sequences are shorter that $H(n, k, p, |\Sigma|)$ or are not k-controlled.

Proof. Fix $n > 0, k, p, \Sigma$ and consider the set B of all k-controlled n-bad finite sequences. Every subsequence of a bad sequence is bad hence B is prefix-closed and the sequences can be naturally arranged in a tree, with the empty sequence at its root. The tree is finitely branching because the sequences are k-controlled (and Σ is finite). If B is infinite, the tree has an infinite branch (Kőnig's Lemma), that is, there exists an infinite sequence $x_1, x_2, \ldots$ for which all finite prefixes are n-bad. Hence the infinite sequence itself is n-bad, which is impossible by Higman's Lemma. Finally, B must be finite and taking $H(n, k, p, |\Sigma|)$ as the length of the longest sequence in B will fulfill the requirements. □

$H(n, k, p, |\Sigma|)$ is our notation for what Cichon and Tahhan Bittar denote $Hig(\omega^{|\Sigma|}.p, n, Succ^k)(k)$.

2.2 Channel Systems

Channel Systems (CS) are systems of automata communication through unbounded FIFO channels [BZ83]. A CS is a tuple $S = (\mathtt{Q}, \mathtt{M}, \mathtt{C}, \Delta)$ where $\mathtt{Q} = \{q_1, q_2, \ldots\}$ is a finite set of *(control) state*, $\mathtt{M} = \{a_1, a_2, \ldots, a_k\}$ is a finite *message alphabet*, $\mathtt{C} = \{\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_l\}$ is a finite set of *channels*, and $\Delta \subseteq \mathtt{Q}\times\mathtt{C}\times\{!, ?\}\times\mathtt{M}\times\mathtt{Q}$ is a finite set of *transition rules*, with typical elements denoted δ. A rule of the form $(q, \mathtt{c}, !, a, q')$ (respectively, $(q, \mathtt{c}, ?, a, q')$) is called a *writing rule* (resp., a *reading rule*). Rules are often also denoted $q \xrightarrow{!_{\mathtt{c}}a} q'$ for a writing rule $(q, \mathtt{c}, !, a, q')$ and $q \xrightarrow{?_{\mathtt{c}}a} q'$ for a reading rule $(q, \mathtt{c}, ''\,, a, q')$.

Assume that $S = (\mathtt{Q}, \mathtt{M}, \mathtt{C}, \Delta)$ is a CS with l channels. A *configuration* of S is a pair $(q, \boldsymbol{u})$, where $q \in \mathtt{Q}$ is the current control state and $\boldsymbol{u} \in (\mathtt{M}^*)^l$, is the contents of the channels. $(q, \boldsymbol{u})$ is sometimes written $(q, u_1, \ldots, u_l)$ where $u_i \in \mathtt{M}^*$ is the sequence of messages contained in channel $\mathtt{c}_i$ (by convention, reading occurs at the head of u_i and writing at its tail). We write $\mathit{Conf} = \{\sigma, \rho, \ldots\}$ for the set $\mathtt{Q} \times (\mathtt{M}^*)^l$ of configurations (of S). Configurations of CSs are compared via the subword ordering:

$$(q, \boldsymbol{u}) \sqsubseteq (q', \boldsymbol{u}') \stackrel{\text{def}}{\Leftrightarrow} q = q' \wedge \boldsymbol{u} \sqsubseteq \boldsymbol{u}'.$$

Observe that, since $\mathtt{Q}$ and $\mathtt{M}$ are finite, $(\mathit{Conf}, \sqsubseteq)$ is a well partial order as a consequence of Higman's Lemma and remark 2.1.1.

On this basis, we will define two kinds of channel systems, *reliable* and *lossy* ones. The only difference is the operational semantics associated with the system.

2.2.1 Perfect Channel Systems

The operational semantics of *reliable* or *perfect* S is given under the form of a transition system $\mathcal{T}_S^{\text{perf}} = (\mathit{Conf}, \rightarrow_{\text{perf}})$. Assume that $\sigma = (q, u_1, \ldots, u_l)$ and $\sigma' = (q', u'_1, \ldots, u'_l)$ are two configurations. There is a step from σ to σ' via rule δ, denoted $\sigma \xrightarrow{\delta}_{\text{perf}} \sigma'$, when:

— **case 1**: $\delta \in \Delta$ is a reading rule of the form $(q, \mathtt{c}_i, ?, a, q')$ and $u_i = a u'_i$ while $u_j = u'_j$ for $j \neq i$, or
— **case 2**: δ is a writing rule $(q, \mathtt{c}_i, !, a, q')$ and $u'_i = u_i a$ while $u_j = u'_j$ for $j \neq i$

2.2.2 Lossy Channel Systems

A *lossy channel systems* (LCS) is a channel system with an extended operational semantics. Several different notions of message losses were proposed in the literature. We choose to present two of them, the standard semantics $\mathcal{T}_S^{\text{std}} = (\mathit{Conf}, \rightarrow_{\text{sl}})$ [AJ93] and the *write-lossy* semantics $\mathcal{T}_S^{\text{wl}} =$

$(Conf, \rightarrow_{\mathrm{wl}})$. As we focus on problems where the choice between those semantics doesn't matter, we will chose one or the other when it is more convenient. We will usualy use the standard semantics, but in chapter 3 and 5 we use the write-lossy one.

The standard semantics, $\mathcal{T}_S^{\mathrm{std}}$, assumes that any messages can be lost before and after any perfect step. That is, it puts

$$\rightarrow_{\mathrm{sl}} \stackrel{\mathrm{def}}{=} \sqsupseteq \circ \rightarrow_{\mathrm{perf}} \circ \sqsupseteq \tag{2.1}$$

The write-lossy semantics, $\mathcal{T}_S^{\mathrm{wl}}$, only allow to loose message that were just written. Formaly assume that $\sigma = (q, u_1, \ldots, u_l)$ and $\sigma' = (q', u'_1, \ldots, u'_l)$ are two configurations. There is a step from σ to σ' via rule δ, denoted $\sigma \xrightarrow{\delta}_{\mathrm{wl}} \sigma'$, when there is such a step in perfect channel system semantics as above, i.e. $\sigma \xrightarrow{\delta}_{\mathrm{perf}} \sigma'$, or when:

— **case 3**: δ is a writing rule $(q, \mathsf{c}_i, !, a, q')$ and $u_j = u'_j$ for all $j = 1, ..., l$.

Hence a message can be lost during a step that attempts to write it in the channels. Once in the channels, messages cannot be lost, they can only be removed by reading steps.

2.2.3 Notations

When writing steps, we usually omit the δ superscript when it is not useful. We also often don't specify the semantics, when it is unambiguous from the context. We use the standard notations "$\xrightarrow{n}$", "$\xrightarrow{+}$" and "$\xrightarrow{*}$" for, respectively, the n-fold composition, the transitive closure and the reflexive-transitive closure of a transition relation "$\rightarrow$". When it is ambiguous, we can write "$\rightarrow_S$" to specify the system from which the transition belong to.

When there is a writing rule $\delta = (q, \mathsf{c}_i, !, a, q')$ such that $\sigma \xrightarrow{\delta} \sigma'$, we also write $\sigma \xrightarrow{!_{\mathsf{c}}a} \sigma'$, and $\sigma \xrightarrow{?_{\mathsf{c}}a} \sigma'$ for a reading rule $\delta = (q, \mathsf{c}_i, ?, a, q')$

For clarity reasons, when $w = a_1, \ldots, a_n$ is a word, we often write $\sigma \xrightarrow{!_{\mathsf{c}}w} \sigma'$ when we want to say that there are configurations $\sigma_1, \ldots, \sigma_{n-1}$ such that $\sigma \xrightarrow{!_{\mathsf{c}}a_1} \sigma_1 \xrightarrow{!_{\mathsf{c}}a_2} \sigma_2 \ldots \sigma_{n-1} \xrightarrow{!_{\mathsf{c}}a_n} \sigma'$.

We also use this notation to denote rules writing or reading words. When the system has only one channel, we usually don't specify the channel name, denoting rules as $q \xrightarrow{!a} q'$.

2.2.4 Compatibility

Definition 2.2.1 *An ordering $\leq$ is* compatible *with a transition system $\mathcal{T}_S = (Conf, \rightarrow)$ $\stackrel{def}{\Leftrightarrow}$ if there are σ, σ', δ and $\rho \in Conf$ such that $\sigma \sqsubseteq \sigma'$ and and $\sigma \xrightarrow{\delta} \rho$, then there is ρ' such that $\rho \leq \rho'$ and $\sigma' \xrightarrow{\delta} \rho'$.*

Lemma 2.2.2 *$\sqsubseteq$ is compatible with $\mathcal{T}_S^{\mathrm{std}}$*

From a configuration σ' bigger than σ, it is always possible to lose some message during $\rightarrow_{\mathrm{sl}}$ to simulate any transition that can be done from σ. Indeed $(\sqsupseteq \circ \sqsupseteq \circ \rightarrow_{\mathrm{perf}} \circ \sqsupseteq) = (\sqsupseteq \circ \rightarrow_{\mathrm{perf}} \circ \sqsupseteq) = \rightarrow_{\mathrm{sl}}$

Compatibility and $\sqsubseteq$ being a wqo are the key properties for $\mathcal{T}_S^{\mathrm{std}}$ to be Well Structured Transition Systems, giving many decision algorithms, but $\sqsubseteq$ is not compatible with $\mathcal{T}_S^{\mathrm{wl}}$. However the write-lossy semantics is close enouth for their difference not being an hindrance.

Lemma 2.2.3 *Assume σ has the form $(q, \epsilon, \ldots, \epsilon)$. Then*
1. σ' is reachable from σ in $\mathcal{T}_S^{\mathrm{wl}}$ iff it is reachable from σ in $\mathcal{T}_S^{\mathrm{std}}$, and
2. there is an infinite run from σ in $\mathcal{T}_S^{\mathrm{wl}}$ iff there is one in $\mathcal{T}_S^{\mathrm{std}}$.

Thanks to this lemma, our results will apply on both semantics.

Lemma 2.2.4 *For all $n > 0$, $\xrightarrow{n}_{\mathrm{sl}} = \xrightarrow{n}_{\mathrm{wl}} \circ \sqsupseteq$.*

Proof. Note that the only difference between $\rightarrow_{\mathrm{sl}}$ and $\rightarrow_{\mathrm{perf}} \circ \sqsupseteq$ is that $\rightarrow_{\mathrm{sl}}$ can lose a message that has just been written by the $\rightarrow_{\mathrm{perf}}$ part in (2.1). Since this can be done by write-lossy steps $\rightarrow_{\mathrm{wl}}$, $\rightarrow_{\mathrm{sl}}$ and $\rightarrow_{\mathrm{wl}} \circ \sqsupseteq$ coincide.

By induction on n. As we just observed, the base case $n = 1$ holds. For the inductive step, we use

$$\begin{aligned}
\xrightarrow{n+1}_{\mathrm{sl}} = \xrightarrow{n}_{\mathrm{sl}} \circ \rightarrow_{\mathrm{sl}} &= \xrightarrow{n}_{\mathrm{wl}} \circ \sqsupseteq \circ \rightarrow_{\mathrm{sl}} && \text{by ind. hyp.} \\
&= \xrightarrow{n}_{\mathrm{wl}} \circ \rightarrow_{\mathrm{sl}} && \text{using (2.1)} \\
&= \xrightarrow{n}_{\mathrm{wl}} \circ \rightarrow_{\mathrm{wl}} \circ \sqsupseteq && \text{using case ``}n = 1\text{''} \\
&= \xrightarrow{n+1}_{\mathrm{wl}} \circ \sqsupseteq\ .
\end{aligned}$$

□

Proof.[lemma 2.2.3] Since $\rightarrow_{\mathrm{wl}} \subseteq \rightarrow_{\mathrm{sl}}$, we only have to prove the "$\Leftarrow$" implications.
1. Since $\sigma \xrightarrow{*}_{\mathrm{sl}} \sigma'$ then from Lemma 2.2.4: there is ρ, $\sigma \sqsupseteq \rho \xrightarrow{*}_{\mathrm{wl}} \sigma'$. When σ has empty channels, $\rho \sqsubseteq \sigma$ requires $\rho = \sigma$.
2. The sets of runs of $\mathcal{T}_S^{\mathrm{std}}$ and $\mathcal{T}_S^{\mathrm{wl}}$ starting from σ can be arranged in trees with the length 0 run at their roots. They are finitely branching, hence, using König's Lemma, they are infinite iff they have an infinite branch. A consequence of 1. is that $\forall n, \exists m, \exists \sigma', m \geq n, \sigma \xrightarrow{m}_{\mathrm{sl}} \sigma' \iff \forall n, \exists m, \exists \sigma', m \geq n, \sigma \xrightarrow{m}_{\mathrm{wl}} \sigma'$, which complete the proof. □

Therefore, when the initial configuration has empty channels, a LCS satisfy exactly the same reachability and termination properties under the standard semantics, or under the write-lossy semantics. In particular, exactly the same algorithms can be used.

Remark 2.2.5 *In the general case where the initial configuration is not necessarily empty, it is easy to reduce reachability and termination from one semantics to the other: one simply encodes the initial channel contents (and its residuals) in the control states, and adds transition rules for these extra states, encoding the original semantics.*

Problems on LCS

The two problems we will consider on LCSs will be reachability and termination. Reachability is the historical decidability result. It was also the first problem on LCS to be proved nonprimitive recursive and was reduced many times to show hardness on other problems. Termination was the first problem on LCS solved by a different kind of algorithm than reachability. Those problems exhibits the two kinds of algorithms existing on LCSs. Reachability is solved by backward exploration, i.e. computing $Pre^*(s)$, the set of configuration reaching a set s. Termination use forward search, i.e. computing the set of bad sequences of configurations.

Reachability

Theorem 2.2.6 (Abdulla, Jonsson [AJ93]) *Let U be an upward closed set of configurations of an LCS S, $Pre^*(U)$ is computable.*

Proof.[Sketch] The main ideas are that
— **1**: the set $Pre(U)$ of predecessor in one step of an upward closed set of configuration U is computable and is upward closed.
— **2**: the sequence $(Pre^i(U))_{i\in\mathbb{N}}$ of sets of configuration reaching an upward closed set U in at most i steps is an increasing sequence of upward closed sets, then it stabilizes (Remark 2.1.4). □

LCS reachability problem, ReachLcs

Instance: An LCS S and two configurations σ ρ of S.

Question: Does $\sigma \xrightarrow{*}_S \rho$?

Theorem 2.2.7 (Abdulla, Jonsson [AJ93]) ReachLcs *is decidable.*

Proof. By definition of Pre and $\uparrow$, $\sigma \in Pre^*(\uparrow \rho)$ iff. $\exists \rho' \sqsupseteq \rho, \sigma \xrightarrow{*} \rho'$. On standard semantics $\exists \rho' \sqsupseteq \rho, \sigma \xrightarrow{*}_{\mathrm{sl}} \rho' \iff \sigma \xrightarrow{*}_{\mathrm{sl}} \rho'$. Pre^* being computable (Lemma 2.2.6), reachability is decidable on $\mathcal{T}_S^{\mathrm{std}} = (Conf, \rightarrow_{\mathrm{sl}})$.

Thanks to Lemma 2.2.3 and Remark 2.2.5 this also holds on write-lossy semantics. □

Termination

Theorem 2.2.8 *Let σ be a configuration of an LCS S, the set of bad run (runs that are bad sequences) starting from σ is finite and computable.*

Proof.[Sketch] Runs are controlled, indeed a transition rule can only add one letter to a channel. It then suffice to remember that controlled bad sequence have a length bounded by $H(|Q|, 1, |C|, |\mathtt{M}|)$ and that H is computable (Lemma 2.1.5) to conclude that we can exhaustively search bad sequences. □

LCS termination problem

Instance: An LCS S and a configuration σ from S.

Question: Are all runs of S starting from σ finite ?

Theorem 2.2.9 (Finkel [Fin94]) *Termination is decidable.*

Proof. With standard semantics, all runs from a terminating LCS are bad. If that were not the case, there would be an good run, i.e. some $\sigma_0 \xrightarrow{*}_{\mathrm{sl}} \sigma_i \xrightarrow{\delta_1}_{\mathrm{sl}} \cdots \xrightarrow{\delta_n}_{\mathrm{sl}} \sigma_{i+n}$ with $\sigma_i \sqsubseteq \sigma_{i+n}$. Then, $\sqsubseteq$ compatibility tells us that the sequence of transition $\delta_1, \ldots, \delta_n$ could be fired from σ_{i+n} leading to a configuration greater than σ_{i+n}, and that could be repeated indefinitely, giving an infinite run.

The set of bad sequences being finite and computable (Theorem 2.2.8), it is possible to check that there is no other runs, i.e. good runs, which conclude the proof.

As for reachability, here also, thanks to Lemma 2.2.3 and Remark 2.2.5 termination is decidable on write-lossy semantics. □

One channel suffice

A classic assumption is to restrict to LCSs with only one channel. In fact, systems with multiples channels can always be encoded in systems with only one channel and an alphabet extended with a separation message #. A configuration $(q, u_1, u_2, \ldots, u_n)$ is encoded to $(q, u_1 \# u_2 \# \ldots \# u_n)$. The loss of # characters can easily be detected by the structure of the system. Although this encoding permit to consider decidability question only on one channel LCSs, for complexity questions we can't. Indeed as shown in chapter 3, the key factor to complexity is the size of the alphabet, not the number of channels.

2.3 F_{ω^ω}, $\mathfrak{F}_{\omega^\omega}$ and $\mathsf{F}_{\omega^\omega}$ hierarchies

We will now introduce these three different but related notions. The fast growing hierarchy, which is an ordinal-indexed family of rapidly increasing functions $F_\alpha : \mathbb{N} \longrightarrow \mathbb{N}$; $\mathfrak{F}_\alpha$ the class of functions "elementary" in F_α and F_α the complexity class of problems in time or space bounded by F_α and closed by primitive recursive reductions.

2.3.1 Primitive recursive and multiply recursive functions

The *primitive recursive functions* are the integer functions definable using only:

- for every n the n-ary constant function: $0_n : \mathbb{N}^n \longrightarrow \mathbb{N}$.
- the 1-ary successor and predecessor functions: $Succ, Pred : \mathbb{N} \longrightarrow \mathbb{N}$.
- for every $n \geq 1$, for each i with $1 \leq i \leq n$ the n-ary projection P_n^i which returns the i-th component, i.e. $P_n^i(a_1, \ldots, a_n) = a_i$.
- composition of primitive recursive functions.
- primitive recursion.

The *primitive recursion* being a restricted recursion such that, for primitive recursive functions f, g and k respectively k, $k+2$ and 1-ary, the function h is defined by primitive recursion from f, g and p, if p is decreasing and

$$h(n, x_1, \ldots, x_k) = \begin{cases} f(x_1, \ldots, x_k) & \text{if } n = 0 \\ g(p(y), h(p(y), x_1, \ldots, x_k), x_1, \ldots, x_k) & \text{if } n > 0. \end{cases}$$

For instance, addition, Add is primitive recursive and can be defined with $f = P_2^2$ and $g(x, y, z) = Succ(P_3^2(x, y, z))$ which is more clearly stated as

$$Add(n, x) = \begin{cases} P_2^2(n, x) & \text{if } n = 0 \\ Succ(P_3^2(Pred(n), Add(Pred(n), x), x)) & \text{if } n > 0. \end{cases}$$

The important fact is that the Ackermann function, usually defined by

$$Ack(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ Ack(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ Ack(m - 1, Ack(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

is not primitive recursive[2]. This function is a diagonalization of the class of primitive recursive function, i.e. each $Ack_m(n) = Ack(m, n)$ is definable using m primitive recursion, but can't be defined with less.

[2]This result and following one are classical ones can find for instance in [Odi92]

The classes of Péter's *k-recursive functions* [Pé35, Odi92] are extensions of primitive recursive ones where a more powerful recursion is authorized. The function p can be k-ary and needs to decrease following the lexicographic ordering on $\mathbb{N}^k$. We can see for instance that primitive recursive functions are 1-recursives and *Ack* is 2-recursive. The union of k-recursive functions classes is the class of *multiply recursive functions* [Pé35].

Ordinals below ω^ω. We use Ω to denote the ordinal ω^ω. We shall work with set-theoretical ordinals less than Ω, written in Cantor's Normal Form.

We say that a given ordinal $0 < \alpha < \Omega$ has *degree* $d \in \mathbb{N}$, written $\deg(\alpha) = d$, if $\omega^{d+1} > \alpha \geq \omega^d$. In that case, α can be decomposed in a unique way under the form $\alpha = \omega^d.a + \alpha'$ with $0 < a \in \mathbb{N}$ and $\alpha' < \omega^d$. (We further let $\deg(0) = 0$.) For any $p \geq \deg(\alpha)$, $\alpha < \Omega$ can be written in a unique way under the form $\alpha = \omega^p.a_p + \omega^{p-1}.a_{p-1} + \cdots + \omega^1.a_1 + \omega^0.a_0$, shortly written $\sum_{i \leq p} \omega^i.a_i$, with $a_0, \ldots, a_p \in \mathbb{N}$. The set of limit ordinals $\leq \Omega$ is denoted *Lim*. Each $\lambda \in \mathit{Lim}$ comes with its canonical *fundamental sequence* $(\lambda_n)_{n\in\mathbb{N}}$ satisfying $\lambda_0 < \lambda_1 < \cdots < \lambda_n < \lambda_{n+1} < \cdots$ and $\lambda = \sup_n \lambda_n$. For limit ordinals below Ω, the fundamental sequence is given by

$$\Big(\sum_{i\leq p} \omega^i.a_i\Big)_n \stackrel{\text{def}}{=} \omega^p.a_p + \cdots + \omega^{r+1}.a_{r+1} + \omega^r(a_r - 1) + \omega^{r-1}.n$$

assuming a_r is the last nonzero coefficient, i.e., $0 = a_0 = a_1 = \ldots = a_{r-1} < a_r$. Equivalently,

$$\Big((\alpha + 1).\omega^{i+1}\Big)_n = \alpha.\omega^{i+1} + \omega^i.n \quad \text{for all } \alpha < \Omega \text{ and } i \in \mathbb{N}.$$

For example, if $\lambda = \omega^9.2 + \omega^3.6$, then $\lambda_n = \omega^9.2 + \omega^3.5 + \omega^2.n$. Observe that, for all $\lambda \in \mathit{Lim}$, $\lambda_n \sqsubseteq^o \lambda_{n+1}$ and $|\lambda_n| = |\lambda| + n - 1$. This scheme extends canonically up to ϵ_0 (and beyond) with $(\omega^\lambda)_n \stackrel{\text{def}}{=} \omega^{\lambda_n}$ etc. [Ros84, FW98].

2.3.2 Fast growing functions F_{ω^ω}

The functions $F_\alpha : \mathbb{N} \to \mathbb{N}$ are defined by induction over α:

$$F_0(n) \stackrel{\text{def}}{=} n + 1, \tag{D1}$$

$$F_{\alpha+1}(n) \stackrel{\text{def}}{=} F_\alpha^{n+1}(n) = \overbrace{F_\alpha(F_\alpha(\ldots F_\alpha}^{n+1 \text{ times}}(n)\ldots)), \tag{D2}$$

$$F_\lambda(n) \stackrel{\text{def}}{=} F_{\lambda_n}(n) \qquad \text{if } \lambda \in \mathit{Lim}. \tag{D3}$$

This induces $F_1(n) = 2n + 1$ and $F_2(n) = (n + 1)2^{n+1} - 1$. Expanding $F_3(n)$ needs a tower of n exponents. $F_\omega(n) = F_n(n)$, so that F_ω is a variant of Ackermann's function and is the first F_α that is not primitive-recursive. Notice that $F_{\omega^\omega}(n) = F_{\omega^n}(n)$.

Since we later construct a channel system that evaluates the F_α functions for $\alpha < \Omega$, it is a good exercise for the reader to try and get some intuition of what would $F_{\omega+1}(n)$, $F_{\omega+2}(n)$, $F_{\omega.2}(n)$ and $F_{\omega^2}(n)$ look like. For example

$$\begin{aligned} F_{\omega^2.3}(5) &= F_{\omega^2.2+\omega.5}(5) \\ &= F_{\omega^2.2+\omega.4+5}(5) \\ &= \underbrace{F_{\omega^2.2+\omega.4+4}(\ldots(F_{\omega^2.2+\omega.4+4}}_{\text{6 times}}(5))\ldots). \end{aligned}$$

2.3.3 Extended Grzegorczyk Hierarchy $\mathfrak{F}_{\omega^\omega}$

Our exposition is based on [Ros84, FW98, CT98] where more details can be found.

It is possible to define an ad-hoc primitive recursion on any data-structures equiped with a well order, but it is more convenient to have a general definition where the data-structure does not matter. This was achieved by defining hierarchies indexed by ordinals. Indeed, every well order corresponds to an ordinal. Kreisel [Kre52] developed such an extension, called *ordinal recursive functions* using the same kind of definition, but allowing more powerful well orders on integers. His definition gives rise to a hierarchy indexed by ordinal, where level α is defined with recursions using a function p decreasing according to an order $\leq_\alpha$ of ordinal α[3].

Weiner [Wai70, Wai72] defined an equivalent, but more convenient hierarchy, the *Extended Grzegorczyk Hierarchy*, a class $(\mathfrak{F}_\alpha)_\alpha$ of functions indexed by (an initial segment of the) ordinals[4]. $\mathfrak{F}_\alpha$ is the class of functions "elementary" in F_α, i.e. containing F_α, addition, zero, projections, and closed under compositions and limited recursion.

Write $\mathfrak{F}_{<\alpha}$ for $\bigcup_{\beta<\alpha}\mathfrak{F}_\beta$: It is known that $\mathfrak{F}_{<\omega}$ is exactly the set of primitive-recursive functions. That $\mathfrak{F}_{<\omega^k}$ is the set of Péter's k-recursive functions for $k \in \mathbb{N}$ [Rob65], that $\mathfrak{F}_{<\omega^\omega}$ is the set of multiply-recursive functions, and that $\mathfrak{F}_{<\epsilon_0}$ is the set of functions provably total in first-order Peano arithmetic [Wai72].

2.3.4 $\mathsf{F}_{\omega^\omega}$ complexity classes

Our purpose here needs a complexity class in the classical meaning of sets of problems computable by some time or space bounded Turing machine. The classes of functions here, does not fits our needs, first because those are function classes. Completeness for such kind of classes are tricky to express. For instance ReachLcs is computable by a function in $\mathfrak{F}_{\omega^\omega}$ not in $\mathfrak{F}_{<\omega^\omega}$. Furthermore, we can obtain results tighter than that using more classical

[3] i.e. there is a bijection between $(\mathbb{N}, \leq_\alpha)$ and $(\{\gamma|\gamma \leq \alpha\}, \leq)$ preserving the well order.
[4] below ϵ_0

notions of reduction. And finally the verification community is more used to classical complexity classes.

We define our new classes F_α as the problems solvable in time or space $F_\alpha \circ p$ for some p primitive recursive. We will always use primitive recursive reduction to show F_α-hardness. From the strictness of $\mathfrak{F}_\alpha$ hierarchy, we directly know that the F_α hierarchy is also strict.

The fact that p can be non elementary allow us to indistinguishably consider time or space bounds. Indeed, going from space to time bound only adds an exponential to the bound and $F_{\omega^\omega} \circ exp \circ p \geq exp \circ F_{\omega^\omega} \circ p$ (See section 3.1). $exp \circ p$ is of course primitive recursive if p is.

2.3.5 F_{ω^ω} and Higman's lemma

The result from Cichon and Tahhan Bittar on which we will base our upper bound is a concrete value to the H function defined in lemma 2.1.5.

Theorem 2.3.1 (Cichon, Tahhan Bittar [CT98, Cic07]) *There exists a primitive-recursive function f such that, for $n, k, p \in \mathbb{N}$, and Σ a finite alphabet, $H(n, k, p, |\Sigma|) \leq F_{\omega^{f(|\Sigma|)}}(\max(n, k, p))$*

the function f is left implicit in [CT98], for more informations see [Tou97].

Part I

Equivalences

Chapter 3

Fast-growing functions

This chapter is devoted to show

Theorem 3.0.2 ReachLcs *is* $\mathsf{F}_{\omega^\omega}$*-complete.*

To this end we will first prove that the longest controled bad sequences that are also valid LCS's runs are not significantly shorter than the overall longest controled bad sequences (given an alphabet and a first word).

3.1 The Fast-Growing Hierarchy

Fast-growing functions and monotonicity. We state some standard monotonicity properties in the form that will be convenient for our later developments. The size $|\alpha|$ of $\alpha = \sum_{i\leq p} \omega^i.a_i$ is $\sum_{i\leq p} a_i$.

Lemma 3.1.1 (Monotonicity) *For every $\alpha < \Omega$ and $n \in \mathbb{N}$:*

$$n < F_\alpha(n), \tag{3.1.1.a}$$
$$F_\alpha(n) \leq F_\alpha(n+1), \tag{3.1.1.b}$$
$$|\alpha| < F_\alpha(n) \qquad \text{if } n > 0. \tag{3.1.1.c}$$

In general, $\beta < \alpha$ does not entail $F_\beta(n) \leq F_\alpha(n)$, e.g., $F_m(n) > F_\omega(n)$ when $0 < n < m < \omega$. What is true is that, for all $\beta < \alpha$, F_β is *eventually* dominated by F_α, i.e., $F_\beta(n) < F_\alpha(n)$ for n large enough.

The next lemma provides more precise information on this issue.

Definition 3.1.2 (Embedding over ω) *Assume than, in normal form, $\alpha = \sum_{i\leq p} \omega^i.a_i$ and $\beta = \sum_{i\leq p} \omega^i.b_i$ are two ordinals below Ω. We say that α* embeds in *β, written $\alpha \sqsubseteq^o \beta$, when $a_i \leq b_i$ for all $i = 0, \ldots, p$.*

Observe that embedding between ordinals is only a partial order (in which, e.g., ω and 1 are incomparable), compatible with the usual linear ordering of ordinals ($\alpha \sqsubseteq^o \beta$ implies $\alpha \leq \beta$).

Lemma 3.1.3 (Monotonicity w.r.t. α) *For every $\alpha, \beta, \gamma < \Omega$ and $n, p \in \mathbb{N}$:*

$$F_\beta(n) \leq F_\alpha(n) \qquad \text{if } \beta \sqsubseteq^o \alpha, \tag{3.1.3.a}$$
$$F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n) \qquad \text{if } n > |\gamma|. \tag{3.1.3.b}$$

Observe that (3.1.3.b) is not a special case of (3.1.3.a) since $\gamma + \alpha \sqsubseteq^o \gamma + \omega^p + \alpha$ does not hold in general (e.g. $1 + 1 \not\sqsubseteq^o 1 + \omega + 1 = \omega + 1$) .

We now prove lemmas 3.1.1 and 3.1.3. The first four inequalities are proved by induction over α. We sometimes use simultaneous induction as when proving (3.1.1.b) and (3.1.3.a). Proving (3.1.3.b) requires the introduction of extra notations and tools, and is done in a later step.

3.1.1.a. $F_\alpha(n) > n$:

An easy induction over α. This directly entails

$$F^i_\alpha(n) \geq n + i. \tag{3.1.1.a'}$$

3.1.1.b. We actually prove $F_\alpha(n + i) \geq F_\alpha(n)$ for all $i \in \mathbb{N}$:

If $\alpha = 0$, we are done with $n + i + 1 \geq n + 1$.

If $\alpha = \alpha' + 1$ is a successor ordinal, then $F_\alpha(n + i) = F^{n+i+1}_{\alpha'}(n + i)$ (by D2) $\geq F^{n+1}_{\alpha'}(n + i)$ (by 3.1.1.a) $\geq F^{n+1}_{\alpha'}(n)$ (by ind. hyp.) $= F_\alpha(n)$.

If $\alpha \in \mathit{Lim}$, we rely on $\alpha_n \sqsubseteq^o \alpha_{n+i}$: $F_\alpha(n + i) = F_{\alpha_{n+i}}(n + i)$ (by D3) $\geq F_{\alpha_{n+i}}(n)$ (by ind. hyp.) $\geq F_{\alpha_n}(n)$ (by 3.1.3.a and ind. hyp.) $= F_\alpha(n)$.

3.1.1.c. $F_\alpha(n) > |\alpha|$ if $n > 0$:

If $\alpha = 0$, then $F_\alpha(n) = n + 1 > 0 = |\alpha|$.

If $\alpha = \alpha' + 1$ is a successor ordinal, then $F_\alpha(n) = F^{n+1}_{\alpha'}(n)$ (by D2) $> |\alpha'| + n$ (by ind. hyp. and using 3.1.1.a') $\geq |\alpha|$ since $|\alpha| = |\alpha'| + 1$ and $n > 0$.

If $\alpha \in \mathit{Lim}$, we rely on $F_\alpha(n) = F_{\alpha_n}(n) > |\alpha_n|$ (by ind. hyp.) $= |\alpha| - 1 + n \geq |\alpha|$ since $n > 0$.

3.1.3.a. $F_\beta(n) \leq F_\alpha(n)$ if $\beta \sqsubseteq^o \alpha$:

If $\alpha = 0$, then necessarily $\beta = \alpha$ and we are done.

If $\alpha = \alpha' + 1$ is a successor ordinal, we consider two cases. If $\beta = \beta' + 1$ is a successor, then $\beta' \sqsubseteq^o \alpha'$ so that $F_{\beta'}(n) \leq F_{\alpha'}(n)$ by ind. hyp. Now, using 3.1.1.b we deduce $F^{n+1}_{\beta'}(n) \leq F^{n+1}_{\alpha'}(n)$, i.e., $F_\beta(n) \leq F_\alpha(n)$ as required. If β is a limit, then $\beta \sqsubseteq^o \alpha'$ and $F_\beta(n) \leq F_{\alpha'}(n)$ (by ind. hyp) $\leq F^{n+1}_{\alpha'}(n)$ (by 3.1.1.a) $= F_\alpha(n)$.

If $\alpha \in \mathit{Lim}$, then $\beta \in \mathit{Lim}$ too and there are two cases: either $\beta \sqsubseteq^o \alpha_n$ or $\beta_n \sqsubseteq^o \alpha_n$. In both cases the induction hypothesis concludes immediately.

Proof of (3.1.3.b)**.** Recall that, for any $p \in \mathbb{N}$, an ordinal α can be decomposed in a unique way under the form $\alpha = \alpha_1.\omega^p + \alpha_2$ such that $\alpha_2 < \omega^p$. This decomposition satisfies both $\alpha_1.\omega^p \sqsubseteq^o \alpha$ and $\alpha_2 \sqsubseteq^o \alpha$. Also note that $\alpha + \omega^p = \alpha_1.\omega^p + \omega^p = (\alpha_1 + 1).\omega^p$.

3.1.3.b. $F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n)$ if $n > |\gamma|$: The proof is by induction over α. There are three cases.
1. $\alpha = 0$**:** we must prove that $F_\gamma(n) \leq F_{\gamma+\omega^p}(n)$. When $p = 0$, i.e., $\omega^p = 1$, we note that $\gamma \sqsubseteq^o \gamma + \omega^p$ so that (3.1.3.a) concludes. When $p > 0$, $\gamma + \omega^p \in Lim$. Decomposing γ as $\gamma_1.\omega^p + \gamma_2$ we obtain

$$\begin{aligned}
F_{\gamma+\omega^p}(n) &= F_{(\gamma_1+1).\omega^p}(n) \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.n}(n) \quad \text{(by D3)} \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.(n-1)+\omega^{p-2}.n}(n) \quad \text{(D3 again)} \\
&= F_{\gamma_1.\omega^p+\omega^{p-1}.(n-1)+\omega^{p-2}.(n-1)+\omega^{p-3}.n}(n) \\
&\cdots \\
&= F_{\gamma_1.\omega^p+[\sum_{i<p}\omega^i.(n-1)]+1}(n) \text{ (written } F_{\gamma_1.\omega^p+\gamma'}(n)).
\end{aligned}$$

Now $\gamma_2 \sqsubseteq^o \gamma'$ since $n > |\gamma|$. Hence $\gamma \sqsubseteq^o \gamma_1.\omega^p + \gamma'$ and (3.1.3.a) concludes.
2. $\alpha = \alpha' + 1$**:** then $F_{\gamma+\alpha'}(n) \leq F_{\gamma+\omega^p+\alpha'}(n)$ by ind. hyp. One deduces that $F^k_{\gamma+\alpha'}(n) \leq F^k_{\gamma+\omega^p+\alpha'}(n)$ for all $k \in \mathbb{N}$ using 3.1.1.b (and also 3.1.1.a to guarantee that all arguments are $> |\gamma|$). Putting $k = n + 1$, one obtains $F_{\gamma+\alpha}(n) \leq F_{\gamma+\omega^p+\alpha}(n)$ as required.
3. $\alpha \in Lim$**:** Let $d = \deg(\alpha)$. If $d = p$ then $\gamma + \alpha \sqsubseteq^o \gamma + \omega^p + \alpha$ so that (3.1.3.a) concludes. If $d > p$, then $\gamma + \alpha = \gamma + \omega^p + \alpha$ which is even more direct.

If now $d < p$ then $(\omega^p + \alpha)_n$ is $\omega^p + \alpha_n$. Decompose γ both as $\gamma_1.\omega^p + \gamma_2$ and as $\gamma'_1.\omega^d + \gamma'_2$. Note that $\gamma_1 + \omega^p = \gamma'_1 + \omega^p$ since $d < p$. Finally

$$\begin{aligned}
F_{\gamma+\omega^p+\alpha}(n) &= F_{(\gamma+\omega^p+\alpha)_n}(n) \quad \text{by D3} \\
&= F_{\gamma_1+\omega^p+\alpha_n}(n) \\
&= F_{\gamma'_1+\omega^p+\alpha_n}(n) \\
&\leq F_{\gamma'_1+\alpha_n}(n) \quad \text{by ind. hyp., noting that } |\gamma'_1| \leq |\gamma| \\
&= F_{(\gamma+\alpha)_n}(n) \\
&= F_{\gamma+\alpha}(n) \quad \text{by D3.}
\end{aligned}$$

3.2 Stacking ordinals

We use "stacks" to define a small-steps semantics for the F_α's that will be easier to simulate in channel systems.

Definition 3.2.1 *A* stack *(of length $k \in \mathbb{N}$) is a finite sequence $\pi = \alpha_1, \alpha_2, \ldots, \alpha_k$ of increasing ordinals $< \Omega$, i.e., $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k < \Omega$. We denote the empty stack ϵ and α, π the stack with α on top and continued by the stack π.*

Since a stack must list its elements in increasing order, there is a natural bijection between stacks and finite multisets over Ω. Hence we let $\mathcal{M}_f(\Omega)$ denote the set of stacks, and write $\pi <_{\text{ms}} \pi'$ when π is strictly smaller than π' in the multiset ordering inherited from the ordering of ordinals below Ω. This is a well-founded linear ordering with ϵ as minimal element [DM79].

We now extend the $(F_\alpha)_\alpha$ family with fast-growing functions indexed by stacks, denoted $F_\pi : \mathbb{N} \to \mathbb{N}$, and defined with:

$$F_\epsilon(n) \stackrel{\text{def}}{=} n, \qquad\qquad F_{\alpha,\pi}(n) \stackrel{\text{def}}{=} F_\pi(F_\alpha(n)).$$

Note that F_α is the same when we see α as an ordinal or as a stack of length one, hence we will not disambiguate.

The evaluation of some $F_\pi(n)$ can be expressed as a transformation system, where the manipulated objects are pairs $\langle\langle \pi \,;\, n \rangle\rangle$ of a stack π and a natural number n. Formally, we define a relation over $\mathcal{M}_f(\Omega) \times \mathbb{N}$, denoted $\to_R$, and defined by the three following "rewrite" rules:

$$\langle\langle 0, \pi \,;\, n \rangle\rangle \;\to_R\; \langle\langle \pi \,;\, n+1 \rangle\rangle \tag{R1}$$

$$\langle\langle \alpha+1, \pi \,;\, n \rangle\rangle \;\to_R\; \langle\langle \overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi \,;\, n \rangle\rangle \tag{R2}$$

$$\langle\langle \lambda, \pi \,;\, n \rangle\rangle \;\to_R\; \langle\langle \lambda_n, \pi \,;\, n \rangle\rangle \qquad \text{if } \lambda \in \mathit{Lim}. \tag{R3}$$

Observe that if π is a stack and $\langle\langle \pi \,;\, n \rangle\rangle \to_R \langle\langle \pi' \,;\, n' \rangle\rangle$ then π' is indeed a stack (i.e. ordinals are still ordered in π'), $\pi' <_{\text{ms}} \pi$ and $n' \geq n$. Note that $\to_R$ is deterministic.

Corollary 3.2.2 *$\to_R$ is terminating and convergent.*

The normal forms are the pairs $\langle\langle \pi \,;\, n \rangle\rangle$ with $\pi = \epsilon$.

Since rules R1–3 merely reformulate definitions D1–3 in terms of stacks, it follows that $\langle\langle \pi \,;\, n \rangle\rangle \to_R \langle\langle \pi' \,;\, n' \rangle\rangle$ implies $F_\pi(n) = F_{\pi'}(n')$. With Cor. 3.2.2, one deduces $\langle\langle \pi \,;\, n \rangle\rangle \to_R^* \langle\langle \epsilon \,;\, F_\pi(n) \rangle\rangle$.

Write $\leftrightarrow_R$ for $\to_R \cup \to_R^{-1}$. The previous observations entail

Lemma 3.2.3 *$\langle\langle \pi \,;\, n \rangle\rangle \leftrightarrow_R^* \langle\langle \pi' \,;\, n' \rangle\rangle$ iff $F_\pi(n) = F_{\pi'}(n')$.*

Notation 3.2.4 *When dealing with $\leftrightarrow_R$, it is convenient to decompose it as the union $\to_{R1} \cup \to_{R2} \cup \to_{R3} \cup \to_{S1} \cup \to_{S2} \cup \to_{S3}$ of the six relations defined by rules R1 to R3 and by* inverse rules *denoted S1 to S3, and defined such that $\to_{Si} = \to_{Ri}^{-1}$.*

$$\langle\langle \pi\ ;\ n+1\rangle\rangle \ \to_S\ \langle\langle 0, \pi\ ;\ n\rangle\rangle \tag{S1}$$

$$\langle\langle \overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi\ ;\ n\rangle\rangle \ \to_S\ \langle\langle \alpha+1, \pi\ ;\ n\rangle\rangle \tag{S2}$$

$$\langle\langle \lambda_n, \pi\ ;\ n\rangle\rangle \ \to_S\ \langle\langle \lambda, \pi\ ;\ n\rangle\rangle \text{ if } \pi = \alpha, \pi' \text{ with } \alpha \not< \lambda. \tag{S3}$$

3.3 A differential encoding of stacks

For $K \in \mathbb{N}$, we let $\Sigma_K \stackrel{\text{def}}{=} \{\omega^0, \omega^1, \omega^2, \ldots, \omega^{K-1}\} \cup \{\mathsf{I}\}$ be an alphabet with $K+1$ symbols, that we use to encode stacks (restricted to ordinals $< \omega^K$). The symbols "ω^p" denote the corresponding finite powers of the ordinal ω. In particular, "ω^0" and "ω^1" denote, respectively, the ordinals 1 and ω.

We first explain the encoding informally. Consider the following word $u \in \Sigma_K^*$:

$$u \ = \ \omega^0\omega^0\mathsf{I}\omega^3\omega^1\mathsf{I}\mathsf{I}\omega^1\omega^0\mathsf{I}.$$

One reads u from left to right. While reading u, all the encountered ordinal symbols are added up, giving rise to a notion of current sum, or height. A tally symbol "I" codes for an ordinal in the stack: *each* I *stands for one copy of the current sum.* In our example, the stack of length 4 associated with u, is

$$\Pi(u) \ = \ 2,\ \omega^3+\omega,\ \omega^3+\omega,\ \omega^3+\omega.2+1.$$

(Indeed $\omega^0+\omega^0 = 2$ and $\omega^0+\omega^0+\omega^3+\omega^1 = \omega^3+\omega$. Furthermore, $\Pi(u)$ contains two occurrences of $\omega^3+\omega$ because u contains two tally symbols immediately after the first occurrence of ω^1.)

Formally, the correspondence $\Pi : \Sigma_K^* \to \mathcal{M}_f(\Omega)$ and the height function $h : \Sigma_K^* \to \Omega$ are defined by induction over u:

$$h(\epsilon) \stackrel{\text{def}}{=} 0; \qquad h(u\mathsf{I}) \stackrel{\text{def}}{=} h(u); \qquad h(u\omega^i) \stackrel{\text{def}}{=} h(u) + \omega^i;$$

$$\Pi(\epsilon) \stackrel{\text{def}}{=} \epsilon; \qquad \Pi(u\mathsf{I}) \stackrel{\text{def}}{=} \Pi(u), h(u); \qquad \Pi(u\omega^i) \stackrel{\text{def}}{=} \Pi(u).$$

Observe that $\Pi(u)$ is indeed a stack, i.e., $\Pi(u)$ lists increasing ordinals, since $h(u.v) \geq h(u)$ for all u, v.

Remark 3.3.1 *We call this encoding* differential *since the ω^p symbols in Σ_K are not used to directly represent an α_j in a stack $\pi = \alpha_1, \ldots, \alpha_k$. Rather they represent the "difference" $\alpha_j - \alpha_{j-1}$ that must be added to the previous ordinal in order to obtain α_j.*

Any $u \in \Sigma_K^*$ encodes a stack, and any stack below ω^K can be encoded with some $u \in \Sigma_K^*$. Such an encoding is not unique. However, there is a unique shortest one, called a *pure* encoding.

Definition 3.3.2 (Pure encodings) *An encoding $u \in \Sigma_K^*$ is* pure *if (1) it does not end with an ω^i symbol, and (2) it does not contain a factor of the form $\omega^i\omega^j$ with $i < j$.*

Note that the pure encodings are a regular subset of Σ_K^*.

The idea behind purity is to forbid useless symbols in an encoding. If u is not pure, this is witnessed by some occurrence of some ω^i. Removing that occurrence yields some shorter u' with $\Pi(u') = \Pi(u)$. Hence any impure u can be replaced by a shorter equivalent encoding. Reciprocally, if u is pure and u' is shorter than u, then $\Pi(u') \neq \Pi(u)$.

Purity allows transferring the monotonicity lemmas from stacks to their encodings. The rest of this section proves the following proposition.

Proposition 3.3.3 *Let $u, v \in \Sigma_K^*$ and $n > 0$. If $u \sqsubseteq v$ and v is pure, then $F_{\Pi(u)}(n) \leq F_{\Pi(v)}(n)$.*

The crux of the proof is the case where u and v only differ by one ordinal symbol:

Lemma 3.3.4 *$F_{\Pi(v_1v_2)}(n) \leq F_{\Pi(v_1\omega^p v_2)}(n)$ when $v_1\omega^p v_2$ is pure and $n > 0$.*

Proof. Write $\pi = \alpha_1, \ldots, \alpha_k$ for $\Pi(v_1\omega^p v_2)$ and $\pi' = \alpha'_1, \ldots, \alpha'_k$ for $\Pi(v_1v_2)$ (clearly, π and π' have same length). Write $l \in \{0, \ldots, k-1\}$ for the length of $\Pi(v_1)$. Then $\alpha'_i = \alpha_i$ for $i = 1, \ldots, l$ and, for $i = l+1, \ldots, k$, we can write α_i and α'_i under the following form:

$$\alpha_i = h(v_1) + \omega^p + \beta_i, \qquad \alpha'_i = h(v_1) + \beta_i,$$

where $\beta_{l+1}, \ldots, \beta_k$ is simply $\Pi(v_2)$. There are now two cases:
(1) If v_1 ends with some "l" symbol (or $v_1 = \epsilon$), then $h(v_1) = \alpha_{l-1}$, putting $\alpha_0 = 0$ by convention. Observe that $F_{\alpha'_1,\ldots,\alpha'_l}(n) > |\alpha_l|$ as a consequence of (3.1.1.c) and (3.1.1.a). Thus (3.1.3.b) applies and we can prove that $F_{\alpha'_1,\ldots,\alpha'_i}(n) \leq F_{\alpha_1,\ldots,\alpha_i}(n)$ for all $i = l+1, \ldots, k$ by induction over i.
(2) Otherwise v_1 ends with some ω^r symbol. Observe that $r \geq p$ since $v_1\omega^p v_2$ is pure. This implies that $\alpha'_i \sqsubseteq^o \alpha_i$ for $i \geq l$ (and hence for all i's). We conclude with (3.1.3.a) and the other monotonicity properties. □

The case where u and v differ by one tally symbol is easier.

Lemma 3.3.5 *$F_{\Pi(v_1v_2)}(n) \leq F_{\Pi(v_1\mathsf{l}v_2)}(n)$.*

Proof. **[Sketch]** $\Pi(v_1v_2)$ is obtained by removing one ordinal somewhere in $\Pi(v_1\mathsf{l}v_2)$. Hence we can conclude with (3.1.1.a) and the other monotonicity properties. □

There remains to deal with the case where u and v differ by more than one symbol. Write $u \sqsubseteq_k v$ when $u \sqsubseteq v$ and $|v| = |u| + k$. Write $u \equiv_\Pi v$ when $\Pi(u) = \Pi(v)$.

Lemma 3.3.6 *If $u \sqsubseteq v$ and v is pure then there is a sequence*

$$u \equiv_\Pi u_1 \sqsubseteq_1 u_2 \sqsubseteq_1 \cdots \sqsubseteq_1 u_n = v$$

where all u_i's, $i = 1, \ldots, n$, are pure.

Proof. We let u_1 be the pure encoding of $\Pi(u)$: this is a subword of u, hence of v too. The sequence $u_1 \sqsubseteq_1 u_2 \sqsubseteq_1 \cdots \sqsubseteq_1 u_n$ is obtained by inserting in u_1, one by one, all the (occurrences of) symbols that are in v but missing in u_1. One first inserts all the missing tally symbols (in no particular order) and then, in a second phase, all the missing ordinal symbols (in no particular order). This ensures that all the u_i's are pure: In the first phase, a u_i inherits purity from u_{i-1}, starting with u_1, since $x\mathsf{I}y$ is pure when xy is. In the second phase, a u_i inherits purity from u_{i+1}, starting from $u_n = v$, since xy is pure when $x\omega^j y$ is. □

With Lemma 3.3.6 one can reduce Prop. 3.3.3 to repetitive applications of Lemmas 3.3.4 and 3.3.5, which concludes the proof of Proposition 3.3.3.

3.4 Fast-growing functions via lossy channels

3.4.1 A channel system that computes fast-growing functions

In this section, we construct a LCS, called W_K, that weakly computes the F_α functions for all $\alpha < \omega^K$. It can also weakly compute their inverses F_α^{-1} as we explain later.

W_K uses two channels. The first channel, `p`, stores a word $u \in \Sigma_K^*$ that encodes a stack of ordinals as in Section 3.3. The second channel, `d`, stores a number $n > 0$ in unary (using n times the tally symbol, or I^n). Thus a pair $\langle\langle \pi \,;\, n \rangle\rangle$ is stored in two channels. An extra marker symbol # is written at the end of these encodings to recognize their extremity during the manipulations.

The overall structure of W_K is illustrated in Fig. 3.1 (see Appendix 3.6.1 for the details of the components). When explaining its behaviour, we call "*single-pass* run" any run that does not visit the state `loop`. In state `beg`, W_K will traverse one of six possible "components" where it transforms the pair $\langle\langle \pi \,;\, n \rangle\rangle$ (more precisely, its encoding) stored in the channels by one application of the rewriting rules R1 to R3 (from section 3.1), or the inverse rules S1 to S3. With our encodings of pairs, each of these rules can be seen as a finite-state transduction. The LCS's that implement these components are described in Appendix 3.6.1. Implementing one rewriting step, W_K will replace $\langle\langle \pi \,;\, n \rangle\rangle$ with the resulting $\langle\langle \pi' \,;\, n' \rangle\rangle$, that is, unless message losses corrupt the result. Then W_K reaches state `wrap` where it reads the end

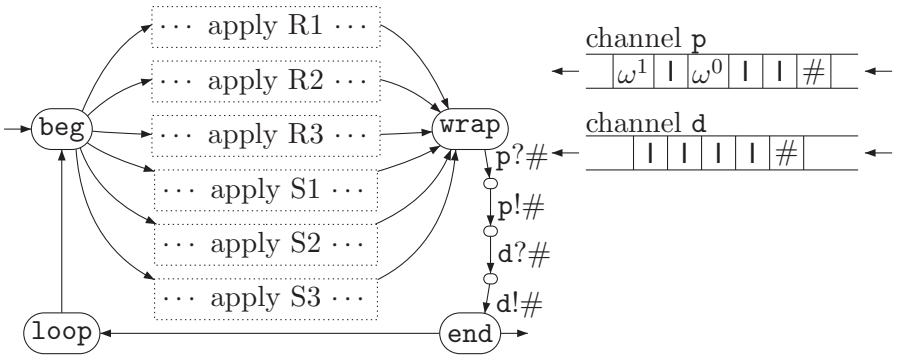


Figure 3.1: A schematic view of W_K.

markers and writes them back after $\langle\langle \pi' \,;\, n' \rangle\rangle$. In state `end` W_K can terminate and exit, or loop back to `beg` and transform $\langle\langle \pi' \,;\, n' \rangle\rangle$ again, therefore computing the transitive closure of $\leftrightarrow_R$.

The construction ensures the following features:

sanity check: The rule components assume that each channel contain a Σ_K-word followed by at most one marker symbol #. With this assumption, the components check that the channels contain proper inputs. Formally, there is a single-pass run from $(\texttt{beg}, u\#, v\#)$ to state `end` only if u is some pure encoding, and v is some I^n for some $n > 0$. If this is not the case, on impure u or incorrect v, W_K will stop in a deadlock. If a final # is missing, W_K will loop without reaching `end`.

one-pass transduction: If the channels contain proper inputs, a single-pass run from $(\texttt{beg}, u\#, v\#)$ to some $(\texttt{wrap}, w, w')$ reads u and v completely, write some new data u' and v', and does not touch the end markers. Hence $w = \#u'$ and $w' = \#v'$.

rule applicability: When going from `beg` to `end`, W_K chooses non-deterministically what rule component will be traversed. It may be the case that the corresponding rule is not applicable to the current channel contents: this is checked by W_K and it will stop in a deadlock if the rule is not applicable.

We can now state formally how W_K implements $\leftrightarrow_R$.

Lemma 3.4.1 (Single-pass perfect runs in W_K) *Assume that $u, u' \in \Sigma_K^*$ are the pure encodings of two stacks π and π'. Assume $n, n' > 0$. Then $\langle\langle \pi \,;\, n \rangle\rangle \leftrightarrow_R \langle\langle \pi' \,;\, n' \rangle\rangle$ if, and only if, W_K has a single-pass* perfect *run of the form*

$$(\texttt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*}_{\text{perf}} (\texttt{end}, u'\#, \mathsf{I}^{n'}\#).$$

Proof. [**Idea**] The "⇒" direction is obvious since W_K implements exactly the six rules that define $\leftrightarrow_R$ (see Appendix 3.6.1). Reciprocally, the **rule-applicability** features ensure that `end` is only reached by one proper step of rewriting. Hence the "⇐" direction. □

The corollary is:

Theorem 3.4.2 (W_K weakly computes the F_α's) *Assume that $u, u' \in \Sigma_K^*$ are the pure encodings of two stacks π and π'. Assume $n, n' > 0$. Then $F_\pi(n) \geq F_{\pi'}(n')$ if, and only if, W_K has a* lossy *run of the form*

$$(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{end}, u'\#, \mathsf{I}^{n'}\#).$$

Proof. (⇒): Write a for $F_\pi(n)$ and b for $F_{\pi'}(n')$. By Lemma 3.2.3, there are rewriting sequences of the form $\langle\langle\pi;\ n\rangle\rangle \leftrightarrow_R^* \langle\langle\epsilon;\ a\rangle\rangle$ and $\langle\langle\epsilon;\ b\rangle\rangle \leftrightarrow_R^* \langle\langle\pi';\ n'\rangle\rangle$, and it is even possible to ensure $\langle\langle\pi\ ;\ n\rangle\rangle \leftrightarrow_R^+ \langle\langle\epsilon\ ;\ a\rangle\rangle$ by inserting extra rewriting steps. These rewriting steps entail the existence of corresponding single-pass perfect runs (Lemma 3.4.1). Concatenating these, we deduce that W_K has two perfect runs of the form $(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*}_{\text{perf}} (\mathtt{end}, \#, \mathsf{I}^a\#)$ and $(\mathtt{beg}, \#, \mathsf{I}^b\#) \xrightarrow{*}_{\text{perf}} (\mathtt{end}, u'\#, \mathsf{I}^{n'}\#)$. Since $a \geq b$, there also exists a lossy run $(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{end}, \#, \mathsf{I}^b\#)$ obtained by losing $a - b$ tally symbols in `d` during the last single-pass of the first run. Concatenating with the second run we obtain the required lossy run $(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*}_{\text{perf}} (\mathtt{end}, u'\#, \mathsf{I}^{n'}\#)$.

(⇐): Write k for the number of times the run $(\mathtt{end}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{end}, u'\#, \mathsf{I}^{n'}\#)$ visits state `loop`. We prove the implication by induction over k. If $k = 0$, then the run has length zero, $u = u'$, $n = n'$ and we are done. Now assume $k > 0$. The run has the form

$$(\mathtt{end}, u\#, \mathsf{I}^n\#) \to (\mathtt{loop}, u\#, \mathsf{I}^n\#) \to \overbrace{(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{end}, w, w')}^{\text{single-pass}} \underbrace{\xrightarrow{*} (\mathtt{end}, u'\#, \mathsf{I}^{n'}\#)}_{k-1 \text{ remaining visits}}.$$

After two steps, the first single-pass reaches $(\mathtt{end}, w, w')$ by traversing one of the six components of W_K. Traversing the same component, W_K has a perfect single-pass run $(\mathtt{beg}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{end}, v\#, \mathsf{I}^m\#)$ satisfying

$$F_{\Pi(u)}(n) = F_{\Pi(v)}(m) \tag{3.1}$$

thanks to Lemma 3.4.1. With our write-lossy semantics, the **one-pass transduction** features ensure that w and w' are subwords of, respectively, $v\#$ and $\mathsf{I}^m\#$. Observe that w and w' are proper inputs, i.e., w is some $v'\#$ for some pure v, and w' is some $\mathsf{I}^{m'}\#$ for some $m' > 0$. Indeed, either

$k > 1$ and the **sanity check** features require a proper input (otherwise the next single-pass would not succeed), or $k = 1$, implying that $w = u'\#$ and $w' = \mathsf{l}^{n'}\#$. Therefore, the induction hypothesis applies, yielding

$$F_{\Pi(v')}(m') \geq F_{\Pi(u')}(n'). \tag{3.2}$$

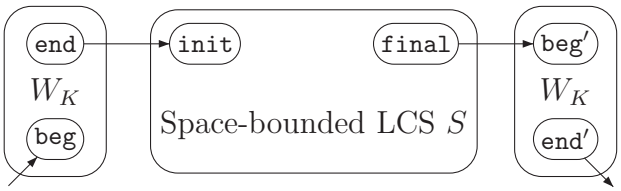
Now, since $v'\# \sqsubseteq v\#$ and $\mathsf{l}^{m'} \sqsubseteq \mathsf{l}^{m}\#$, i.e., $v' \sqsubseteq v$ and $m' \leq m$, since v is pure and $m' > 0$, Lemmas 3.1.1.b and 3.3.3 imply

$$F_{\Pi(v')}(m') \leq F_{\Pi(v)}(m). \tag{3.3}$$

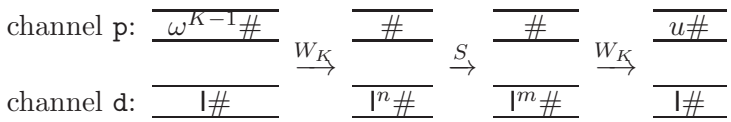
Combining (3.1–3.3) provides the required $F_{\Pi(u)}(n) \geq F_{\Pi(u')}(n')$. □

3.4.2 Lower bounds for LCS's

W_K can be used to check that a possibly lossy run is actually perfect in space-bounded LCS's. Formally, a space-bounded LCS is a LCS operating on one channel and whose transition rules write exactly as many messages as they read (see [Sch02]). Hence the number of messages in the channel remains constant during perfect runs, and it can only decrease during lossy runs. Given a space-bounded S, and some $K \in \mathbb{N}$, we build the LCS S_K



by inserting two copies of W_K, one before and one after S, as schematically depicted above. S does not use `p`, only `d`. The idea is that the first W_K will be started with a pair $\langle\langle \omega^{K-1} ; 1\rangle\rangle$ in the channels, will write some large $\mathsf{l}^n\#$ in `d`, that will be used by S, that will return $\mathsf{l}^m\#$ to be fed to the second W_K:



The construction of S_K has some simple sanity checks (not depicted) between the W_K's and the S part, ensuring that the # markers are not lost, etc.

Now, assume S_K has a run of the form

$$\begin{aligned}(\mathtt{beg}, \omega^{K-1}\#, \mathsf{l}\#) &\xrightarrow{*} (\mathtt{end}, u\#, \mathsf{l}^n\#) \\ &\to (\mathtt{init}, u\#, \mathsf{l}^n\#) \xrightarrow{*} (\mathtt{final}, u\#, \mathsf{l}^m\#) \\ &\to (\mathtt{beg}', u\#, \mathsf{l}^m\#) \xrightarrow{*} (\mathtt{end}', \omega^{K-1}\#, \mathsf{l}\#)\end{aligned} \tag{†}$$

Then the construction of W_K ensures that $n \leq F_{\omega^{K-1}}(1)$ and $F_{\omega^{K-1}}(1) \leq m$ (by Theorem 3.4.2). Since S is space-bounded, $n \geq m$. Hence in a run like (†), for the sub-run $(\mathtt{init}, u\#, \mathsf{I}^n\#) \xrightarrow{*} (\mathtt{final}, u\#, \mathsf{I}^m\#)$ to be perfect, n needs to be equal to m $(= F_{\omega^{K-1}}(1))$. Reciprocally, a run $(\mathtt{beg}, \omega^{K-1}\#, \mathsf{I}\#) \xrightarrow{*} (\mathtt{end}', \omega^{K-1}\#, \mathsf{I}\#)$ in S'_K must be decomposable under the form of (†).

Corollary 3.4.3 *S_K has a run from $(\mathtt{beg}, \omega^{K-1}\#, \mathsf{I}\#)$ to $(\mathtt{end}', \omega^{K-1}\#, \mathsf{I}\#)$ if, and only if, S has an accepting* perfect *run using space $F_{\omega^{K-1}}(1)$.*

Theorem 3.4.4 ReachLcs *is $\mathsf{F}_{\omega^\omega}$-hard.*

Proof. Let P be a $\mathsf{F}_{\omega^\omega}$ problem, i.e. in space $F_{\omega^\omega} \circ p$ for some primitive recursive p and x be an input to that problem. First recall that perfect space-bounded CS's have the same computational power than space-bounded Turing machines and are in fact equivalent modulo LogSpace reduction. Then we can consider, without loss of generality, that P is given as a space-bounded CS S_P. Using S_K, it is possible to reduce the problem of whether a space-bounded LCS S has an accepting *perfect* run using space $\leq F_{\omega^{K-1}}(1)$ to a LCS-reachability question of size polynomial in K and $|S|$. Then, using this construction which is obviously primitive recursive, we can build in primitive recursive time a system S'_P of polynomial size in $p(|x|)$ and S_P that simulates S_p on space bounded by $F_{\omega^{p(|x|)+1}}(1)$. Since $F_{\omega^\omega}(p(|x|)) = F_{\omega^{p(|x|)}}(p(|x|)) \leq F_{\omega^{p(|x|)}}(|\omega^{p(|x|)}|) \leq F^2_{\omega^{p(|x|)}}(1) = F_{\omega^{p(|x|)+1}}(1)$ from lemma 3.1.1.c, S'_P has enough space to effectively simulate S_P. □

There exists a similar construction, again using W_K, that reduces the existence of perfect space-bounded runs to *termination* of LCS's, rather than *reachability* (along the lines of [Sch02, section 4.2]). The consequences are similar:

Theorem 3.4.5 *Termination for lossy channel systems is $\mathsf{F}_{\omega^\omega}$-hard.*

3.5 Upper bounds

In this section, we explain how Cichon's and Tahhan Bittar's analysis of Higman's Lemma 2.1.5 leads to:

Observation 3.5.1 *Reachability and termination for lossy channel systems are computable in time $F_{\omega^\omega} \circ p$ with p primitive recursive.*

Since we showed that these problems are $\mathsf{F}_{\omega^\omega}$-hard, this concludes the proof of our main result 3.0.2.

Bounding termination and reachability. When configurations of a LCS are compared with $\sqsubseteq$, there are similar notions of a bad, and of an r-bad, run $\sigma_0 \to \sigma_1 \to \ldots \to \sigma_n$. With such a run, we associate its sequence of channel contents $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_n$, obtained by forgetting the control state part of a configuration $\sigma_i = (q_i, \boldsymbol{u}_i)$. Observe that if the run is bad then the sequence $(\boldsymbol{u}_i)_{i=0,\ldots,n}$ is $(|\mathtt{Q}| - 1)$-bad (by the pigeonhole principle, in a system with $|\mathtt{Q}|$ states). Hence thanks to 2.3.1, bad runs in systems with $|\mathtt{Q}|$ states, $|\mathtt{C}|$ channels, starting and with σ_0 with alphabet of size $|\mathtt{M}|$, have length bounded in $F_{\omega^{f(|\mathtt{M}|)}}(\max(|\mathtt{Q}|, |\mathtt{C}|, |\sigma_0|))$
Now, since deciding termination can be done by checking that all runs from σ_0 are bad (this is the classic algorithm, see [Fin94, AJ96b, FS01]), termination of LCS is in primitive recursive time in $F_{\omega^{f(|\mathtt{M}|)}}(\max(|\mathtt{Q}|, |\mathtt{C}|, |\sigma_0|))$, hence in $\mathsf{F}_{\omega^{f(|\mathtt{M}|)}}$, thus in $\mathsf{F}_{\omega^\omega}$.
Regarding reachability, the backward-chaining algorithm [AJ96b, FS01] also builds a bad sequence of configurations: the minimal elements of $\mathit{Pre}^*(\mathit{Goal})$ for some upward-closed $\mathit{Goal} \subseteq \mathit{Conf}$ defined by its minimal elements. By construction, this sequence is controlled (even though it is not a run *per se*). Hence the running time of the algorithm is bounded by some $F_{\omega^{f(|\mathtt{M}|)}} \circ p$ too.

We observe that these two algorithms handle equally well the different lossy semantics (see Section 2.2.4).

Variants and restrictions. From the above observations, one concludes that termination and reachability are in $\mathsf{F}_{\omega^{f(p)}}(|S|)$ if we restrict ourselves to LCS's S having a message alphabet of cardinal at most p. This indicates that the cardinal of $\mathtt{M}$, not the number of channels, or the number of control states, or the size of the initial configuration, is the key parameter affecting complexity. (Note that, in section 3.4, we used an alphabet of size $K + 2$ to build LCS's whose complexity was not in $\mathsf{F}_{\omega^{K-1}}$.) Since the cumulative hierarchy $(\mathsf{F}_\alpha)_{\omega \leq \alpha < \omega^\omega}$ is strict, we deduce that increasing the alphabet size of LCS's gives rise to a strict hierarchy of verification problems (more precisely, a hierarchy that contains a strict sub-hierarchy). This further even "allows to prove" why LCS's with large message alphabets cannot be simulated by LCS's with a fixed alphabet (more exactly, not via a primitive recursive reduction) unlike the way Turing machines can be restricted to alphabets of size 2. Contrast this with the fact that LCS's with l channels can be simulated (via a many-one polynomial-time reduction) by LCS's with a single channel and an alphabet enlarged with a single extra symbol.

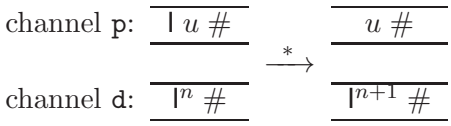
In the same spirit, let us observe that Lossy Counter Machines [May03a], which can be seen as LCS's where the alphabet has size 1, can be verified in $\mathfrak{F}_l$, where l is the number of counters. This is a direct consequence of McAloon's bounds on the length of bad sequences in $\mathbb{N}^l$ ordered by the component-wise ordering [FFSS10] (see also [McA84]). When l is not

fixed, reachability and termination for these Lossy Counter Machines is F_ω-complete [Sch02].

3.6 Appendix

3.6.1 Channel systems that implement stack rewriting

Rule R1 is "$\langle\langle 0, \pi\,;\, n\rangle\rangle \;\to_R\; \langle\langle \pi\,;\, n+1\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



where u is pure. This transformation is performed by the LCS depicted in

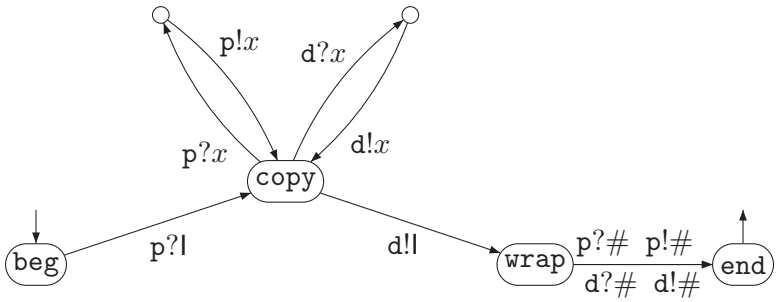


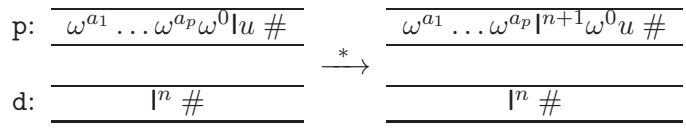
Figure 3.2: LCS component that implements rule R1 (assuming purity)

Fig. 3.2. Here, and in the rest of this section, two simplifying conventions are assumed:

Purity check: the system depicted in Fig. 3.2 does not check that p contains a pure encoding. This is for improving the clarity of the diagram but, of course, it is easy to check purity (a simple regular property) while performing the transformation. We assume our system deadlocks before reaching state end when purity is not satisfied.

Abbreviated rules: our pictures for LCS uses implicit variables or patterns in order to describe several similar rules at once. For example, the loop $\texttt{copy} \xrightarrow{\texttt{p?}x\,\texttt{p!}x} \texttt{copy}$ in Fig. 3.2 uses x as a variable standing for any message $m \in \texttt{M}$ so that, letting $k = |\texttt{M}|$, it abbreviates k loops (each with a different intermediary state). Other examples are i in Fig. 3.3, a in Fig. 3.4, and so on. For these variables, the allowed instantiations are sometimes constrained, as with "$(i > 0)$" or "$(i > a)$" in Fig. 3.3 and 3.4.

Rule R2 is "$\langle\langle \alpha+1, \pi\,;\, n\rangle\rangle \;\rightarrow_R\; \langle\langle \overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi\,;\, n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



where we assume that $\omega^0 u$ is pure, otherwise the ω^0 is not copied to the right-hand side, as is done in state $*$ (Fig. 3.3).

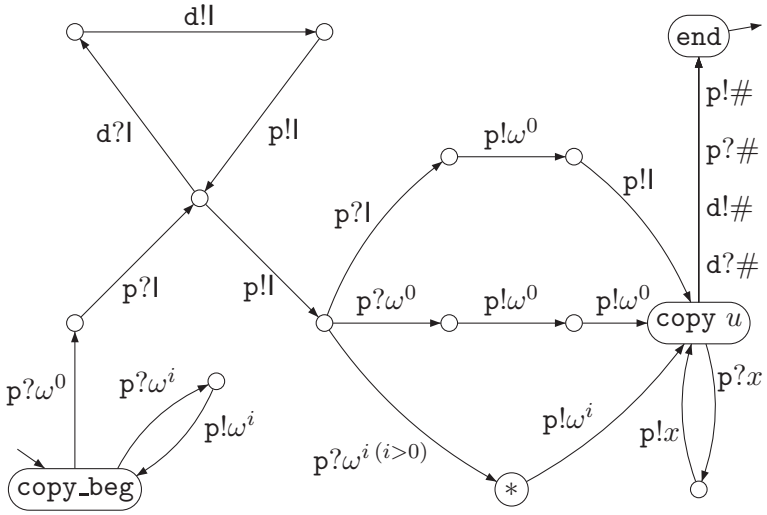
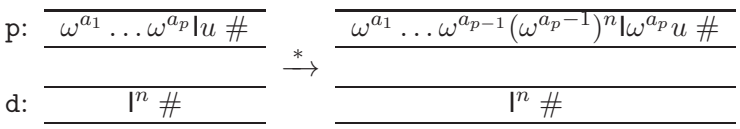


Figure 3.3: LCS component that implements rule R2 (assuming purity)

Our channel system is actually more complex than depicted in Fig. 3.3 since it only accepts pure encodings. For example, it will check that $K > a_1 \geq a_2 \geq \cdots a_{p-1} \geq a_p = 0$ while performing the first copy loop (in state `copy_beg`).

Rule R3 is "$\langle\langle \lambda, \pi\,;\, n\rangle\rangle \;\rightarrow_R\; \langle\langle \lambda_n, \pi\,;\, n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:



where it is assumed that $\omega^{a_p} u$ is pure, otherwise the ω^{a_p} is not copied to the right-hand side (see state $*$ in Fig. 3.4). On top of the usual implicit check for purity "$a_1 \geq a_2 \geq \cdots \geq a_p$", the system depicted in Fig. 3.4 checks that $(a =) a_p > 0$ so that $\alpha_1 \in \mathit{Lim}$.

Rule S1 is "$\langle\langle \pi\,;\, n+1\rangle\rangle \;\rightarrow_S\; \langle\langle 0, \pi\,;\, n\rangle\rangle$". With our differential encoding of stacks, this requires the following transformation:

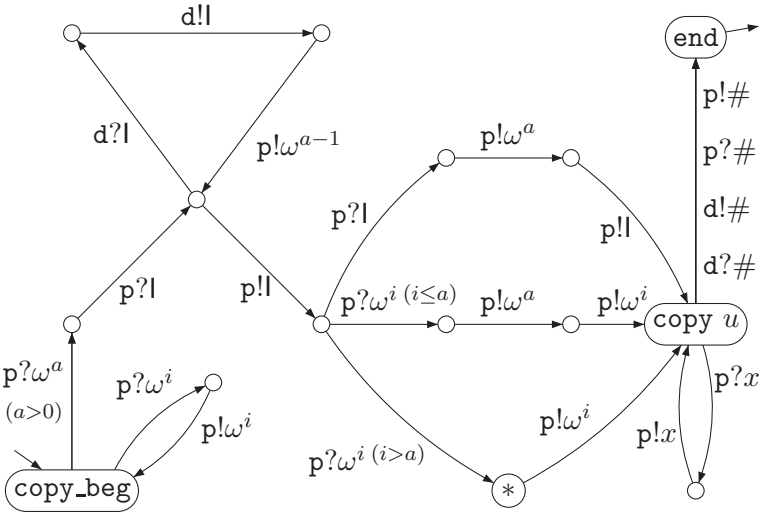


Figure 3.4: LCS component that implements rule R3 (assuming purity)

$$\begin{array}{lcc} \mathtt{p}: & \overline{u\ \#} & \\ & & \xrightarrow{*} \\ \mathtt{d}: & \overline{\mathtt{I}^{n+1}\ \#} & \end{array} \qquad \begin{array}{c} \overline{\mathtt{I}\ u\ \#} \\ \\ \overline{\mathtt{I}^{n}\ \#} \end{array}$$

The component that implements S1 behaves like the component for R1, only backwards.

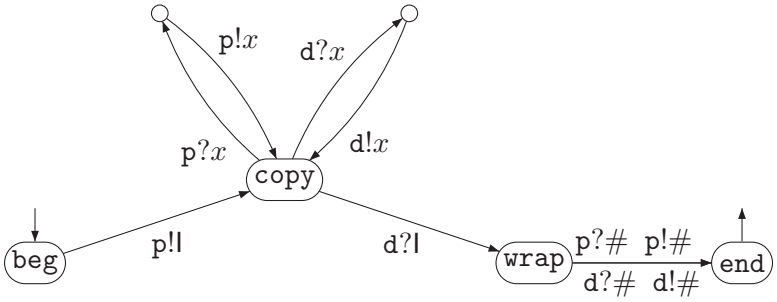


Figure 3.5: LCS component that implements rule S1 (assuming purity)

Rule S2 is "$\langle\langle\overbrace{\alpha, \alpha, ..., \alpha}^{n+1 \text{ times}}, \pi\ ;\ n\rangle\rangle \rightarrow_S \langle\langle \alpha+1, \pi\ ;\ n\rangle\rangle$" assuming that α does not occur in π.

With our differential encoding of stacks, this requires the following transformation:

$$\begin{array}{lcc} \mathtt{p}: & \overline{\omega^{a_1} \ldots \omega^{a_p} \mathtt{I}^{n+1} u\ \#} & \\ & & \xrightarrow{*} \\ \mathtt{d}: & \overline{\mathtt{I}^{n}\ \#} & \end{array} \qquad \begin{array}{c} \overline{\omega^{a_1} \ldots \omega^{a_p} \omega^{0} \mathtt{I} u\ \#} \\ \\ \overline{\mathtt{I}^{n}\ \#} \end{array}$$

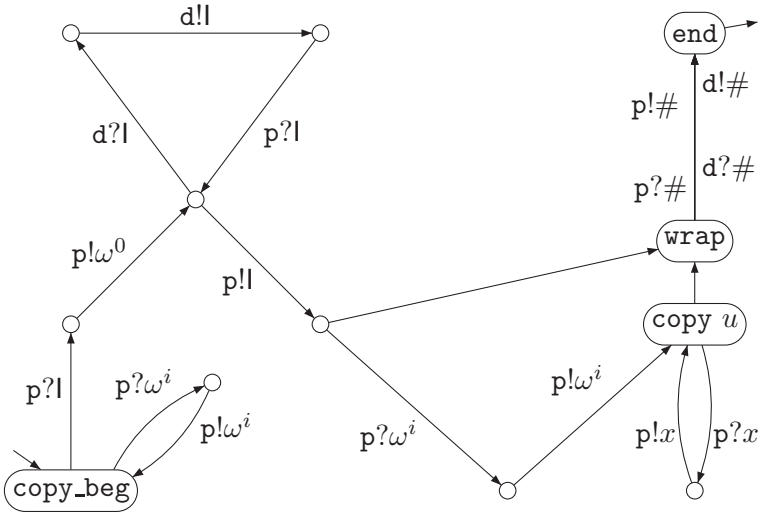


Figure 3.6: LCS component that implements rule S2 (assuming purity)

where it is now checked that u does not start with I. The component that implements S2 is depicted in Fig. 3.6. An important feature is the ability to check that the number $n+1$ of tally symbols after the first ordinal symbols in p matches the number in I^n in d. If there is a mismatch, our system will never reach end.

Rule S3 is "$\langle\langle \lambda_n, \pi\,;\ n\rangle\rangle \ \to_S\ \langle\langle \lambda, \pi\,;\ n\rangle\rangle$" assuming that π does not start with some $\alpha_2 < \lambda$.

With our differential encoding of stacks, this requires the following transformation:

$$\begin{array}{lcc} \mathtt{p}: & \overline{\omega^{a_1}\ldots\omega^{a_p}(\omega^a)^n\mathsf{I}u\,\#} & \overline{\omega^{a_1}\ldots\omega^{a_p}\omega^{a+1}\mathsf{I}v\,\#} \\ & \xrightarrow{\ *\ } & \\ \mathtt{d}: & \overline{\mathsf{I}^n\,\#} & \overline{\mathsf{I}^n\,\#} \end{array}$$

where it is required that $a_p > a$, and where v is obtained from u. More precisely, if u is ϵ then $v = \epsilon$, while if u is some $\omega^b u'$, then $v = u$ if $b > a+1$ and $v = u'$ if $b = a+1$. The rule does not apply if $b \le a$ or if u starts with I, indicating that $\alpha_2 < \lambda$. Here again, the component has to perform a crucial comparison: the number n that is encoded in d must match the number of ω^a symbols in the encoding of the first ordinal in the stack. (As before, the depiction in Fig. 3.7 does not feature the implicit purity check.)

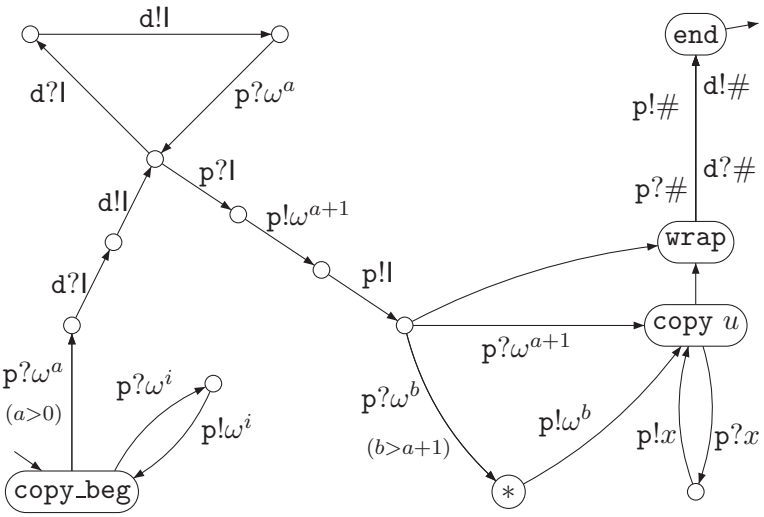


Figure 3.7: LCS component that implements rule S3 (assuming purity)

Chapter 4

Post Embedding Problem

Problem $\mathsf{PEP}^{\mathrm{reg}}$

Instance: Two finite alphabets Σ and Γ, two morphisms $u, v : \Sigma^* \to \Gamma^*$, and a regular language $R \subseteq \Sigma^*$.

Question: Does there exists a $\sigma \in R$ such that $u_\sigma \sqsubseteq v_\sigma$?

Even if $\mathsf{PEP}^{\mathrm{reg}}$ is to be our central problem, the base completeness result for $\mathsf{F}_{\omega^\omega}$ is on ReachLcs. The reason for this is the same that lead PCP to be shown undecidable through Turing machine termination and not the contrary. LCSs where here first. We now need to show that they are equivalent, to give $\mathsf{PEP}^{\mathrm{reg}}$ its rightful place.

In the above definition, the regular constraint applies to σ but this is inessential and our results still hold when the constraint applies to u_σ, or v_σ, or both (see Section 6.4).

For complexity issues, we assume that the constraint R in a $\mathsf{PEP}^{\mathrm{reg}}$ instance is given as a nondeterministic finite-state automaton (NFA) $\mathcal{A}_R$.

PEP is the special case of $\mathsf{PEP}^{\mathrm{reg}}$ where R is Σ^+, i.e., where there are no constraints over the form of a non-trivial solution. As far as we know, PEP and $\mathsf{PEP}^{\mathrm{reg}}$ have never been considered in the literature and this is probably because PEP is trivial : 6.2.2

reduction ideas Our journey from ReachLcs to $\mathsf{PEP}^{\mathrm{reg}}$ is not direct. Even if there is the same order limiting the exploration of solutions on both problems, LCS configuration have a natural order guiding the exploration: the transition rules. following it, searching backward always terminate on LCS. Indeed, we know that it is not needed to continue exploration when we find a sequence that is not bad. On $\mathsf{PEP}^{\mathrm{reg}}$ there seems to be no such convenient way to explore words to find solutions. The subword constraint only applies when we have the full word, where it applies at each step on an LCS. The

fact that, for a word w, $u_w \sqsubseteq v_w$ tells us nothing on words where w is a factor. The direct algorithm presented in chapter 7 tells us that the exploration instead of words, should take place on families indexed by residual of R.

We will need a few steps to tackle those differences. The First step is a small one, it is to only consider morphisms u and v such that images of letters are of size 1. The interest of this limitation is to control precisely the point where for a word $w.a$ its image embeds, i.e. $u_{w.a} \sqsubseteq v_{w.a}$ but its prefix w does not, $u_w \not\sqsubseteq v_w$. Then we will cut $\mathsf{PEP}^{\mathrm{reg}}$ solutions exactly at those points. Those sub parts are all solutions to a bit different problem, $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$. In this problem, we recover the good property from LCS runs: the $\sqsubseteq$ constraint apply also on all prefixes of solutions. We could give a direct algorithm to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$, but it is quicker to see that it is exactly $\mathsf{ReachLcs}$.

4.1 The directed Post embedding problem

Let u, v, R be a $\mathsf{PEP}^{\mathrm{reg}}$ instance and $\sigma \in R$ be a solution. We say that σ is a *direct* solution if $u_\rho \sqsubseteq v_\rho$ for every prefix ρ of σ. Hence, in a direct solution, v_ρ is always ahead of u_ρ when ρ grows from ϵ to σ.

An equivalent formulation is: $\sigma = i_1 \dots i_m$ is a direct solution iff there are words $v'_1, \dots, v'_m$ such that:
1. $v'_k \sqsubseteq v_{i_k}$ for all $k = 1, \dots, m$,
2. $u_{i_1} \dots u_{i_m} = v'_1 \dots v'_m$,
3. $|u_{i_1} \dots u_{i_k}| \leq |v'_1 \dots v'_k|$ for all $k = 1, \dots, m$.

A *codirect* solution is defined in a similar way, with the difference that we now require $|u_{i_1} \dots u_{i_k}| \geq |v'_1 \dots v'_k|$ for all $k = 1, \dots, m$ (i.e., the u_i's are ahead of the v'_i's instead of lagging behind).

We let $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ denote the questions whether a $\mathsf{PEP}^{\mathrm{reg}}$ instance has a direct (resp. codirect) solution. Obviously, $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ are equivalent problems since an instance u, v, R has a codirect solution iff the mirror instance $\widetilde{u}, \widetilde{v}, \widetilde{R}$ has a direct solution.

Let first note that:

Remark 4.1.1 $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ *and* $\mathsf{ReachLcs}$ *are equivalent.*

In fact $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ is only a reformulation of $\mathsf{ReachLcs}$ when the system has only one component and one channel. Suppose the language R of an instance $\mathcal{I} = (u, v, R)$ is given by an NFA $\mathcal{A} = (\Sigma, Q, T, q_{\mathrm{init}}, F)$ with $u, v : \Sigma^* \longrightarrow \Gamma^*$. We can define an LCS with the same structure as $\mathcal{A}$ and one canal $S = (Q, \Gamma, \Delta)$. The effects of the transitions Δ are directly given by the morphisms u and v. More formally $\Delta = \{q \xrightarrow{!v_a, ?u_a} q' | q \xrightarrow{a} q' \in T\}$. q_{init} and F comes from the reachability question on S.

A solution of x of $\mathcal{I}$ is a trace of a valid run from q_{init} to a state of F in S. The validity, i.e. that a letter cannot be consumed before being written,

is ensured by x being a direct solution; a prefix y of x correspond to the beginning of the run reaching a configuration where v_y was written to the canal and u_y read. What remains in the canal is at most (if there is no loss in that part) the "available suffix" $v_y \oslash u_y$.

In the remaining of this section we show that $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ and $\mathsf{PEP}^{\mathrm{reg}}$ are equivalent.

4.1.1 $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$

$\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$ are versions of $\mathsf{PEP}^{\mathrm{reg}}$ and $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ restricted to *short morphisms*, i.e., morphisms $u, v : \Sigma^* \to \Gamma^*$ such that $|u_i| + |v_i| \leq 1$ for all $i \in \Sigma$. In other words, for every i, at least one of u_i and v_i is ϵ and the other is either ϵ or a letter from Γ.

Their only interest is technical, it helps separate cases during following proofs.

Proposition 4.1.2 *1.* $\mathsf{PEP}^{\mathrm{reg}}$ *reduces to* $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$.
2. $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ *reduces to* $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$.

Proof.[Sketch] 1. Let u, v, R be a $\mathsf{PEP}^{\mathrm{reg}}$ instance. For all $i \in \Sigma$, write u_i in the form $a_i^1 \dots a_i^{l_i}$ and v_i in the form $b_i^1 \dots b_i^{m_i}$. Let $k = \max\{l_i, m_i \mid i \in \Sigma\}$. One builds a $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance u', v', R' by letting $\Sigma' \stackrel{\mathrm{def}}{=} \Sigma \times \{1, 2, \dots, 2k\}$, $u'(i,p) \stackrel{\mathrm{def}}{=} a_i^p$ if $1 \leq p \leq l_i$, and $u'(i,p) \stackrel{\mathrm{def}}{=} \epsilon$ otherwise. Similarly, $v'(i, k+p)$ is v_i^p, the p-th letter in v_i, when $1 \leq p \leq m_i$, and it is ϵ otherwise. Clearly u', v' are short morphisms. We now let $R' \stackrel{\mathrm{def}}{=} h(R)$ where $h : \Sigma \to \Sigma'$ is the morphism defined by $h(i) = (i,1)(i,2)\dots(i,2k)$. Finally u', v', R' is a $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance that is positive iff u, v, R is positive.

2. Exactly the same construction reduces from $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir},\leq 1}$. Reducing from $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$ can be done by simply modifying R', this time using $h(i) = (1, k+1)(i, k+2)\dots(i,2k)(i,1)\dots(i,k)$. □

4.1.2 From $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ to $\mathsf{PEP}^{\mathrm{reg}}$

Let u, v, R be a fixed $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ instance with $u, v : \Sigma^* \to \Gamma^*$. With u, v, R we associate a $\mathsf{PEP}^{\mathrm{reg}}$ instance u, v, R' with extended alphabets $\Sigma' \stackrel{\mathrm{def}}{=} \Sigma \cup \{0,1,2\}$ and $\Gamma' \stackrel{\mathrm{def}}{=} \Gamma \cup \{\#\}$, and where u, v are extended with

$$u_0 = \epsilon, \quad u_1 = \#, \quad u_2 = \#,$$
$$v_0 = \#, \quad v_1 = \#, \quad v_2 = \epsilon.$$

Finally, R' is $0(R \parallel 1^*)2 \setminus \Sigma'^* 11 \Sigma'^*$ where "$\parallel$" and "$\setminus$" denote, respectively, the shuffle product and the set difference, of two languages (two regularity-preserving operations). Intuitively, a word of R' is obtained from a word of

R by inserting 1's as long as they remain separated by Σ-letters from the original word, and wrapping with a 0 in front and a 2 at the end.

The next two lemmas show that this reduction is correct.

Lemma 4.1.3 *If u, v, R' admits a solution, then u, v, R admits a direct solution.*

Proof. Assume $x \in R'$ is a solution: $u_x \sqsubseteq v_x$. Since $R' = 0(R \parallel 1^*)2$, x can be written (uniquely) under the form $0.x_1.1.x_2 \cdots 1.x_m.2$ with $x' \stackrel{\text{def}}{=} x_1 \cdots x_m$ belonging to R. Observe that $u_x = u_{x_1}\# \cdots u_{x_m}\#$ and $v_x = \#v_{x_1} \cdots \#v_{x_m}$. Both contain exactly n occurrences of the # symbol, thus these occurrences must be matched exactly in the embedding $u_x \sqsubseteq v_x$. Hence $u_{x_1} = \epsilon$ and, for $1 \leq i < m$, $u_{x_{i+1}} \sqsubseteq v_{x_i}$. Finally x' is a direct solution (of u, v, R). □

A reciprocal of Lemma 4.1.3 holds when u and v are *short morphisms*.

Lemma 4.1.4 *If u, v are short morphisms and $x \in \Sigma^+$ is a direct solution of u, v, R, then there exists a factorization $x = x_1 \cdots x_m$ of x (with no $x_i = \epsilon$) such that $u_{x_1} = \epsilon$ and, for $1 \leq i < m$, $u_{x_{i+1}} \sqsubseteq v_{x_i}$.*

Proof. Let $x \in \Sigma^+$ be a direct solution. We define a sequence $y_0, y_1, y_2, \ldots$ of Σ-words by letting $y_0 \stackrel{\text{def}}{=} \epsilon$ and, for $i > 0$, letting y_i be the longest prefix of x such that $u_{y_i} \sqsubseteq v_{y_{i-1}}$. The sequence is well-defined and we can see, by induction on i, that every y_i is a prefix of y_{i+1}, hence y_i can be written uniquely as $y_i = y_{i-1}x_i$ and we take this as our definition of the x_i's.

We now show that $y_i \neq x$ implies $x_{i+1} \neq \epsilon$. Indeed let $a \in \Sigma$ be the letter that follows y_i in x. We know that $u_{y_i} \sqsubseteq v_{y_{i-1}}$ and $v_{y_i} = v_{y_{i-1}}v_{x_i}$. If $u_a \not\sqsubseteq v_{x_i}$ then $u_a \neq \epsilon$, hence $v_a = \epsilon$ since the morphisms are short. Finally $u_{y_i a} \not\sqsubseteq v_{y_i} = v_{y_i a}$, contradicting the assumption that x is a directed solution. We conclude that necessarily $u_a \sqsubseteq v_{x_i}$ and then a will occur in x_{i+1}. Finally, when eventually $y_{m+1} = y_m$ for some m, we deduce that $y_m = x = x_1 \ldots x_m$.

That $u_{x_1} = \epsilon$ is a consequence of $u_{y_1} \sqsubseteq v_{y_0}$.

For $i > 0$, from $u_{y_i} \sqsubseteq v_{y_{i-1}}$ and $u_{y_i}u_{x_{i+1}} = u_{y_{i+1}} \sqsubseteq v_{y_i} = v_{y_{i-1}}v_{x_i}$, we deduce that $u_{x_{i+1}} \sqsubseteq r.v_{x_i}$ for $r \stackrel{\text{def}}{=} [v_{y_i}]u_{y_{i-1}}$ (Lemma 7.1.4). But if a is the first letter of x_{i+1}, $u_a \not\sqsubseteq r$ (otherwise y_i would not be longest s.t. $u_{y_i} \sqsubseteq v_{y_{i-1}}$). Hence $u_{x_{i+1}} \sqsubseteq v_{x_i}$ (since $|u_a| \leq 1$). □

Lemma 4.1.5 *If u, v, R admits a direct solution and u, v are short morphisms, then u, v, R' admits a (direct) solution.*

Proof. Let $x \in R$ be a direct solution. Since u, v are short, x has a factorization $x = x_1 \cdots x_m$ as in Lemma 4.1.4. From $u_{x_1} = \epsilon$ and $u_{x_i} \sqsubseteq v_{x_{i-1}}$ we deduce that $0.x_1.1.x_2.1 \ldots 1.x_m.2$ is a (direct) solution in R'. □

Combining Lemmas 4.1.3 and 4.1.5, we see that in the case of short morphisms, u, v, R has a direct solution iff u, v, R' has a solution.

Corollary 4.1.6 $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$ *(and then* $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir},\leq 1}$*) reduce to* $\mathsf{PEP}^{\mathrm{reg}}$*.*

Now, since $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ reduces to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir},\leq 1}$, we conclude with:

Proposition 4.1.7 $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ *(and then* $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$*) reduce to* $\mathsf{PEP}^{\mathrm{reg}}$*.*

4.1.3 From $\mathsf{PEP}^{\mathrm{reg}}$ to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$

If we now look at a general solution to a $\mathsf{PEP}^{\mathrm{reg}}$ instance (more precisely a $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance) it can be decomposed as a succession of alternating direct and codirect solutions to sub-problems that are constrained by residuals of R. For denoting these residuals, we assume that R is given by a NFA $\mathcal{A} = (\Sigma, Q, T, q_{\mathrm{init}}, F)$ and write $L_{q,q'}$ for the regular language accepted by $\mathcal{A}$ between states q and q' (so that $R = \sum_{q' \in F} L_{q_{\mathrm{init}},q'}$).

Assume that u, v, R is a $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance and $\sigma = i_1 \dots i_m$ is a solution. Then there are words $v'_1, \dots, v'_m$ with $v'_k \sqsubseteq v_{i_k}$ for $k = 1, \dots, m$, and such that $u_{i_1} \dots u_{i_m} = v'_1 \dots v'_m$. Now, for $0 \leq k \leq m$, define $d_k \stackrel{\mathrm{def}}{=} |u_{i_1} \dots u_{i_k}| - |v'_1 \dots v'_k|$. Thus, for ρ a length-k prefix of σ, d_k measures how much u_ρ is ahead of v_ρ (assuming a fixed embedding of u_σ into v_σ given by the v'_i's).

Since σ is a solution, obviously $d_0 = d_m = 0$. σ is a direct solution if $d_k \leq 0$ for all k. It is codirect if $d_k \geq 0$ for all k. In general, d_k may oscillate between positive and negative values. But since all u_i's and v_i's have length ≤ 1, the difference $d_{k+1} - d_k$ is in $\{-1, 0, 1\}$. Hence d_k cannot change sign without being zero.
In conclusion, the following holds:

Lemma 4.1.8 *A* $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ *instance* u, v, R *is positive iff there are states* q_0*,* q_1*, ...,* q_{2m} *in* $\mathcal{A}$ *with* $q_0 = q_{\mathrm{init}}$*,* $q_{2m} \in F$*, and such that, for all* $0 \leq i < m$*,* $u, v, L_{q_{2i},q_{2i+1}}$ *is a positive* $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ *instance and* $u, v, L_{q_{2i+1},q_{2i+2}}$ *is a positive* $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ *instance.*

Now, if $\mathcal{A}$ has m states, there are only m^2 $L_{q,q'}$ residuals, hence the $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance can be reduced to a positive Boolean combination ϕ of a quadratic number of $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ instances (the $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{codir}}$ instances are turned into $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ instances by taking their mirror images).

Boolean combinations

With $\mathcal{B}(\mathsf{PEP}^{\mathrm{reg}})$ we denote the problem of solving Boolean combinations of PEP instances. An instance of $\mathcal{B}(\mathsf{PEP}^{\mathrm{reg}})$ is some $\langle \mathcal{I}_1, \dots, \mathcal{I}_n, \phi(x_1, \dots, x_n) \rangle$ where each $\mathcal{I}_i$ is some $\mathsf{PEP}^{\mathrm{reg}}$ instance u, v, R, and where $\phi : \{0,1\}^n \to \{0,1\}$ is a Boolean function with free variables in $\{x_1, \dots, x_n\}$. The instance is positive iff ϕ evaluates to 1 when each x_i is replaced by 0 or 1 depending on whether $\mathcal{I}_i$ is negative or positive.

$\mathcal{B}^+(\mathsf{PEP}^{\mathrm{reg}})$ is the restriction of $\mathcal{B}(\mathsf{PEP}^{\mathrm{reg}})$ where ϕ is a *positive* Boolean function, while $\mathcal{B}(\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}})$ and $\mathcal{B}^+(\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}})$ are the same problems based on directed instances.

Proposition 4.1.9 *$\mathcal{B}^+(\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}})$ and $\mathcal{B}^+(\mathsf{PEP}^{\mathrm{reg}})$ reduce to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ (and to $\mathsf{PEP}^{\mathrm{reg}}$).*

Proof.[Sketch] Assume $\mathcal{I} = (u, v, R)$ and $\mathcal{I}' = (u', v', R')$ are two $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ instances. We can ensure that they use disjoint alphabets, using renamings if necessary. Then the disjunction $\mathcal{I} \vee \mathcal{I}'$ is equivalent to $\langle u+u', v+v', R \cup R' \rangle$ while the conjunction $\mathcal{I} \wedge \mathcal{I}'$ is equivalent to $\langle u+u', v+v', R.R' \rangle$. This many-one reduction also works for $\mathsf{PEP}^{\mathrm{reg}}$. It extends directly to any positive ϕ and can be made polynomial-space when ϕ is given under the form of a positive Boolean circuit and the regular constraints R_i as NFAs. □

Now, with proposition 4.1.7, 4.1.8 and remark 4.1.1 this concludes the proof of:

Theorem 4.1.10 *$\mathsf{PEP}^{\mathrm{reg}}$ and $\mathsf{ReachLcs}$ are equivalents.*

Chapter 5

Generalised channel systems

This chapter present the first problem we studied, the classification of *mixed* channel systems (with reliable and lossy channels) according to their computationnal power. That study leaded to the definition of $\mathsf{PEP}^{\mathrm{reg}}$. It turns out that only two classes of topologies of systems remained, one equivalent to LCS, and the other to CS.

5.1 Systems with reliable and lossy channels

We classify channel systems according to their *network topology*, which is a graph describing who are the participant processes and what channels they are connected to.

5.1.1 Network topologies

Formally, a *network topology*, or shortly a *topology*, is a tuple $T = \langle N, R, L, s, d\rangle$ where N, R and L are three mutually disjoint finite sets of, respectively, *nodes*, *reliable channels*, and *lossy channels*, and where, writing $C \stackrel{\text{def}}{=} R \cup L$ for the set of channels, $s, d : C \to N$ are two mappings that associate a *source* and a *destination* node to each channel. We do not distinguish between isomorphic topologies since N, R and L simply contain "names" for nodes and channels: these are irrelevant here and only the directed graph structure with two types of edges matters.

Graphical examples of simple topologies will be found below: we use dashed arrows to single out the lossy channels (reliable channels are depicted with full arrows).

5.1.2 Mixed channel systems and their operational semantics

Assume $T = \langle N, R, L, s, d\rangle$ is a topology with n nodes, i.e., with $N = \{P_1, P_2, ..., P_n\}$. Write $C = R \cup L$ for the set of channels. A *mixed channel system* (MCS) having topology T is a tuple $S = \langle T, \mathtt{M}, Q_1, \Delta_1, ..., Q_n, \Delta_n\rangle$

where $\mathtt{M} = \{\mathtt{a}, \mathtt{b}, ...\}$ is a finite *message alphabet* and where, for $i = 1, ..., n$, Q_i is the finite set of (control) states of a process (also denoted P_i) that will be located at node $P_i \in N$, and Δ_i is the finite set of *transition rules*, or shortly "rules", governing the behaviour of P_i. A rule $\delta \in \Delta_i$ is either a *writing rule* of the form $(q, c, !, \mathtt{a}, q')$, usually denoted "$q \xrightarrow{c!\mathtt{a}} q'$", with $q, q' \in Q_i$, $s(c) = P_i$ and $\mathtt{a} \in \mathtt{M}$, or it is a *reading rule* $(q, c, ?, \mathtt{a}, q')$, usually denoted "$q \xrightarrow{c?\mathtt{a}} q'$", with this time $d(c) = P_i$. Hence the way a topology T is respected by a channel system is via restrictions upon the set of channels to which a given participant may read from, or write to.

Our terminology "*mixed* channel system" is meant to emphasize the fact that we allow systems where lossy channels coexist with reliable channels.

The behaviour of some $S = \langle T, \mathtt{M}, Q_1, \Delta_1, ..., Q_n, \Delta_n \rangle$ is given under the form of a transition system. Assume $C = \{c_1, ..., c_k\}$ contains k channels. A configuration of S is a tuple $\sigma = \langle q_1, ..., q_n, u_i, ..., u_k \rangle$ where, for $i = 1, ..., n$, $q_i \in Q_i$ is the current state of P_i, and where, for $i = 1, ..., k$, $u_i \in \mathtt{M}^*$ is the current contents of channel c_i.

Assume $\sigma = \langle q_1, ..., q_n, u_i, ..., u_k \rangle$ and $\sigma' = \langle q'_1, ..., q'_n, u'_i, ..., u'_k \rangle$ are two configurations of some system S as above, and $\delta \in \Delta_i$ is a rule of participant P_i. Then δ witnesses a transition between σ and σ', also called a *step*, and denoted $\sigma \xrightarrow{\delta} \sigma'$, if and only if

- the control states agree with, and are modified according to δ, i.e., $q_i = q$, $q'_i = q'$, $q_j = q'_j$ for all $j \neq i$;
- the channel contents agree with, and are modified according to δ, i.e., either
  - $\delta = (q, c_l, ?, \mathtt{a}, q')$ is a reading rule, and $u_l = \mathtt{a}.u'_l$, or
  - $\delta = (q, c_l, !, \mathtt{a}, q')$ is a writing rule, and $u'_l = u_l.\mathtt{a}$, or $c_l \in L$ is a lossy channel and $u'_l = u_l$;

  in both cases, the other channels are untouched: $u'_j = u_j$ for all $j \neq l$.

Such a step is called "*a step by* P_i" and we say that its *effect* is "reading $\mathtt{a}$ on c", or "writing $\mathtt{a}$ to c", or "losing $\mathtt{a}$". A *run* (from σ_0 to σ_p) is a sequence of steps of the form $r = \sigma_0 \xrightarrow{\delta_1} \sigma_1 \xrightarrow{\delta_2} \sigma_2 \cdots \xrightarrow{\delta_p} \sigma_p$, sometimes shortly written $\sigma_0 \xrightarrow{*} \sigma_p$. A run is *perfect* if none of its steps loses a message.

Remark 5.1.1 *This semantics is* write-lossy. *In this chapter, where we only consider reachability problems, the semantic doesn't matter, but this one makes some proofs significantly simpler.*

5.1.3 The reachability problem for network topologies

The *reachability problem* for mixed channel systems asks, for a given S and two configurations $\sigma_{\text{init}} = \langle q_1, \ldots, q_n, \epsilon, \ldots, \epsilon \rangle$ and $\sigma_{\text{final}} = \langle q'_1, \ldots, q'_n, \epsilon, \ldots, \epsilon \rangle$ in which the channels are empty, whether S has a run from σ_{init} to σ_{final}. That we restrict reachability questions to configurations with empty channels (ϵ denotes the empty word in $\mathtt{M}^*$) is technically convenient, but it is no real loss of generality.

The *reachability problem* for a topology T is the restriction of the reachability problem to mixed systems having topology T.

Clearly, if T' is a subgraph of T and reachability is decidable for T, then it is for T' too.

In this chapter, our goal is to determine for which topologies reachability is decidable. Let us illustrate the question with T_1^{ring} a topology describing a directed ring of processes, where each participant sends to its right-hand neighbour, and receives from its left-hand neighbour. A folk claim is that such cyclic networks have decidable reachability as soon as one channel is lossy (as here with c_2). The proof ideas behind this claim have not been formally published and they do not easily adapt to related questions like "what about T_2^{ring}?", where a lossy channel in the other direction is added, or about T_3^{ring} where more channels are lossy in the ring.

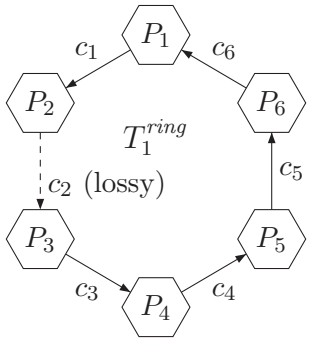


Figure 5.1: Unidirectional ring topology with a lossy channel

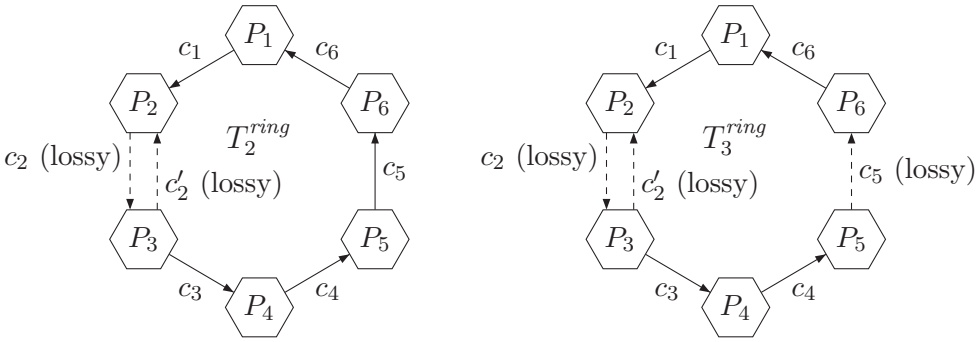


Figure 5.2: Ring-like network topologies

Our techniques answer all three questions uniformly. One of our results states that all channels along the path c_3 to c_4 to c_5 to c_6 to c_1 can be fused into a single channel going from P_3 to P_2 without affecting the decidability of reachability. The transformations are modular (we fuse one channel at a time). Depending on the starting topology, we end up with different two-node topologies, from which we deduce that T_1^{ring} and T_3^{ring} have decidable

reachability, while T_2^{ring} does not (see Corollary 5.3.6 below).

5.2 Reachability for basic topologies

This section is concerned with the basic topologies to which we will later reduce all larger cases.

5.2.1 Unidirectional Channel Systems

We start by introducing Unidirectional Channel Systems (UCS) an important topology, closely related to $\mathsf{PEP}^{\mathrm{reg}}$, the same way $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ is to LCS's. It is the topology described by figure 5.3. We will show that ReachUcs, the reachability problem in systems having an UCS topology, is decidable and equivalent to $\mathsf{PEP}^{\mathrm{reg}}$. To this end, we will introduce $\mathsf{2PCEP}^{\mathrm{reg}}$, an intermediate problem between ReachUcs and $\mathsf{PEP}^{\mathrm{reg}}$.

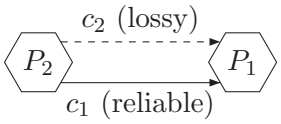


Figure 5.3: UCS

Definition 5.2.1 ($\mathsf{2PCEP}^{\mathbf{reg}}$)
a. *The* 2-dimensional correspondence plus embedding problem *asks, given two pairs of morphisms* $f_1, g_1 : \Sigma_1^* \to \Gamma^*$ *and* $f_2, g_2 : \Sigma_2^* \to \Gamma^*$*, to find words* σ_1 *and* σ_2 *s.t.* $f_1(\sigma_1) = f_2(\sigma_2)$ *(correspondence) and* $g_1(\sigma_1) \sqsubseteq g_2(\sigma_2)$ *(embedding).*
b. $\mathsf{2PCEP}^{reg}$ *is the decision problem, where given* f_1, g_1, f_2, g_2 *and two regular languages* $R_1 \subseteq \Sigma_1^*$ *and* $R_2 \subseteq \Sigma_2^*$*, one asks whether there is a solution with* $\sigma_1 \in R_1$ *and* $\sigma_2 \in R_2$*.*

A solution (σ_1, σ_2) of an instance $\mathsf{2PCEP}^{\mathrm{reg}}$ is essentially a run on an UCS, where σ_1 is the trace of the P_1 (reading) automaton and σ_2 of the P_2 (writing) automaton. We can see the given functions f_1, f_2, g_1, g_2 as projections from the labeling of steps to what is written or read by that step. For instance $f_2(\sigma_2)$ is what writes P_2 to the channel c_1 (reliable), it is projecting $q \xrightarrow{!_1 a} q'$ to a, and the steps that don't write are projected to ϵ. The same way g_2 correspond to what is written on c_2 (lossy), f_1 to reads on c_1 and g_1 to reads on c_2. The constraints $f_1(\sigma_1) = f_2(\sigma_2)$ ensures that the communication is perfect. What is read correspond to what is written. The constraints $g_1(\sigma_1) \sqsubseteq g_2(\sigma_2)$ ensures that what is read have been written, but don't ensure that every message reach P_1.

Lemma 5.2.2 ReachUcs *and* $\mathsf{2PCEP}^{reg}$ *are equivalent.*

The complete proof of this lemma is available in section6.5.1, where an infinitary version is also proved.

2PCEP$^{\text{reg}}$ and PEP$^{\text{reg}}$ are equivalent.

Lemma 5.2.3 2PCEP^{reg} *and* PEP$^{\text{reg}}$ *are equivalent.*

Proof. We consider a 2PCEP$^{\text{reg}}$ instance f_1, g_1, f_2, g_2 where we assume that the morphisms are short, i.e., f_i and g_i can be seen as having type $(\Sigma_i \cup \{\epsilon\}) \to (\Gamma \cup \{\epsilon\})$. Thanks to the possibility offered by the regular constraints, this assumption is no loss of generality.
Let $\Sigma \stackrel{\text{def}}{=} (\Sigma_1 \cup \{\epsilon\}) \times (\Sigma_2 \cup \{\epsilon\})$ and define $X \subseteq \Sigma$ by

$$(i, j) \in X \text{ if and only if } f_1(i) = f_2(j).$$

Then $(i_1, j_1).(i_2, j_2) \ldots (i_n, j_n) \in X^*$ implies that $f_1(i_1.i_2 \ldots i_n) = f_2(j_1.j_2 \ldots j_n)$. Reciprocally, if $f_1(\sigma_1) = f_2(\sigma_2)$, then σ_1 and σ_2 can be decomposed under the form $\sigma_1 = i_1.i_2 \ldots i_n$ and $\sigma_2 = j_1.j_2 \ldots j_n$ such that $(i_k, j_k) \in X$ for $k = 1, \ldots, n$. Observe that in this decomposition, $n \geq |\sigma_i|$ is possible since $i_k = \epsilon$ or $j_k = \epsilon$ (or both) is allowed.

Now define projection morphisms $h_1 : \Sigma^* \to \Sigma_1^*$ and $h_2 : \Sigma^* \to \Sigma_2^*$ in the obvious way, and let $u, v : \Sigma^* \to \Gamma^*$ be two morphisms given by $u \stackrel{\text{def}}{=} g_1 \circ h_1$ and $v \stackrel{\text{def}}{=} g_2 \circ h_2$. Then $u_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)} \sqsubseteq v_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)}$ if and only if $g_1(i_1.i_2 \ldots i_n) \sqsubseteq g_2(j_1.j_2 \ldots j_n)$.

Finally, the 2PCEP$^{\text{reg}}$ instance with regular constraints R_1, R_2 translates into an equivalent PEP$^{\text{reg}}$ instance, with morphisms u and v as above, and with constraint

$$R \stackrel{\text{def}}{=} X^* \cap {h_1}^{-1}(R_1) \cap {h_2}^{-1}(R_2),$$

which is regular.

Obviously the other direction holds since an instance of PEP$^{\text{reg}}$ it is an instance of 2PCEP$^{\text{reg}}$ having $f_1 = f_2 = Id$. □

Corollary 5.2.4 PEP$^{\text{reg}}$ *and* ReachUcs *are equivalent.*

5.2.2 Other basic topologies

Theorem 5.2.5 (Basic topologies) *Reachability is decidable for the network topologies T_1^d and T_2^d (see Fig. 5.4). It is not decidable for the topologies T_1^u, T_2^u, T_3^u, T_4^u, T_5^u, and T_6^u (see Fig. 5.5).*

We start with the decidable cases:

That T_1^d, and more generally all topologies with only lossy channels (aka LCS's), leads to decidable problems is the classic result from [AJ96b].
Now to the undecidable cases:

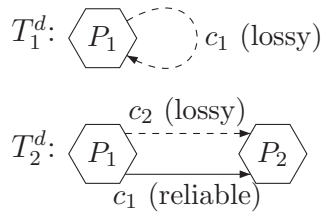


Figure 5.4: Basic decidable topologies

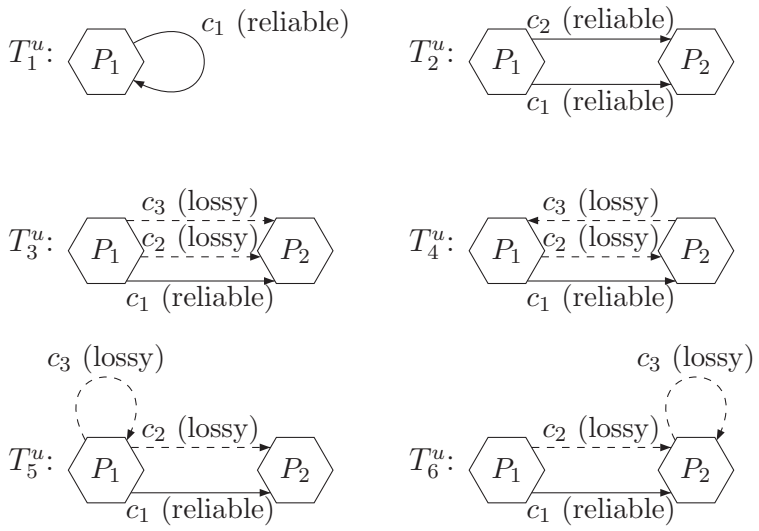


Figure 5.5: Basic topologies with undecidable reachability

It is well-known that T_1^u may lead to undecidable problems [BZ83], and this is also known, though less well, for T_2^u (restated, e.g., as the non-emptiness problem for the intersection of two rational transductions). The other four results mix lossy and reliable channels and are new. We actually prove all six cases in a uniform framework, by reduction from Post's Correspondence Problem, aka PCP, or its directed variant, $\mathsf{PCP}_{\mathrm{dir}}$.

Recall that an instance of PCP is a family $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$ of $2n$ words over some alphabet. The question is whether there is a non-empty sequence (a *solution*) $i_1, \ldots, i_k$ of indexes such that $x_{i_1}x_{i_2}\ldots x_{i_k} = y_{i_1}y_{i_2}\ldots y_{i_k}$. $\mathsf{PCP}_{\mathrm{dir}}$ asks whether there is a *directed* solution $i_1, \ldots, i_k$, i.e., a solution such that, in addition, $y_{i_1}y_{i_2}\ldots y_{i_h}$ is a prefix of $x_{i_1}x_{i_2}\ldots x_{i_h}$ for all $h = 1, \ldots, k$. It is well-known that PCP and $\mathsf{PCP}_{\mathrm{dir}}$ are undecidable, and more precisely Σ_0^1-complete.

Reducing PCP to T_2^u-networks. With a PCP instance $(x_i, y_i)_{i=1,\ldots,n}$, we associate a process P_1 having a single state p_1 and n loops[1] $p_1 \xrightarrow{c_1!x_i \;\; c_2!y_i} p_1$, one for each index $i = 1, ..., n$. Process P_1 guesses a solution $i_1 i_2 i_3 \ldots$ and sends the concatenations $x_{i_1}x_{i_2}x_{i_3}\ldots$ and $y_{i_1}y_{i_2}y_{i_3}\ldots$ on, respectively, c_1 and c_2. Process P_2 checks that the two channels c_1 and c_2 have the same contents, using reading loops $p_2 \xrightarrow{c_1?\mathsf{a} \;\; c_2?\mathsf{a}} p_2$, one for each symbol $\mathsf{a}, \mathsf{b}, \ldots$ in the alphabet. An extra control state, for example p_1' with rules $p_1' \xrightarrow{c_1!x_i \;\; c_2!y_i} p_1$, is

[1]Transition rules like "$p_1 \xrightarrow{c_1!x_i \;\; c_2!y_i} p_1$" above, where several reads and writes are combined in a same rule, and where one writes or reads words rather than just one message at a time, are standard short-hand notations for sequences of rules using intermediary states that are left implicit. We avoid using this notation in situations where the specific ordering of the combined actions is important as, e.g., in $(*)$ below.

required to check that P_1 picks a non-empty solution. Then, in the resulting T_2^u-network, $\langle p_1', p_2, \epsilon, \epsilon\rangle \xrightarrow{*} \langle p_1, p_2, \epsilon, \epsilon\rangle$ if and only if the PCP instance has a solution.

Reducing PCP to T_3^u-networks. For T_3^u, the same idea is adapted to a situation with three channels, two of which are lossy. Here P_1 has rules $p_1 \xrightarrow{c_2!x_i \;\; c_3!y_i \;\; c_1!1^{|x_i y_i|}} p_1$. Thus P_1 sends x_i and y_i on lossy channels and simultaneously sends the number of letters in unary (1 is a special tally symbol) on c_1, the perfect channel. P_2 matches these with reading loops of the form $p_2 \xrightarrow{c_1?1 \;\; c_2?\mathsf{a} \;\; c_3?\mathsf{a}} p_2$ for each letter a. If P_2 can consume all 1's out of c_1, this means that no message has been lost on the lossy channels, and then P_2 really witnessed a solution the PCP instance.

Reducing $\mathsf{PCP}_{\mathrm{dir}}$ to T_1^u-networks. For T_1^u, we consider the directed $\mathsf{PCP}_{\mathrm{dir}}$. P_1 has n loops $p_1 \xrightarrow{c_1!x_i \;\; c_1?y_i} p_1$ where the guessing and the matching is done by a single process. Since at any step $h = 1, ..., k$ the concatenation $x_{i_1}x_{i_2}...x_{i_h}$ is (partly) consumed while matching for $y_{i_1}y_{i_2}...y_{i_h}$, only directed solutions will be accepted.

Reducing $\mathsf{PCP}_{\mathrm{dir}}$ to T_5^u-networks. For T_5^u too, we start from $\mathsf{PCP}_{\mathrm{dir}}$ and use a variant of the previous counting mechanism to detect whether some messages have been lost. P_1 has rules of the form $p_1 \xrightarrow{c_3!1^{|x_i|} \;\; c_1!x_i \;\; c_3?1^{|y_i|} \;\; c_2!y_i} p_1$, i.e., it sends x_i on c_1 (the reliable channel) and y_i on c_2 (unreliable) while P_2 checks the match with loops $p_2 \xrightarrow{c_1?\mathsf{a} \;\; c_2?\mathsf{a}} p_2$. In addition, P_1 also maintains in c_3 a count of the number of symbols written to c_1 minus the number of symbols written to c_2, or $\#_h \stackrel{\text{def}}{=} |x_{i_1} \dots x_{i_h}| - |y_{i_1} \dots y_{i_h}|$. The counting scheme forbids partial sequences $y_{i_1} \dots y_{i_h}$ that would be longer than the corresponding $x_{i_1} \dots x_{i_h}$, but this is right since we look for directed solutions. If tally symbols on c_3 are lost, or if part of the y_i's on c_2 are lost, then it will never be possible for P_2 to consume all messages from c_1. Finally a run from $\langle p_1', p_2, \epsilon, \epsilon, \epsilon\rangle$ to $\langle p_1, p_2, \epsilon, \epsilon, \epsilon\rangle$ must be perfect and witness a directed solution.

Reducing $\mathsf{PCP}_{\mathrm{dir}}$ to T_6^u-networks. For T_6^u, we adapt the same idea, this time having P_2 monitoring the count $\#_h$ on c_3. P_1 has loops $p_1 \xrightarrow{c_1!x_i 1^{|y_i|} \;\; c_2!y_i 1^{|x_i|}} p_1$ where a guessed solution is sent on c_1 and c_2 with interspersed tally symbols. The guessed solution is checked with the usual loops $p_2 \xrightarrow{c_1?\mathsf{a} \;\; c_2?\mathsf{a}} p_2$. The 1's on c_2 are stored to c_3 and matched (later) with the 1's on c_1 via two loops: $p_2 \xrightarrow{c_2?1 \;\; c_3!1} p_2$ and $p_2 \xrightarrow{c_3?1 \;\; c_1?1} p_2$. In a perfect run, there are always as many messages on c_1 as there are on c_2 and c_3 together, and strictly more if a message is lost. Hence a run from $\langle p_1', p_2, \epsilon, \epsilon, \epsilon\rangle$ to $\langle p_1, p_2, \epsilon, \epsilon, \epsilon\rangle$ must be

perfect and witness a solution. Only direct solutions can be accepted since the tally symbols in c_3 count $\#_h$ that cannot be negative.

Reducing $\mathsf{PCP}_{\text{dir}}$ to T_4^u-networks. For T_4^u, we further adapt the idea, again with the count $\#_h$ stored on c_3 but now sent from P_2 to P_1. The loops in P_1 now are

$$p_1 \xrightarrow{c_1!x_i \;\; c_2!y_i 1^{|x_i|}} q_i \xrightarrow{c_3?1^{|y_i|}} p_1. \qquad (*)$$

The 1's on c_2 are sent back via c_3 to be matched later by P_1, thanks to a loop $p_2 \xrightarrow{c_2?1 \;\; c_3!1} p_2$. Again a message loss will leave strictly more messages in c_1 than in c_2 and c_3 together, and cannot be recovered from. Only direct solutions can be accepted since the tally symbols in c_3 count $\#_h$.

5.3 Fusion for essential channels

Sections 5.3 and 5.4 develop techniques for "simplifying" topologies while preserving the decidability status of reachability problems. We start with a reduction called "fusion".

Let $T = \langle N, R, L, s, d\rangle$ be a network topology. For any channel $c \in C$, $T - c$ denotes the topology obtained from T by deleting c. For any two distinct nodes $P_1, P_2 \in N$, $T[P_1 = P_2]$ denotes the topology obtained from T by merging P_1 and P_2 in the obvious way: channel extremities are redirected accordingly.

Clearly, any MCS with topology $T - c$ can be seen as having topology T. Thus $T - c$ has decidable reachability when T has, but the converse is not true in general.

Similarly, any MCS having topology T can be transformed into an equivalent MCS having topology $T[P_1 = P_2]$ (using the asynchronous product of two control automata). Thus T has decidable reachability when $T[P_1 = P_2]$ has, but the converse is not true in general.

For any channel c such that $s(c) \neq d(c)$, we let T/c denote $T[s(c) = d(c)] - c$ and say that T/c is "*obtained from T by contracting c*". Hence T/c is obtained by merging c's source and destination, and then removing c.

Since T/c is obtained via a combination of merging and channel removal, there is, in general, no connection between the decidability of reachability for T and for T/c. However, there is a strong connection for so-called "essential" channels, as stated in Theorem 5.3.5 below.

Before we can get to that point, we need to explain what are essential channels and how they can be used.

5.3.1 Essential channels are existentially 1-bounded

In this section, we assume a given MCS $S = \langle T, \mathtt{M}, Q_1, \Delta_1, \ldots \rangle$ with $T = \langle N, R, L, s, d \rangle$.

Definition 5.3.1 *A channel $c \in C$ is* essential *if $s(c) \neq d(c)$ and all directed paths from $s(c)$ to $d(c)$ in T go through c.*

In other words, removing c modifies the connectivity of the directed graph underlying T.

The crucial feature of an essential channel c is that causality between the actions of $s(c)$ and the actions of $d(c)$ is constrained. As a consequence, it is always possible to reorder the actions in a run so that reading from c occurs immediately after the corresponding writing to c. As a consequence, bounding the number of messages that can be stored in c does not really restrict the system behaviour.

Formally, for $b \in \mathbb{N}$, we say a channel c is b-bounded along a run $\pi = \sigma_0 \xrightarrow{\delta_1} \ldots \xrightarrow{\delta_n} \sigma_n$ if $|\sigma_i(c)| \leq b$ for $i = 0, \ldots, n$. We say c is *synchronous* in π if it is 1-bounded and at least one of $\sigma_i(c)$ and $\sigma_{i+1}(c)$ is ϵ for all $0 \leq i < n$. Hence a synchronous channel only stores at most one message at a time, and the message is read immediately after it has been written to c.

Proposition 5.3.2 *If c is essential and $\pi = \sigma_0 \xrightarrow{\delta_1} \ldots \xrightarrow{\delta_n} \sigma_n$ is a run with $\sigma_0(c) = \sigma_n(c) = \epsilon$, then S has a run π' from σ_0 to σ_n in which c is synchronous.*

This notion is similar to the existentially-bounded systems of [LM04] but is applies to a single channel, not to the whole system.

We prove Proposition 5.3.2 using techniques and concepts from true concurrency theory and message flow graphs (see, e.g., [HMK+05]). With a run $\pi = \sigma_0 \xrightarrow{\delta_1} \ldots \xrightarrow{\delta_n} \sigma_n$ as above, we associate a set $E = \{1, \ldots, n\}$ of n events, that can be thought of the actions performed by the n steps of π: firing a transition and reading or writing or losing a message. Observe that different occurrences of a same transition with same effect are two different events. We simply identify the events with indexes from 1 to n. We write $e, e', \ldots$ to denote events, and also use the letters r and w for reading and writing events.

Any $e \in E$ is an event of some process $N(e) \in N$ and we write $E = \bigcup_{P \in N} E_P$ the corresponding partition. There exist several (standard) causality relations between events. For every process $P \in N$, the events of P are linearly ordered by $<_P$: $i <_P j$ iff $i, j \in E_P$ and $i < j$. For every channel $c \in C$, the events that write to or read from c are related by $<_c$ with $i <_c j$ iff i is an event that writes some m to c, and j is the event that reads that (occurrence of) m. (Here, events that lose messages are considered as internal actions where no channel is involved.) We let $\prec$ (and $\preceq$) denote

the transitive (resp. reflexive-transitive) closure of $\bigcup_{P\in N} <_P \cup \bigcup_{c\in C} <_c$. $(E, \preceq)$ is then a poset, and $\preceq$ is called the *visual* order (also causality order, or dependency order) in the literature. For $e \in E$, we let $\downarrow e$ denote the past of e, i.e., the set $\{e' \in E \mid e' \preceq e\}$.

It is well-known that any linear extension $e_1, \ldots, e_n$ of $(E, \preceq)$ is causally consistent and can be transformed into a run $\pi' = \sigma_0 \xrightarrow{e_1} \xrightarrow{e_2} \cdots$ starting from σ_0. This run ends in σ_n like π, though it may go through different intermediary configurations. All the runs obtained by considering different linear extensions are causally equivalent to π, denoted $\pi \approx \pi'$, and they all give rise to the same poset $(E, \preceq)$.

We now state properties enjoyed by $(E, \preceq)$ in our context that are useful for proving Proposition 5.3.2. First, observe that, since the channels are FIFO, and since only one process, namely $d(c)$ (resp. $s(c)$), is allowed to read from (resp. write to) a channel c:

$$(w_1 <_c r_1 \text{ and } w_2 <_c r_2) \text{ imply } (w_1 <_{s(c)} w_2 \text{ iff } r_1 <_{d(c)} r_2). \qquad (\dagger)$$

$(\dagger)$ is sometimes taken as a definition of FIFO communication.

Another important observation is the following: assume $e \preceq e'$. Then, and since $\preceq$ is defined as a reflexive-transitive closure, there must be a chain of the form

$$\theta : \quad e = e_0 \leq_{P_0} e'_0 <_{c_1} e_1 \leq_{P_1} e'_1 <_{c_2} \ldots <_{c_l} e_l \leq_{P_l} e'_l = e'$$

where, for $1 \leq i \leq l$, $s(c_i) = P_{i-1}$ and $d(c_i) = P_i$. Hence T has a path $c_1, \ldots, c_l$ going from P_0 to P_l.

Lemma 5.3.3 *If $e_1 \prec e_2 \prec e_3$ and c is essential, then $e_1 \not<_c e_3$.*

Proof. By contradiction. Assume $e_1 \prec e_2 \prec e_3$ and $e_1 <_c e_3$ for an essential c. Since all paths from $P = N(e_1) = s(c)$ to $P' = N(e_3) = d(c)$ go through c (by essentiality), there must exist a pair $w, r \in E$ with $e_1 \preceq w <_c r \preceq e_2$ or, symmetrically, $e_2 \preceq w <_c r \preceq e_3$, depending on whether the $w <_c r$ pair occurs before or after e_2 in the chain from e_1 to e_2 to e_3. If $e_1 \preceq w <_c r \preceq e_2 \prec e_3$, then $r <_{P'} e_3$, hence $w <_P e_1$ using $(\dagger)$. If $e_1 \prec e_2 \preceq w <_c r \preceq e_3$, then $e_1 <_P w$, hence $e_3 <_{P'} r$ using $(\dagger)$. In both cases we obtain a contradiction. □

We now assume that c is essential and that π has $\sigma_0(c) = \sigma_n(c) = \epsilon$ (hence E has the same number, say m, of events reading from c and writing to it). Write P for $s(c)$ and P' for $d(c)$. Let $w_1 <_P w_2 \ldots <_P w_m$ be the m events that write to c, listed in causal order. Let $r_1 <_{P'} e_2 \ldots <_{P'} r_m$ be the m events that read from c listed in causal order.

Lemma 5.3.4 *There exists a linear extension of $(E, \preceq)$ where, for $i = 1, \ldots, m$, w_i occurs just before r_i.*

Proof. The linear extension is constructed incrementally. Formally, for $i = 1, \ldots, m$, let $E_i \stackrel{\text{def}}{=} {\downarrow} r_i$ and $F_i \stackrel{\text{def}}{=} E_i \setminus \{w_i, r_i\}$. Observe that $F_1 \subsetneq E_1 \subseteq F_2 \cdots F_i \subsetneq E_i \subseteq F_{i+1}$, with the convention that $F_{m+1} = E$. Every E_i is a $\preceq$-closed subset of E, also called a down-cut of $(E, \preceq)$. Furthermore, F_i is a down-cut of E_i by Lemma 5.3.3. Hence a linear extension of F_i followed by $w_i.r_i$ gives a linear extension of E_i, and following it with a linear extension of $F_{i+1} \setminus E_i$ gives a linear extension of F_{i+1}. Any linear extension of $F_{i+1} \setminus E_i$ can be chosen since this subset does not contain reads from, or writes to, c. □

The linear extension we just built gives rise to a run π' in which c is synchronous. This concludes the proof of Proposition 5.3.2.

Observe that when several channels are essential in T, it is in general not possible to replace a run π with an equivalent π' where all essential channels are simultaneously synchronous.

5.3.2 Decidability by fusion

We call "*fusion*" the transformation of T to T/c where c is essential, and "*reliable fusion*" the special case where c is also a reliable channel.

Theorem 5.3.5 (Decidability by fusion) *Let c be an essential channel in T:*
1. T has decidable reachability if T/c has.
2. If c is a reliable channel, then T/c has decidable reachability if T has.

Proof. 1. Let S be a T-MCS. We replace it by a system S' where c has been removed and where the processes at nodes $P_1 = s(c)$ and $P_2 = d(c)$ have been replaced by a larger process that simulate both P_1 and P_2 and where communication along c is replaced by synchronizing the sends in P_1 with the reads in P_2 (message losses are simulated even more simply by the P_1 part). S' has topology T/c and simulates S *restricted to runs where c is synchronous*. By Proposition 5.3.2, this is sufficient to reach any reachable configuration. Since reachability in S' is decidable, we conclude that reachability in S is decidable.

2. We now also assume that c is reliable and consider a (T/c)-MCS S. With S we associate a T-MCS S' that simulates S. S' has two nodes P_1 and P_2 where S only had a merged P node.

The construction is illustrated in Fig. 5.6. Informally, P_1 inherits states from P and all rules that read from channels c_1 with $d(c_1) = P_1$ in T, or write to channels c_2 with $s(c_2) = P_1$. Regarding the other rules, the communication action (reading from some c_3 or writing to some c_4) is sent

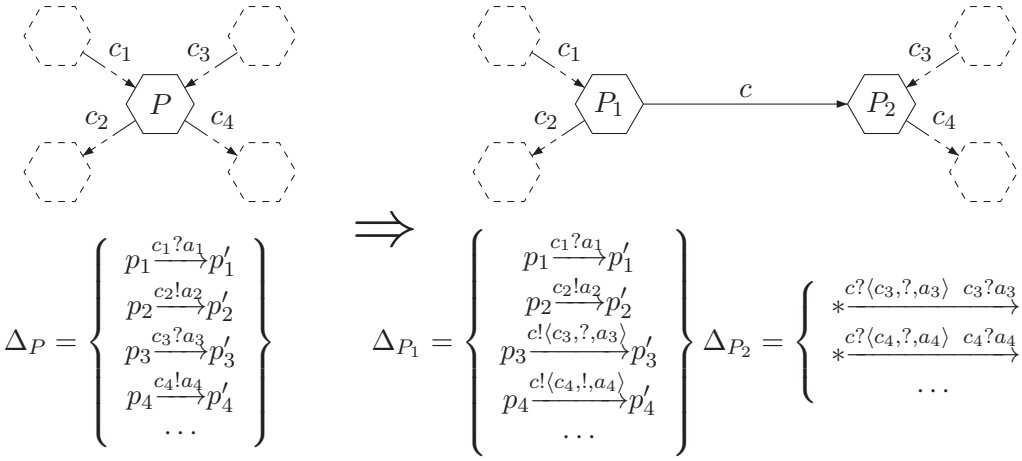


Figure 5.6: Associating a T-MCS with a T/c-MCS

to P_2 via c. S' uses an extended alphabet $\mathtt{M}'$ that extends the message alphabet $\mathtt{M}$ from S via $\mathtt{M}' \stackrel{\text{def}}{=} \mathtt{M} \cup (C \times \{?,!\} \times \mathtt{M})$. P_2 only has simple loops around a central state $*$ that read communication instructions from P_1 via c and carry them out.

S' simulates S in a strong way. Any step in S can be simulated in S', perhaps by two consecutive steps if a communication operation has to transit from P_1 to P_2 via c. In the other direction, there are some runs in S' that cannot be simulated directly by S, e.g., when P_2 does not carry out the instructions sent by P_1 (or carries them out with a delay). But all runs in S' in which c is synchronous are simulated by S.

Since runs in which c is synchronous are sufficient to reach any configuration reachable in S' (Proposition 5.3.2), the two-way simulation reduces reachability in S to reachability in S', which is decidable if T has decidable reachability. □

The usefulness of Theorem 5.3.5 is illustrated by the following two corollaries.

Corollary 5.3.6 *T_1^{ring} and T_3^{ring} (from Section 5.1.1) have decidable reachability. T_2^{ring} does not.*

Proof. Building $T_1^{ring}/c_3/c_4/c_5/c_6/c_1$ only fuses essential channels and ends up with a decidable topology (only lossy channels).
Starting with T_2^{ring}, we can build $T = T_2^{ring}/c_3/c_4/c_5/c_6$ but have to stop there (c_1 is not essential). The resulting T, isomorphic to T_4^u from Fig. 5.5, does not have decidable reachability. Hence T_2^{ring} does not have decidable reachability since we fused reliable channels only.
With T_3^{ring}, it is better to build $T_3^{ring}/c_3/c_4/c_6/c_1$. Here too we cannot fuse any more because of c_2', but the end result is a topology with decidable

reachability since c_5 is lossy. Hence T_3^{ring} has decidable reachability. □

Corollary 5.3.7 *A topology in the form of an undirected forest has decidable reachability.*

Proof.[Sketch] If T is a forest, every channel c is essential, and every T/c is still a forest. Hence T reduces to a topology with lossy channels only. □

5.4 Splitting along lossy channels

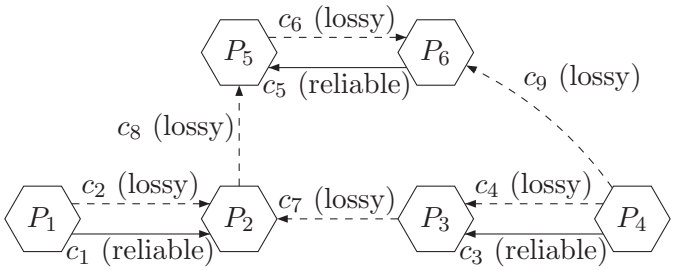


Figure 5.7: A topology that splits in three

Let $T_1 = \langle N_1, R_1, L_1, s_1, d_1\rangle$ and $T_2 = \langle N_2, R_2, L_2, s_2, d_2\rangle$ be two disjoint topologies. We say that $T = \langle N, R, L, s, d\rangle$ is a *(lossy) gluing of* T_1 *on* T_2 if T is a juxtaposition of T_1 and T_2 (hence $N = N_1 \cup N_2$) with an additional set L_3 of lossy channels (hence $R = R_1 \cup R_2$ and $L = L_1 \cup L_2 \cup L_3$) connecting from T_1 to T_2 in a unidirectional way: $s(L_3) \subseteq N_1$ and $d(L_3) \subseteq N_2$.

This situation is written informally "$T = T_1 \rhd T_2$", omitting details on L_3 and its connections. In practice this notion is used to split a large T into subparts rather than build larger topologies out of T_1 and T_2.

Theorem 5.4.1 (Decidability by splitting) *Reachability is decidable for* $T_1 \rhd T2$ *if, and only if, it is for both* T_1 *and* T_2.

The proof of Theorem 5.4.1 (see Appendix 5.8.1) uses techniques that are standard for LCS's but that have to be adapted to the more general setting of MCS's.

We can apply Theorem 5.4.1 to prove that the topology in Fig. 5.7 has decidable reachability. Indeed, this topology can be split along lossy channels (first $\{c_8, c_9\}$, then c_7), giving rise to two copies of T_2^d (from Fig. 5.4) and a two-node ring that can be reduced to T_1^d by fusion.

5.5 A complete classification

In this section, we prove that the results from the previous sections provide a complete classification.

Theorem 5.5.1 (Completeness) *A network topology T has decidable reachability if, and only if, it can be reduced to T_2^d (from Fig. 5.4) and LCS's using fusion and splitting only.*[2]

Note that, via splitting, the reduction above usually transforms T into *several* topologies. All of them must be T_2^d or LCS's for T to have decidable reachability.

The "$\Leftarrow$" direction is immediate in view of Theorems 5.3.5.1 and 5.4.1,

For the "$\Rightarrow$" direction, we can assume w.l.o.g. that T is *reduced*, i.e., it cannot be split as some $T_1 \triangleright T_2$, and it does not contain any reliable essential channel (that could be fused).

We now assume, by way of contradiction, that T cannot be transformed, via general fusions, to T_2^d or to a LCS. From this we show that reachability is not decidable for T. When showing this, we sometimes mention three additional transformations ("simplification", "doubling of loops" and "non-essential fusion") that are described in Appendix 5.8.2. We now start an involved case analysis.

1. Since T cannot be transformed to a LCS, it contains a reliable channel c_r, linking node $A = s(c_r)$ to node $B = d(c_r)$. We can assume $A \neq B$, otherwise T contains T_1^u (from Fig. 5.5) and we conclude immediately with undecidability.

2. T must contain a path θ of the form $A = P_0, c_1, P_1, c_2, \ldots, c_n, P_n = B$ that links A to B without using c_r, otherwise c_r would be essential, contradicting the assumption that T is reduced. We pick the shortest such θ (it is a simple path) and we call T' the subgraph of T that only contains θ, c_r, and the nodes to which they connect.

3. If all c_i's along θ are reliable, T' can be transformed to T_2^u (from Fig. 5.5) by reliable fusions, hence T', and then T itself, have undecidable reachability. Therefore we can assume that at least one c_i along θ is lossy.

4. Assume that there exist two nodes P_i, P_j along θ that are connected via a third path θ' disjoint from c_r and θ. We put no restrictions on the relative positions of P_i and P_j but we assume that θ' is not a trivial empty path if $i = j$. In that case, let T'' be the subgraph of T that contains c_r, θ, and θ', and where all channels except c_r are downgraded to lossy if they were reliable. Using simplification and doubling of lossy loops, T'' can be transformed to an undecidable topology among $\{T_3^u, T_4^u, T_5^u, T_6^u\}$. Hence T''

[2] As is well-known, it is possible to further reduce any LCS into T_1^d. However, we preferred a statement for Theorem 5.5.1 where only our two main transformations are involved.

does not have decidable reachability. Neither has T since taking subgraphs and downgrading channels can only improve decidability.

5. If we are not in case 4, the nodes along θ do not admit a third path like θ'. Therefore all channels along θ must be lossy, since we assumed T is reduced. Thus T' can be transformed to T_2^d by general fusion. Since we assumed T cannot be transformed to T_2^d, T must contain extra nodes or channels beyond those of T'. In particular, this must include extra nodes since we just assumed that T has no third path θ' between the T' nodes. Furthermore these extra nodes must be connected to the T' part otherwise splitting T would be possible. There are now several cases.

6. We first consider the case of an extra node C with a reliable channel c from C to T'. Since T is reduced, c is not essential and there must be a second path θ' from C to T'. Call T'' the subgraph of T that only contains T', C, c and θ'. Applying non-essential fusion on c, θ' becomes a path between some P_i, P_j and we are back to case 4. Hence undecidability.

7. Next is the case of an extra node C with a reliable channel c from T' to C. Again, since c is not essential, there must be a second path θ' from T' to C. Again, the induced subgraph T'' can be shown undecidable as in case 6, reducing to case 4.

8. If there is no extra node linked to T' via a reliable c, the extra nodes must be linked to T' via lossy channels. Now the connection must go both ways, otherwise splitting would be possible. The simplest case is an extra node C with a lossy c from C to T' and a lossy c' from T' to C. But this would have been covered in case 4.

9. Finally there must be at least two extra nodes C and C', with a lossy channel c from C to T' and a lossy c' from T' to C'. We can assume that all paths between T' and C, C' go through c and c', otherwise we would be in one of the cases we already considered. Furthermore C and C' must be connected otherwise T could be split. There are several possibilities here.

10. If there is a path from C' to C we are back to case 4. Hence undecidability.

11. Thus all paths connecting C and C' go from C to C'. If one such path is made of reliable channels only, reliable fusion can be applied on the induced subgraph, merging C and C' and leading to case 8 where undecidability has been shown. If they all contain one lossy channel, T can be split, contradicting our assumption. that it is reduced.

We have now covered all possibilities when T is reduced but cannot be transformed to a LCS or to T_2^d. In all cases is has been shown that reachability is not decidable for T. This concludes the proof of Theorem 5.5.1.

5.6 A classification algorithm

Theorem 5.6.1 (Polynomial-time classification) *There exists a polynomial-time algorithm that classifies topologies according to whether they have decidable reachability.*

The algorithm relies on Theorem 5.5.1:

Stage 1: Starting from a topology T, apply splitting and *reliable* fusion as much as possible. When several transformations are possible, pick any of them nondeterministically. At any step, the transformation reduces the size of the topologies at hand, hence termination is guaranteed in a linear number of steps. At this stage we preserved decidability in both directions, hence T has decidability iff all the reduced topologies $T_1, \ldots, T_n$ have.

Stage 2: Each T_i is now simplified using general fusion (not just reliable fusion). If this ends with a LCS or with T_2^d, decidability for T_i has been proved. When fusion can be applied in several ways, we pick one nondeterministically: a consequence of Theorem 5.5.1's proof is that these choices lead to the same conclusion when starting from a system that cannot be reduced with splitting or reliable fusion. Thus stage 2 terminates in a linear number of steps. When it terminates, either every T_i has been transformed into a LCS or T_2^d, and we conclude that reachability is decidable for T, or one T_i remains unsimplified and we conclude that reachability is not decidable for T.

We observe that when stage 1 finishes, there will never be any new opportunity for reliable fusion or for splitting since stage 2, i.e., general fusion, does not create or destroy any path between nodes.

5.7 Concluding remarks

Summary. We introduced *mixed channel systems*, i.e., FIFO channel systems where both lossy and reliable channels can be combined in arbitrary topologies. These systems are a generalization of the lossy channel system model (where all channels are lossy and where reachability is decidable) and of the standard model (with unbounded reliable FIFO channels, where reachability is undecidable).

For mixed systems, we provide a complete classification of the network topologies according to whether they lead to decidable reachability problems or not. Our main tool are reductions methods that transform a topology into simpler topologies with an equivalent decidability status. These reductions produce small basic topologies for which the decidability status is established in Section 5.2.

Directions for future work. The two main avenues for future work are extending the MCS model (e.g., by considering other kinds of unreliability in the style of [CFP96], or by allowing guards in the style of [BBS06], etc.) and considering questions beyond just reachability and safety (e.g., termination and liveness).

5.8 Appendix

5.8.1 Proofs for Section 5.4

This section proves Theorem 5.4.1, i.e., "$T_1 \triangleright T_2$ has decidable reachability iff T_1 and T_2 have", where $T_1 \triangleright T_2$ is a juxtaposition of T_1 and T_2 with additional glue in the form of lossy channels with source in T_1 and destination in T_2.

First observe that the "$\Rightarrow$" direction is immediate since T_1 and T_2 are subgraphs of T.

For the "$\Leftarrow$" direction, we assume $T = T_1 \triangleright T_2$ with T, T_1 and T_2 as in Section 5.4. We consider a MCS S with topology T. From S we extract two subsystems S_1 and S_2 with topologies T_1' and T_2' that are slight augmentations of T_1 and T_2. More precisely, T_1' is T_1 augmented with the interface channels $c_1, \ldots, c_k$ from L_3, and with dummy extra processes $D_1, \ldots, D_k$, one for each $c_i \in L_3$, so that $d(c_i) = D_i$ is not left undefined. T_2' is T_2 augmented in a similar way, this time with $s(c_i) = D_i$. The MCS's S_1 and S_2 are the restrictions of S to T_1' and T_2' assuming that the extra processes $D_1, \ldots, D_k$ are inactive.

Observe that, for $i = 1, 2$, the channels in L_3 are essential in T_i' (also note that T_i' is in general not a subgraph of T since different interface channels in L_3 may share a common source or a common destination). Since applying fusion on L_3-channels gives exactly T_i, and since we assumed reachability is decidable for T_i, we conclude it is for T_i' too by Theorem 5.3.5.

We now show how to decide reachability for S assuming that reachability is decidable for topologies T_1' and T_2', hence for MCS's S_1 and S_2.

A configuration σ of S can be written under the form $\langle \sigma^1, \sigma^2, u_1, \ldots, u_k \rangle$ where σ^1 is the restriction of σ to T_1, σ^2 is the restriction to T_2, and $u_1, \ldots, u_k$ are the contents of the extra channels from L_3. (In particular, the contents of channels in $R_i \cup L_i$ are part of σ^i).

Lemma 5.8.1 *Let $\sigma_{\text{init}} = \langle \sigma^1_{\text{init}}, \sigma^2_{\text{init}}, \epsilon, \ldots, \epsilon \rangle$ and $\sigma_{\text{final}} = \langle \sigma^1_{\text{final}}, \sigma^2_{\text{final}}, \epsilon, \ldots, \epsilon \rangle$ be two configurations of S with empty channels. There is a run $\sigma_{\text{init}} \xrightarrow{*} \sigma_{\text{final}}$ in S if, and only if, there is a tuple $\langle u_1, \ldots, u_k \rangle$ such that S_1 has a run $\langle \sigma^1_{\text{init}}, \epsilon, \ldots, \epsilon \rangle \xrightarrow{*} \langle \sigma^1_{\text{final}}, u_1, \ldots, u_k \rangle$ and S_2 has a run $\langle \sigma^2_{\text{init}}, u_1, \ldots, u_k \rangle \xrightarrow{*} \langle \sigma^2_{\text{final}}, \epsilon, \ldots, \epsilon \rangle$.*

Proof.[Sketch] Indeed, since the steps in the S_1 part of S never depend on the steps in the S_2 part (interface channels in L_3 only go from S_1 to S_2), it is always possible to use all the S_1 steps first and the S_2 steps later. □

Lemma 5.8.2 *The following problems are decidable:*
(1) Given some $\langle u_1, \ldots, u_k \rangle \in (\mathtt{M}^*)^k$, does S_1 have a run $\langle \sigma^1_{\text{init}}, \epsilon, \ldots, \epsilon \rangle \xrightarrow{*} \langle \sigma^1_{\text{final}}, u_1, \ldots, u_k \rangle$?

(2) Given some $\langle u_1, \ldots, u_k \rangle \in (\mathtt{M}^*)^k$*, does* S_2 *have a run* $\langle \sigma^2_{\text{init}}, u_1, \ldots, u_k \rangle \xrightarrow{*} \langle \sigma^2_{\text{final}}, \epsilon, \ldots, \epsilon \rangle$*?*
(3) Given some regular languages $R_1, \ldots, R_k \subseteq \mathtt{M}^*$*, does there exists a tuple* $\langle u_1, \ldots, u_k \rangle \in R_1 \times \cdots \times R_k$ *such that* S_2 *has a run* $\langle \sigma^2_{\text{init}}, u_1, \ldots, u_k \rangle \xrightarrow{*} \langle \sigma^2_{\text{final}}, \epsilon, \ldots, \epsilon \rangle$*?*

Proof. (1) is almost immediate since reachability is decidable in T_1'. Since we insist on asking reachability questions with empty channels in the initial and final configurations, we have to program the extra components $D_1, \ldots, D_k$ so that they empty the c_i and check that they contained u_i and only accept if this is the case. The resulting system is still a T_1' system.

For (2), the same idea applies but this time the D_i's fill the interface channels with the u_i. Ensuring that u_i is really inserted in c_i is done by upgrading the interface channels from lossy to reliable channels. This does not impact the decidability of reachability since it is established by fusing essential channels and reducing to T_2.

For (3) we program the D_i's so that they nondeterministically write one $u_i \in R_i$ in c_i. Since R_i is regular, a finite-state D_i can do the generation. Hence we reduced (3) to a reachability question on a decidable topology (T_2' with reliable interface channels). □

Lemma 5.8.3 *The set* $R \subseteq (\mathtt{M}^*)^k$ *of all minimal (w.r.t. the subword ordering) tuples* $\langle u_1, \ldots, u_k \rangle$ *allowing* $\langle \sigma^2_{\text{init}}, u_1, \ldots, u_k \rangle \xrightarrow{*} \langle \sigma^2_{\text{final}}, \epsilon, \ldots, \epsilon \rangle$ *is finite and can be computed effectively.*

Proof. R is finite since the subword ordering is a well-quasi-order (Higman's Lemma).

Regarding its computation, we cannot apply the backward reachability algorithm for LCS's since T_2' may contain reliable channels. However, by Lemma 5.8.2.(2), we can check any candidate tuple. Therefore it is possible to build R incrementally by enumerating all candidate tuples. Enumerating them in order of increasing length ensures that only minimal tuples are retained.

This procedure is bound to eventually build R (since it is finite) and there only remains to ensure termination by detecting when the current R is complete. This can be done using Lemma 5.8.2.(3): the set R' of all tuples that do not contain a tuple from R as subword is a regular language, being the complement of the upward-closure of a finite set. Thus we can decide whether R' contains some tuple that is not yet accounted for in R. One detail is that R', though regular, is not in general a product $R'_1 \times \cdots \times R'_k$ of regular languages, one for each part of the tuple. However it is well-known that such sets are a finite union $\sum_i R'_{1,i} \times \cdots \times R'_{k,i}$ of products of regular languages. □

We now have enough tools to implement Lemma 5.8.1 and thereby decide reachability for S. We compute R and check, using Lemma 5.8.2.1, that one of the tuples in R is reachable with S_1. Observe that restricting to minimal tuples does not invalidate the algorithm: $c_1, \ldots, c_k$ being lossy, the set of tuples that S_1 can write there is downward-closed.

5.8.2 Some additional transformations

This section describes additional transformations and how they preserve decidability of reachability. The correctness proofs are only sketched in this extended abstract, but the missing parts are easy to fill in since the transformations are similar to existing ones.

We list these transformations for the sake of completeness (they are used in the proof of Theorem 5.5.1) but the reader should understand that they do not occur in the classification algorithm, or in the statement of the classification theorem, where only essential fusion and splitting are needed.

1. Double lossy loops. We say that T has a *double lossy loop* if there are distinct $c, c' \in L$ with $s(c) = d(c) = s(c') = d(c')$.

Lemma 5.8.4 *If c and c' are a double lossy loop in T then reachability is decidable for T if, and only if, it is for $T - c'$.*

Proof.[Idea] A single loop can simulate two loops the way a single lossy loop can simulate an arbitrary LCS: we concatenate the contents of the two original channels in the remaining one, using special markers to separate the two contents (see, e.g., [Sch02, Section 5]). Acting on one part of the contents requires rotating the contents of the channels, and this can be achieved with the help of the markers. The markers are inserted at the start of the run, and removed at the end. If they are lost during the simulation, correct simulation cannot be guaranteed, but it will be impossible to reach an accepting state. Hence the simulation is correct for reachability questions. The new observation is that it remains correct with an arbitrary mixed topology around the two loops under consideration. □

Remark 5.8.5 *Paradoxically, we do not use Lemma 5.8.4 for simplifying systems. Rather we use it for doubling loops, which may prove useful when we try to obtain basic topologies from Fig. 5.5 via simplification (see below). Hence it is important that Lemma 5.8.4 preserves decidability in both directions.*

2. Simplification. Let T be a topology with a lossy channel system c between two nodes P_1 and P_2. The *simplification of T by c* is a topology

T' where c has been removed and where all channels c' with $s(c') = P_2$ in T are redirected and have $s(c') = P_1$ in T'.

Lemma 5.8.6 *Reachability is decidable for T' if it is for T.*

Proof.[Idea] T' misses many features of T, which only improves decidability. The features of T' that T misses are the channels c' from P_1 to some P that go from P_2 to P in T. In T, these can be simulated by a standard multiplexing trick going through P_2 via c. □

3. Non-essential fusion. Let c be a reliable channel from P_1 to P_2 ($P_1 \neq P_2$) in some topology T. Assume that there is an additional path from P_1 to P_2 that does not use c (hence c is not essential). Further assume that this path only contains lossy channels, and that there is no other path from P_1 to P_2.

Lemma 5.8.7 *Reachability is decidable for T/c if it is for T.*

Proving Lemma 5.8.7 is quite different from proving Theorem 5.3.5. It uses the same simulation we use in section 5.2.1 to link T_2^d and T_1^d, but this time in a more general context since extra channels and processes may occur in T.

Part II

More on PEP

Chapter 6

PEP variants

In this chapter, we introduce multiples PEP variants. In section 6.2 are the versions with no regular constraints. Some are trivial (LogSpace), but some infinite version are PTime-complete. Section 6.3 has the non trivial infinitary versions, both decidable and undecidable. And in section 6.4 we will present versions with differences on the constraints. When we use a regular constraint on different words than the input, we always have the same problem. This is mainly a justification of our definition of $\mathsf{PEP}^{\mathrm{reg}}$. But if we use constraints stronger than regulars, we have undecidable problems.

6.1 Definitions

6.1.1 Infinitary version of PEP, PEP^{ω}

$u \sqsubseteq v$ when there exists an order-preserving injective map $h : \{1, \dots, n\} \to \{1, \dots, m\}$ such that $a_i = b_{h(i)}$ for all $i = 1, \dots, n$. Embeddings between ω-words are defined similarly, with a strictly increasing $h : \mathbb{N} \setminus 0 \to \mathbb{N} \setminus 0$. We explicitly allow the embedding of finite words into infinite ones.

Then using that version of embedding $\mathsf{PEP}^{\omega\text{-reg}}$ is just $\mathsf{PEP}^{\mathrm{reg}}$ where the constraint language is chosen ω-regular.

Problem $\mathsf{PEP}^{\omega\text{-reg}}$

Instance: Two finite alphabets Σ and Γ, two morphisms $u, v : \Sigma^* \to \Gamma^*$, and an ω-regular language $R \subseteq \Sigma^\omega$.

Question: Does there exists a $\sigma \in R$ such that $u_\sigma \sqsubseteq v_\sigma$?

PEP^{ω} is the special case where R is Σ^ω.

Similarly to the finite case, we say that σ is a *direct* solution if $u_\rho \sqsubseteq v_\rho$ for every prefix ρ of σ. It is a *codirect solution* if $u_\rho \sqsubseteq v_\rho$ for every suffix ρ of σ.

The problem $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ ($\mathsf{PEP}^{\omega\text{-reg}}_{\text{codir}}$) asks furthermore that the solution is direct (resp. codirect).

Note that in the finite case, the difference between directness and codirectness was meaningless since a codirect solution is just a direct solution of the mirror instance. It doesn't hold in the infinite case, the prefix of a solution being a finite word and the suffix an infinite one.

6.2 Too simple cases

We thinks that the following variants are responsible for embedding problems never being studied. Their solutions are too easy to be interesting.

In this section, we will state that Σ and Γ are two alphabets and $u, v : \Sigma^* \to \Gamma^*$ are two morphisms defining a Post embedding problem.

6.2.1 PEP, $\mathsf{PEP}_{\text{dir}}$, $\mathsf{PEP}_{\text{codir}}$ and $\mathsf{PEP}^{\omega}_{\text{dir}}$

Fact 6.2.1 (proof in appendix A.1)
1. If $xy \sqsubseteq z$, then there exists a factorization $z = z_1 z_2$ of z such that $x \sqsubseteq z_1$ and $y \sqsubseteq z_2$.
2. If $x \sqsubseteq yz$, then there exists a factorization $x = x_1 x_2$ of x such that $x_1 \sqsubseteq y$ and $x_2 \sqsubseteq z$.

Corollary 6.2.2 *There is a $\sigma \in \Sigma^+$ such that $u_\sigma \sqsubseteq v_\sigma$ if and only if there is some $i \in \Sigma$ such that $u_i \sqsubseteq v_i$.*

For $\mathsf{PEP}_{\text{dir}}$, $\mathsf{PEP}_{\text{codir}}$ and $\mathsf{PEP}^{\omega}_{\text{dir}}$, this is even simpler. By definition, the first prefix (suffix) embeds. So those case are all LogSpace.

6.2.2 PEP^{ω} and $\mathsf{PEP}^{\omega}_{\text{codir}}$

Observe that, between ω-words, embedding is only a (partial) *quasi*-ordering: $u \sqsubseteq v$ and $v \sqsubseteq u$ together do not imply $u = v$. For example, $(ab)^\omega \sqsubseteq (bba)^\omega \sqsubseteq (ab)^\omega$. We write $u \equiv v$ when $u \sqsubseteq v$ and $v \sqsubseteq u$.

Halving ω-words. For some $u \in \Sigma^\omega$, let $\mathit{inf}(u) \subseteq \Sigma$ denote the set of letters that occur infinitely many times in u. The word u can be decomposed under the form $u'.u''$ where u' is a finite prefix and the corresponding suffix $u'' \in \Sigma^\omega$, only contains letters from $\mathit{inf}(u)$. Such a decomposition is called a *halving* of u. There exists several (in fact, infinitely many) halvings of any $u \in \Sigma^\omega$: the *canonical halving* is obtained by selecting the shortest possible prefix u'.

For some $u \in \Sigma^\omega$ or $u \in \Sigma^*$ the set $\mathit{alph}(u)$ is the set of letters (a subset of Σ) that occur in u.

The following lemma is a classic tool when considering embeddings between ω-words (see, e.g., [Fin85]).

Lemma 6.2.3 *Let $u, v \in \Sigma^\omega$ be two ω-words with $u'.u''$ and $v'.v''$ two arbitrary halvings of u and v. Then*

$$u \sqsubseteq v \text{ iff } \begin{cases} alph(u'') \subseteq alph(v''), \text{ and} \\ \text{there exists } x \in alph(v'')^* \text{ such that } u' \sqsubseteq v'x. \end{cases}$$

Furthermore, when $u \sqsubseteq v$, then x can be chosen with $|x| \leq |u'|$, and for any halving $u = u'.u''$ there exists a halving $v = v'.v''$ such that $u' \sqsubseteq v'$.

Corollary 6.2.4 *Let u_1, u_2 be two ω-words such that $inf(u_1) = alph(u_1) = alph(u_2) = inf(u_2)$. Then $u.u_1 \equiv u.u_2$ for all $u \in \Sigma^*$.*

Proposition 6.2.5 *There is an ω-solution in Σ^ω if and only if there is there is a codirect ω-solution if and only if there exists a non-empty subset Σ' of Σ s.t. $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$.*

Proof. Obviously, if $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$ for some non-empty $\Sigma' = \{i_1, \ldots, i_m\}$, then $(i_1 \ldots i_m)^\omega$ is an ω-solution, and even a codirect one. Conversely, given an ω-solution σ, Lemma 6.2.3 entails that, letting $\Sigma' \stackrel{\text{def}}{=} inf(\sigma)$, one has $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$. □

From here we know that PEP^ω and $\mathsf{PEP}^\omega_{\text{codir}}$ are the same problems. We will then show that it is PTime-complete.

PTime-hardness

We reduce CircuitValue to PEP^ω. Let $\mathcal{C} = (G_\vee, G_\wedge, G_\top, G_\bot, f_1, f_2, n_0)$ be an instance of CircuitValue, as illustrated in Fig 6.1. We assume, without loss of

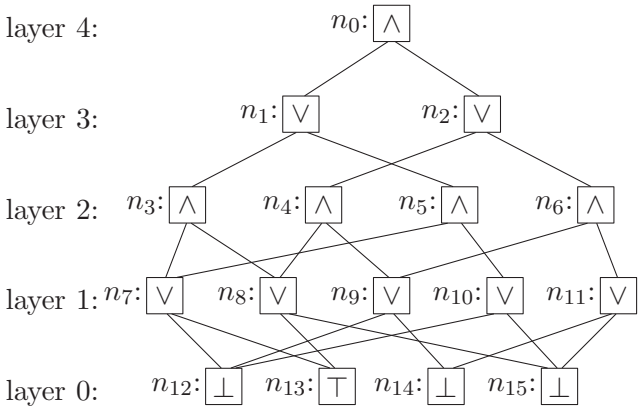


Figure 6.1: An instance of CircuitValue.

generality [GHR95, problem A.1.6], that gates are arranged in layers, that layer 0 contains "constants" gates from $G_\top \cup G_\bot$, that, for any, $k \in \mathbb{N}$ layer

$2k+1$ (resp. $2k+2$) contains OR-gates (resp. AND-gates) from $G_\vee$ (resp. $G_\wedge$), that any gate n in some layer $k>0$ has exactly two inputs, $f_1(n)$ and $f_2(n)$, that belong to layer $k-1$ (NB: $f_1(n)=f_2(n)$ is allowed). Finally, we assume that the output n_0 of $\mathcal{C}$ belongs to $G_\wedge$.

Given a circuit $\mathcal{C}$, we define in the obvious way the value $val(n)\in\{0,1\}$ of gate $n\in G$, where $G \stackrel{\text{def}}{=} G_\vee \cup G_\wedge \cup G_\top \cup G_\bot$ is the set of gates. Let $G_{=1} \stackrel{\text{def}}{=} \{n \in G \mid val(n)=1\}$. In our example, $G_{=1}=\{n_1,n_3,n_7,n_8,n_{13}\}$.

With $\mathcal{C}$ we associate two morphisms $u,v:\Sigma^*\to\Gamma^*$ as follows. Let $\Sigma \stackrel{\text{def}}{=} G_\wedge \cup (G_\vee\times\{1,2\}) \cup G_\top$ and $\Gamma \stackrel{\text{def}}{=} G$.

$$u(n) \stackrel{\text{def}}{=} f_1(n).f_2(n).n_0 \qquad v(n) \stackrel{\text{def}}{=} n \qquad \text{for } n\in G_\wedge, \tag{C1}$$

$$u(n,i) \stackrel{\text{def}}{=} f_i(n).n_0 \qquad v(n,i) \stackrel{\text{def}}{=} n \qquad \text{for } n\in G_\vee\times\{1,2\}, \tag{C2}$$

$$u(n) \stackrel{\text{def}}{=} n_0 \qquad v(n) \stackrel{\text{def}}{=} n \qquad \text{for } n\in G_\top. \tag{C3}$$

The reduction is clearly LogSpace. Its correctness is established by the following two lemmas.

Lemma 6.2.6 *If $val(n_0)=1$, then there is a non-empty Σ' with $alph(u(\Sigma'))\subseteq alph(v(\Sigma'))$.*

Proof. Let

$$\Sigma' \stackrel{\text{def}}{=} \begin{array}{ll} & \{n\in G_\wedge\cup G_\top \mid val(n)=1\} \\ \cup & \{(n,i)\in G_\vee\times\{1,2\} \mid val(f_i(n))=1\}. \end{array}$$

Σ' is not empty since it contains n_0. Observe that $alph(v(\Sigma'))$ is exactly $G_{=1}$. It remains to check, by inspecting (C1–3), that $x\in\Sigma'$ implies $alph(u(x))\subseteq G_{=1}$. □

Lemma 6.2.7 *Assume that $alph(u(\Sigma'))\subseteq alph(v(\Sigma'))$ for some non-empty $\Sigma'\subseteq\Sigma$. Then $val(n_0)=1$.*

Proof. Since necessarily n_0 appears in $alph(u(\Sigma'))$, hence in $alph(v(\Sigma'))$, it is enough to show that $alph(v(\Sigma'))\subseteq G_{=1}$. We do this by induction on layers. Let $x\in\Sigma'$ and consider three cases. If $x\in G_\top$, then $x\in G_{=1}$ obviously. If $x\in G_\wedge$, then $alph(u(x))\subseteq alph(v(\Sigma'))$ implies that both $f_1(x)$ and $f_2(x)$ belong to $alph(v(\Sigma'))$, hence evaluate to 1 by ind. hyp., so that $val(x)=1$. Finally, if x is some $(n,i)\in G_\vee\times\{1,2\}$, then from $f_i(n)=u(x)\in alph(v(\Sigma'))$, we deduce that $f_i(n)\in G_{=1}$ by ind. hyp., hence $val(n)=1$, proving $v(x)\in G_{=1}$. □

Theorem 6.2.8 PEP^ω *and* $\mathsf{PEP}^\omega_{codir}$ *coincide, and are* PTime*-complete.*

Proof. The previous lemmas showed the hardness part. There exists a simple polynomial-time decision procedure for PEP^{ω}. It computes the largest Σ' satisfying $alph(u(\Sigma')) \subseteq alph(v(\Sigma'))$ and then checks that this Σ' is not empty. This largest Σ' is obtained by starting with $\Sigma' := \Sigma$ and then removing from Σ' every i for which $alph(u_i)$ is not included in the current Σ', until eventual stabilization. □

6.3 Non trivial infinite PEP

6.3.1 $\mathsf{PEP}^{\omega\text{-reg}}$ and $\mathsf{PEP}^{\omega\text{-reg}}_{\mathbf{codir}}$

Theorem 6.3.1 $\mathsf{PEP}^{\omega\text{-reg}}$ *and* $\mathsf{PEP}^{\mathrm{reg}}$ *are equivalent (modulo elementary reductions).*

Corollary 6.3.2 $\mathsf{PEP}^{\omega\text{-reg}}$ *is* $\mathsf{F}_{\omega^\omega}$*-complete.*

An application of this result is to explore the link between channel systems and PEP back.

RecReachUcs, the *recurrent reachability problem for UCS's*, is the question whether the system S, having an UCS (def. at section 5.2.1) topology, has an infinite run $\langle q_{\mathrm{init}}, q'_{\mathrm{init}}, \epsilon, \epsilon\rangle \to \langle q_1, q'_1, v_1, v'_1\rangle \to \langle q_2, q'_2, v_2, v'_2\rangle \to \cdots$ with $q_k, q'_k \in F$ for infinitely many $k \in \mathbb{N}$.

Lemma 6.3.3 $\mathsf{PEP}^{\omega\text{-reg}}$ *and* RecReachUcs *are equivalent.*

This is essentially the same idea as $\mathsf{PEP}^{\mathrm{reg}}$ equivalent to ReachUcs, the main difference is the use of Büchi automata instead of FSA.

Corollary 6.3.4 RecReachUcs *is* $\mathsf{F}_{\omega^\omega}$*-complete.*

proof of $\mathsf{PEP}^{\omega\text{-reg}}$ and $\mathsf{PEP}^{\mathrm{reg}}$ equivalence

One direction of Theorem 6.3.1 is obvious: any $\mathsf{PEP}^{\mathrm{reg}}$ instance u, v, R can be seen as a $\mathsf{PEP}^{\omega\text{-reg}}$ instance by adding an extra symbol $\perp$ to Σ and Γ, replacing R with $R.\perp^{\omega}$, and letting $u(\perp) = v(\perp) = \perp$.

For the other direction, we consider a $\mathsf{PEP}^{\omega\text{-reg}}$ instance given by two morphisms $u, v : \Sigma^* \to \Gamma^*$ and an ω-regular language $R \subseteq \Sigma^{\omega}$.

Lemma 6.3.5 *There exists $\sigma \in R$ such that $u_\sigma \sqsubseteq v_\sigma$ if and only if there exists two finite words ρ_1 and ρ_2 in Σ^* such that*

(a) $\rho_1.\rho_2^{\omega} \in R$,

(b) $u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2}$, *and*

(c) $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$.

Proof. The "$\Leftarrow$" direction is easy since taking $\sigma = \rho_1.\rho_2^\omega$ is sufficient. For the "$\Rightarrow$" direction, we assume that $\sigma = a_1a_2a_3\ldots \in R$ satisfies $u_\sigma \sqsubseteq v_\sigma$ and show how to build ρ_1 and ρ_2.

Let $\mathcal{A}_R = (Q, \Sigma, q_0, F, \delta)$ be a Büchi automaton for R, and $\pi = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$ be an accepting run of $\mathcal{A}_R$ over σ. This run is an ω-sequence of transitions "$q_{i-1} \xrightarrow{a_i} q_i$", so that $\pi \in \delta^\omega$ can be halved under the form $\pi = \pi'.\pi''$. This gives rise to two halvings $u'.u''$ and $v'.v''$ of, respectively, u_σ and v_σ.

Let us pick a finite prefix θ of π'' that uses every transition from $\mathit{inf}(\pi)$ at least once, and that ends on the starting state of π''. Hence θ is some $q_n \xrightarrow{a_{n+1}} q_{n+1} \xrightarrow{a_{n+2}} \cdots \xrightarrow{a_{n+k}} q_{n+k}$ with $n = |\pi'|$, $q_n = q_{n+k}$, and $\mathit{inf}(\sigma) = \{a_{n+1}, a_{n+2}, \ldots, a_{n+k}\}$. Let now $\rho_1 \stackrel{\text{def}}{=} a_1a_2\ldots a_n$ and $\rho \stackrel{\text{def}}{=} a_{n+1}a_{n+2}\ldots a_{n+k}$. Clearly $\rho_1.\rho^\omega \in R$ as witnessed by the ultimately periodic run $\pi'.\theta^\omega$. Furthermore, from $u' = u_{\rho_1}$ and $\mathit{inf}(u'') = \mathit{alph}(u'') = \mathit{alph}(u_\rho)$, we deduce $u_\sigma = u'.u'' \equiv u_{\rho_1.\rho^\omega}$ using Corollary 6.2.4. Similarly, $v_\sigma \equiv v_{\rho_1.\rho^\omega}$. Hence $u_\sigma \sqsubseteq v_\sigma$ entails $u_{\rho_1.\rho^\omega} \sqsubseteq v_{\rho_1.\rho^\omega}$. Using Lemma 6.2.3, we conclude that $u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2}$ can be obtained by picking for ρ_2 a large enough power $\rho_2 \stackrel{\text{def}}{=} \rho.\rho\ldots\rho$ of ρ. Such a ρ_2 further ensures $\rho_2^\omega = \rho^\omega$, so that requirements (a) and (c) are inherited from ρ. □

For the next step, we show how to state the existence of two finite ρ_1 and ρ_2 as in Lemma 6.3.5 under the form of a $\mathsf{PEP}^{\text{reg}}$ problem.

Let $\mathcal{A}_R = (Q, \Sigma, q_0, F, \delta)$ be the Büchi automaton defining R. As is standard, for $q, q' \in Q$, we let $L_{q,q'} \subseteq \Sigma^*$ denote the (regular) language accepted by starting $\mathcal{A}_R$ in q and stopping in q'.

Let $\Sigma' = \{1', 2', \ldots\}$ be a copy of $\Sigma = \{1, 2, \ldots\}$ where letters have been primed: for $x \in \Sigma^*$ and $L \subseteq \Sigma^*$, we let $x' \in \Sigma'^*$ and $L' \subseteq \Sigma'^*$ denote primed versions of x and L.

We can now express condition (a) as a regularity constraint on $\rho_1.\rho_2'$: by definition, $\rho_1.\rho_2^\omega$ belongs to R iff for some $q \in Q$, $\rho_1 \in L_{q_0,q}$ and $\rho_2 \in (L_{q,q} \smallsetminus \epsilon)$. That is, if and only if $\rho_1.\rho_2' \in R_1$ with

$$R_1 \stackrel{\text{def}}{=} \bigcup_{q \in Q} L_{q_0,q}.(L'_{q,q} \smallsetminus \epsilon).$$

Condition (b) can be stated as an embedding property on $\rho_1.\rho_2'$: let $u', v' : (\Sigma \cup \Sigma')^* \to \Gamma^*$ be the extensions of u and v given by $u'_{i'} \stackrel{\text{def}}{=} \epsilon$ and $v'_{i'} \stackrel{\text{def}}{=} v_i$. Then

$$u_{\rho_1} \sqsubseteq v_{\rho_1.\rho_2} \text{ if and only if } u'_{\rho_1.\rho_2'} \sqsubseteq v'_{\rho_1.\rho_2'}.$$

Finally, condition (c) can be expressed as another regularity constraint. Indeed, for $X \subseteq \Gamma$, $\mathit{alph}(u_{\rho_2}) \subseteq X$ and $\mathit{alph}(v_{\rho_2}) \subseteq X$ require $\rho_2 \in u^{-1}(X^*)$ and, respectively, $\rho_2 \in v^{-1}(X^*)$. These are regular conditions on ρ_2 since

inverse morphisms preserve regularity. Let now

$$R_2 \stackrel{\text{def}}{=} \bigcup_{X\subseteq\Gamma}\Big(u^{-1}(X^*)\cap v^{-1}(X^*)\cap \bigcap_{a\in X}\overbrace{\Sigma^*\{i\in\Sigma \mid a\in alph(v_i)\}\Sigma^*}^{a\in alph(v_{\rho_2})}\Big).$$

Clearly, $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$ if and only if $\rho_2 \in R_2$. Hence $alph(u_{\rho_2}) \subseteq alph(v_{\rho_2})$ if, and only if, $\rho_1.\rho_2' \in \Sigma^*.(R_2)'$ where we observe that R_2, hence $\Sigma^*.(R_2)'$ too, are regular.

Finally, u, v has an ω-solution in R iff u', v' has a finite solution in $R_1 \cap (R_2)'$, which provides the reduction from $\mathsf{PEP}^{\omega\text{-reg}}$ to $\mathsf{PEP}^{\text{reg}}$.

Remark 6.3.6 *The automaton for R_1 has size linear in $|\mathcal{A}_R|$. The automaton for R_2 has size exponential in $|\Sigma|$: this is because we consider all subsets $X \subseteq \Sigma$. Hence the reduction from* $\mathsf{PEP}^{\omega\text{-}reg}$ *to* $\mathsf{PEP}^{\text{reg}}$ *is not* LogSpace*when the constraint R is given by a non-deterministic FSA. It is polynomial-space, which is certainly fine enough to state "equivalence" by inter-reducibility between problems that are not primitive-recursive.*

There exists other possible choices for the precise finitary way with which R is supposed to be provided in a PEP *instance: for many of these choices, from various logical formalisms (e.g., MSO) to various automata-based framework (e.g., alternating automata),* LogSpace*reductions from* $\mathsf{PEP}^{\omega\text{-}reg}$ *to* $\mathsf{PEP}^{\text{reg}}$ *exist.*

We conclude this section with the following observation:

Theorem 6.3.7 $\mathsf{PEP}^{\omega\text{-}reg}_{codir}$ *and* $\mathsf{PEP}^{\text{reg}}_{\text{codir}}$ *are equivalent (inter-reducible).*

This can be proved using the same techniques we used in this section, in particular one can state a version of Lemma 6.3.5 that accounts for codirect solutions (while this is not possible for direct solutions). Then a *codirect* infinite solution σ induces the existence of a *codirect* $\rho_1.\rho_2^\omega$, and the existence of such an infinite $\rho_1.\rho_2^\omega$ can be witnessed by a finite $\rho_1.\rho_2'$ that solves a derived $\mathsf{PEP}^{\text{reg}}_{\text{codir}}$ instance.

6.3.2 $\mathsf{PEP}^{\omega\textbf{-reg}}_{\textbf{dir}}$ undecidable

As we have seen, ReachLcs is closely coupled to $\mathsf{PEP}^{\text{reg}}_{\text{dir}}$ and ReachUcs to $\mathsf{PEP}^{\text{reg}}$. On the infinitary case those links still hold since RecReachUcs and $\mathsf{PEP}^{\omega\text{-reg}}$ are equivalent and decidable whereas, as we will see RecReachLcs and $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ are both undecidable.

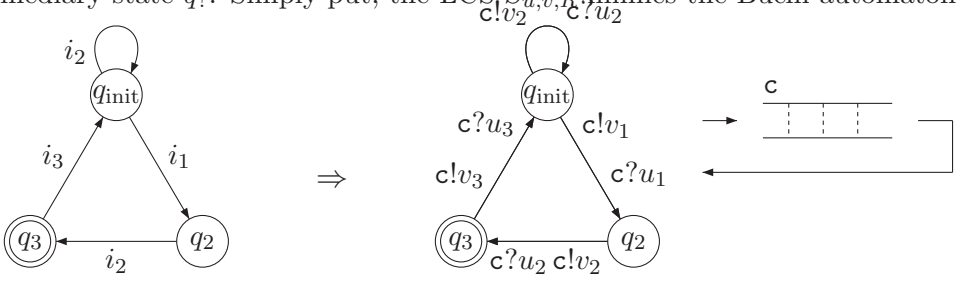
RecReachLcs, the *recurrent reachability problem for LCS's*, is the question whether S has an infinite run $\langle q_{\text{init}}, \epsilon\rangle \to \langle q_1, v_1\rangle \to \langle q_2, v_2\rangle \to \cdots$ with $q_k \in F$ for infinitely many $k \in \mathbb{N}$. RecReachLcs is undecidable [AJ96a] (albeit r.e.).

Lemma 6.3.8 *The following are equivalent:*
(a). *σ is a* direct *solution,*
(b). *For all $k \in \mathbb{N}$, there exists an embedding $h_k : \{1, 2, \ldots, l_k\} \to \{1, 2, \ldots, l'_k\}$ that witnesses $u_{i_1 i_2 \ldots i_k} \sqsubseteq v_{i_1 i_2 \ldots i_k}$,*
(c). *There exists a general embedding $h : \mathbb{N} \to \mathbb{N}$ that witnesses $u_\sigma \sqsubseteq v_\sigma$ and such that its restriction to $\{1, 2, \ldots, l_k\}$ witnesses $u_{i_1 i_2 \ldots i_k} \sqsubseteq v_{i_1 i_2 \ldots i_k}$.*

Proof.[Sketch] (a) and (b) are equivalent by definition of being a direct solution. (c) obviously implies (b). We prove (c) from (b) by defining $h(i) \stackrel{\text{def}}{=} \min_{k=1,2,\ldots} h_k(i)$. □

Lemma 6.3.9 $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$ *reduces to* RecReachLcs.

Proof. The reduction from $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ to RecReachLcs is illustrated in Fig. 6.2, where the "rules" of the form $q \xrightarrow{\mathsf{c}!x\ \mathsf{c}?y} q'$ are just a shorthand description for two consecutive rules $q \xrightarrow{\mathsf{c}!x} q_?$ and $q_? \xrightarrow{\mathsf{c}?y} q'$ that traverse an anonymous intermediary state $q_?$. Simply put, the LCS $S_{u,v,R}$ mimics the Büchi automaton



From $\mathcal{A}_R$ To an LCS $S_{u,v,R}$

Figure 6.2: Reductions between $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ and RecReachLcs

$\mathcal{A}_R$ that defines the constraint $R \subseteq \Sigma^\omega$. A run of the LCS that visits F infinitely often will performs steps $1, 2, 3, \ldots$, writing to the channel some v'_1, v'_2, v'_3, ..., that are subwords (because of message losses) of v_{i_1}, v_{i_2}, v_{i_3}, ... (the writes prescribed by the rules). During these same steps, it reads u_{i_1}, u_{i_2}, u_{i_3}, ..., from the channel. These read letters must have been written earlier, hence for $k = 1, 2, 3, \ldots$, $u_{i_1} \ldots u_{i_k}$ is a prefix of $v'_1 \ldots v'_k$, hence a subword of $v_{i_1} \ldots v_{i_k}$. Finally, $\sigma \stackrel{\text{def}}{=} i_1.i_2.i_3 \ldots$ is a direct solution.

Reciprocally, given a direct solution $\sigma = i_1.i_2.i_3 \ldots$, it is possible (using the general embedding provided by Lemma 6.3.8) to find subwords v'_1, v'_2, v'_3, ... of v_{i_1}, v_{i_2}, v_{i_3}, ... s.t., for all $k = 1, 2, \ldots$, $u_{i_1} \ldots u_{i_k}$ is a prefix of $v'_1 \ldots v'_k$. Using these v'_k, one easily obtains an infinite run of the LCS that shows the associated RecReachLcs is positive. □

Lemma 6.3.10 RecReachLcs *reduces to* $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$.

Proof. Consider a RecReachLcs instance $S = (Q, \mathtt{M}, \{\mathtt{c}\}, \Delta)$ with given q_{init} and F. With it, we associate a $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ instance where $\Sigma = \Delta$ and where $R \subseteq \Sigma^\omega$ is given by the Büchi automaton that is exactly like S, with the difference that any rule δ between some states q and q' is now a transition $q \xrightarrow{\delta} q'$ in $\mathcal{A}_R$. The morphisms u, v are defined by $u(\delta) \stackrel{\text{def}}{=}$ "what rule δ reads in channel $\mathtt{c}$", $v(\delta) \stackrel{\text{def}}{=}$ "what δ writes in $\mathtt{c}$". Since $u(\delta) = \epsilon$ or $v(\delta) = \epsilon$ for every rule (LCS's rules either read or write to $\mathtt{c}$, not both), S (essentially) coincides with $S_{u,v,R}$ (Fig. 6.2). Hence the proof of Lemma 6.3.9 shows that u, v, R is a positive $\mathsf{PEP}^{\omega\text{-reg}}$ instance iff the original RecReachUcs instance is positive. □

Directly from these two lemmas $\mathsf{PEP}^{\omega\text{-reg}}_{\text{dir}}$ is equivalent to RecReachLcs and

Theorem 6.3.11 $\mathsf{PEP}^{\omega\text{-}reg}_{dir}$ *is (r.e. but) undecidable.*

6.4 Varying constraint

Here are first presented the different methods to place the regular constraint that make sense we could think of. They are all essentially equivalent. This section is a justification of our choice of $\mathsf{PEP}^{\text{reg}}$ as our central problem.

The next variants are stronger versions where the constraint has some counting capacity. It turns out that every kind of constraint that are stronger than regular languages turns out to make the problem undecidable.

6.4.1 Constraining u_σ and v_σ

$\mathsf{PEP}^{\text{u\_reg}}$ is like $\mathsf{PEP}^{\text{reg}}$ except that the constraint $R \subseteq \Gamma^*$ now applies to u_σ: a solution is some $\sigma \in \Sigma^*$ with $u_\sigma \in R$ (and $u_\sigma \sqsubseteq v_\sigma$). Similarly, $\mathsf{PEP}^{\text{v\_reg}}$ has the constraint apply to v_σ, while $\mathsf{PEP}^{\text{uv\_reg}}$ has two constraints, $R_1, R_2 \subseteq \Gamma^*$, that apply to, respectively and simultaneously, u_σ and v_σ. These problems also have directed versions.

Proposition 6.4.1 *1.* $\mathsf{PEP}^{\text{uv\_reg}}$ *reduces to* $\mathsf{PEP}^{\text{reg}}$.
2. $\mathsf{PEP}^{\text{uv\_reg}}_{\text{dir}}$ *reduces to* $\mathsf{PEP}^{\text{reg}}_{\text{dir}}$.

Proof. Let u, v, R_1, R_2 be a $\mathsf{PEP}^{\text{uv\_reg}}$ instance. Let $R \stackrel{\text{def}}{=} u^{-1}(R_1) \cap v^{-1}(R_2)$. (Recall that the image of a regular R by an inverse morphism is regular and can easily be constructed from R.) By definition $\sigma \in R$ iff $u_\sigma \in R_1$ and $v_\sigma \in R_2$. Thus the $\mathsf{PEP}^{\text{reg}}$ instance u, v, R is positive iff u, v, R_1, R_2 is. We further note that the directness of σ is untouched by the transformation. □
Reductions exist in the other direction, as the next two propositions show.

Proposition 6.4.2 *1.* $\mathsf{PEP}^{\mathrm{reg}}$ *reduces to* $\mathsf{PEP}^{\mathrm{v\_reg}}$.
2. $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ *reduces to* $\mathsf{PEP}^{\mathrm{v\_reg}}_{\mathrm{dir}}$.

Proof.[Sketch] Let u, v, R be a $\mathsf{PEP}^{\mathrm{reg}}$ instance. W.l.o.g., we may assume that $\Sigma \cap \Gamma = \emptyset$. Define a $\mathsf{PEP}^{\mathrm{v\_reg}}$ instance u', v', R' by letting $v' : \Sigma^* \to (\Gamma \cup \Sigma)^*$ be given by $v'_i \stackrel{\mathrm{def}}{=} i.v_i$ and keeping $u' = u$ unchanged. Let $R' \stackrel{\mathrm{def}}{=} h^{-1}(R)$ where $h : (\Gamma \cup \Sigma)^* \to \Gamma^*$ is the erasing morphism that suppresses letters from Σ. Note that $v'_\sigma \in R'$ iff $\sigma = h(v'_\sigma) \in R$, so that u', v', R' is a positive $\mathsf{PEP}^{\mathrm{v\_reg}}$ instance iff u, v, R is a positive $\mathsf{PEP}^{\mathrm{reg}}$ instance. Finally, this reduction preserves the directness of solutions. □

Proposition 6.4.3 *1.* $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ *reduces to* $\mathsf{PEP}^{\mathrm{u\_reg}}$.
2. $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ *reduces to* $\mathsf{PEP}^{\mathrm{u\_reg}}$.

Proof.[Sketch] Let u, v, R be a $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ instance. W.l.o.g., we assume $\Sigma = \{1, 2, \ldots, k\}$ and let $\Sigma' \stackrel{\mathrm{def}}{=} \{0\} \cup \Sigma$ with $g : \Sigma'^* \to \Sigma^*$ the associated erasing morphism. We also assume $\Gamma \cap \Sigma' = \emptyset$ and let $\Gamma' \stackrel{\mathrm{def}}{=} \Gamma \cup \Sigma'$, with $h : \Gamma'^* \to \Sigma^*$ as erasing morphism.

With u, v, R, we associate a $\mathsf{PEP}^{\mathrm{u\_reg}}$ instance u', v', R' based on Σ' and Γ', and defined by $u'_0 \stackrel{\mathrm{def}}{=} \epsilon$, $v'_0 \stackrel{\mathrm{def}}{=} 1.2 \ldots k$, and, for $i \in \Sigma$, $u'_i \stackrel{\mathrm{def}}{=} i.u_i$ and $v'_i \stackrel{\mathrm{def}}{=} v_i$. Letting $R' = h^{-1}(R)$ ensures that $u'_\sigma \in R'$ iff $g(\sigma) \in R$. Clearly, if $u'_\sigma \sqsubseteq v'_\sigma$, then $u_{g(\sigma)} \sqsubseteq v_{g(\sigma)}$. Conversely, if $u_{\sigma'} \sqsubseteq v_{\sigma'}$, it is possible to find a $\sigma \in g^{-1}(\sigma')$ that satisfies $u'_\sigma \sqsubseteq v'_\sigma$: this is just a matter of inserting enough 0's at the appropriate places (and this is where we use the assumption that all v_i's have length ≤ 1).

Finally, this reduction preserves the directness of solutions. □

Now, since $\mathsf{PEP}^{\mathrm{u\_reg}}$ and $\mathsf{PEP}^{\mathrm{v\_reg}}$ are special cases of $\mathsf{PEP}^{\mathrm{uv\_reg}}$, and since $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$ is a special case of $\mathsf{PEP}^{\mathrm{reg}}$, Propositions 4.1.2, 6.4.1, 6.4.2 and 6.4.3 entail the following.

Theorem 6.4.4 $\mathsf{PEP}^{\mathrm{reg}}$, $\mathsf{PEP}^{\mathrm{reg}}_{\leq 1}$, $\mathsf{PEP}^{\mathrm{u\_reg}}$, $\mathsf{PEP}^{\mathrm{v\_reg}}$ *and* $\mathsf{PEP}^{\mathrm{uv\_reg}}$ *are inter-reducible. Furthermore, they are also inter-reducible with their directed versions.*

6.4.2 Context-free and Presburger constraints on solutions

Write $\mathsf{PEP}^{\mathrm{cf}}$ for the extension of $\mathsf{PEP}^{\mathrm{reg}}$ where R can be any context-free language (say, given in the form of a context-free grammar) and $\mathsf{PEP}^{\mathrm{dcf}}$ for $\mathsf{PEP}^{\mathrm{cf}}$ restricted to *deterministic* context-free constraints. Further write $\mathsf{PEP}^{\mathrm{Pres}}$ for the extension where $R \subseteq \Sigma^*$ can be any language defined by a Presburger constraint over the number of occurrences of each letter from Σ

(or, equivalently, the commutative image of R is a semilinear subset of the commutative monoid $\mathbb{N}^{\Sigma}$).

Theorem 6.4.5 $\mathsf{PEP}^{\mathrm{dcf}}$, $\mathsf{PEP}^{\mathrm{cf}}$ *and* $\mathsf{PEP}^{\mathrm{Pres}}$ *are undecidable.*

Proof. The (classic) PCP problem reduces to $\mathsf{PEP}^{\mathrm{dcf}}$ or $\mathsf{PEP}^{\mathrm{Pres}}$ by associating, with an instance $u, v : \Sigma^* \to \Gamma^*$, the constraint $R_{\geq} \subseteq \Sigma^+$ defined by

$$\sigma \in R_{\geq} \overset{\text{def}}{\Leftrightarrow} |u_\sigma| \geq |v_\sigma| \text{ and } \sigma \neq \epsilon.$$

Obviously, $u_\sigma \sqsubseteq v_\sigma$ and $\sigma \in R_{\geq}$ iff $u_\sigma = v_\sigma$. Observe that $R_{\geq}$ is easily defined in the quantifier-free fragment of Presburger logic. Furthermore, since $R_{\geq}$ can be recognized by a counter machine with a single counter, it is indeed deterministic context-free. □

6.5 Appendix

6.5.1 $\mathsf{PEP}^{\mathrm{reg}}$ is equivalent to ReachUcs and $\mathsf{PEP}^{\omega\text{-reg}}$ is equivalent to RecReachUcs

In this section we will prove the link between UCS and PEP for both finite and infinite case. To this end, we will use $\mathsf{2PCEP}^{\mathrm{reg}}$, an intermediate problem closer to the behaviour of UCS's than PEP. It uses correspondence and embedding between two words to mimic the behaviour of both parts of an UCS. The first step, from ReachUcs to $\mathsf{2PCEP}^{\mathrm{reg}}$ is essentially a detailed explanation of why this abstraction is correct. The second part, from $\mathsf{2PCEP}^{\mathrm{reg}}$ to $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ relies on the fact that two languages which must match through two morphisms can be seen as the intersection of those languages.

Lemma 6.5.1 *1.* ReachUcs *and* $\mathsf{2PCEP}^{reg}$ *are equivalent.*
2. RecReachUcs *and* $\mathsf{2PCEP}^{\omega\text{-}reg}$ *are equivalent.*

Lemma 6.5.2 *1.* $\mathsf{2PCEP}^{reg}$ *reduces to* $\mathsf{PEP}^{\mathrm{reg}}$.
2. $\mathsf{2PCEP}^{\omega\text{-}reg}$ *reduces to* $\mathsf{PEP}^{\omega\text{-}reg}$.

Commuting UCS steps

We first state a trivial but important property about runs of unidirectional systems. Let $S = (Q_1, Q_2, \mathtt{M}, \{\mathtt{r}, \mathtt{l}\}, \Delta_1, \Delta_2)$ be some UCS, and $\langle q_1, q_2, x, y\rangle \xrightarrow{\delta_2} \langle q_1, q_2', x', y'\rangle \xrightarrow{\delta_1} \langle q_1', q_2', x'', y''\rangle$ be two consecutive steps with $\delta_1 \in \Delta_1$ and $\delta_2 \in \Delta_2$, i.e., where the receiver performs the first step, and the sender the second step. Then it is possible to fire δ_1 before δ_2 and reach the same configuration. More precisely, there exists x''' and y''' with

$\langle q_1, q_2, x, y\rangle \xrightarrow{\delta_1} \langle q_1', q_2, x''', y'''\rangle \xrightarrow{\delta_1} \langle q_1', q_2', x'', y''\rangle$.

The corollaries are

Lemma 6.5.3 *If S has a run $\langle q_1, q_2, x, y\rangle \xrightarrow{\Delta_1 \cup \Delta_2}{}^* \langle q_1', q_2', x', y'\rangle$ then it has one such run of the form*

$$\langle q_1, q_2, x, y\rangle \xrightarrow{\Delta_1}{}^* \langle q_1', q_2, x'', y''\rangle \xrightarrow{\Delta_2}{}^* \langle q_1', q_2', x', y'\rangle.$$

Lemma 6.5.4 *If S has an infinite run from $\langle q_0^1, q_0^2, x_0, y_0\rangle$ of the form*

$$\langle q_0^1, q_0^2, x_0, y_0\rangle \to \langle q_1^1, q_1^2, x_1, y_1\rangle \to \langle q_2^1, q_2^2, x_2, y_2\rangle \to \cdots$$

with $q^1 = q_i^1$ for infinitely many i's, and $q^2 = q_i^2$ for infinitely many i's (not necessarily the same), then it has one such run with $(q^1, q^2) = (q_i^1, q_i^2)$ for infinitely many i's.

from ReachUcs **to** 2PCEP$^{\mathbf{reg}}$

Lemma 6.5.5 2PCEP^{reg} *is equivalent to* ReachUcs*, and* 2PCEP$^{\omega\text{-}reg}$ *is equivalent to* RecReachUcs*.*

The proof rely on the two following lemmas

Lemma 6.5.6 2PCEP^{reg} *reduces to* ReachUcs*, and* 2PCEP$^{\omega\text{-}reg}$ *to* RecReachUcs*.*

Proof. For this, consider a 2PCEP$^{\text{reg}}$ instance $f_1, g_1, f_2, g_2, R_1, R_2$ as in Definition 5.2.1. Further assume that, for $i = 1, 2$, R_i is given by some FSA $\mathcal{A}_i = (Q_i, \Sigma_i, q_{\text{init}}^i, F_i, \delta_i)$.

With this instance, we associate an UCS where the the sender is obtained from $\mathcal{A}_2$ by replacing transitions $q \xrightarrow{i} q' \in \delta_2$ with rules $q \xrightarrow{\mathtt{r}!f_2(i)\ \mathtt{1}!g_2(i)} q'$, and the receiver is obtained from $\mathcal{A}_1$ by replacing transitions $q \xrightarrow{i} q' \in \delta_1$ with rules $q \xrightarrow{\mathtt{r}?f_1(i)\ \mathtt{1}?g_1(i)} q'$.

If the 2PCEP$^{\text{reg}}$ instance is positive, then a solution σ_1, σ_2 can be used in a straightforward way to build, out of σ_2, a run in the UCS that will start from $\langle q_{\text{init}}^2, q_{\text{init}}^1, \epsilon, \epsilon\rangle$, will reach some $\langle q_{\text{final}}^2, q_{\text{init}}^1, f_2(\sigma_2), x\rangle$ for some $q_{\text{final}}^2 \in F_2$, and where, using message losses, we can choose to reach any $x \sqsubseteq g_2(\sigma_2)$. By picking $x = g_1(\sigma_1)$, we can now continue the run, using σ_1, and reach $\langle q_{\text{final}}^1, q_{\text{final}}^2, \epsilon, \epsilon\rangle$ for some $q_{\text{final}}^1 \in F_1$.

Reciprocally, using Lemma 6.5.3, a run from $\langle q_{\text{init}}^2, q_{\text{init}}^1, \epsilon, \epsilon\rangle$ to some $\langle q_{\text{final}}^1, q_{\text{final}}^2, \epsilon, \epsilon\rangle$ can be reordered into some

$$\langle q_{\text{init}}^2, q_{\text{init}}^1, \epsilon, \epsilon\rangle \underbrace{\xrightarrow{r_1} \xrightarrow{r_2} \cdots \xrightarrow{r_n}}_{\text{rules from } \Delta_1} \langle q_{\text{final}}^2, q_{\text{init}}^1, x, y\rangle \underbrace{\xrightarrow{r_1'} \xrightarrow{r_2'} \cdots \xrightarrow{r_m'}}_{\text{rules from } \Delta_2} \langle q_{\text{final}}^1, q_{\text{final}}^2, \epsilon, \epsilon\rangle$$

where all sender's steps occur first, followed by the receiver steps. This translates into a path $q^2_{\text{init}} \xrightarrow{\sigma_2} q^2_{\text{final}}$ in $\mathcal{A}_2$, and $q^1_{\text{init}} \xrightarrow{\sigma_1} q^1_{\text{final}}$ in $\mathcal{A}_1$ where $f_2(\sigma_2) = x = f_1(\sigma_1)$, and where $g_2(\sigma_2) \sqsupseteq y = g_1(\sigma_1)$, solving the 2PCEP$^{\text{reg}}$ instance.

Finally, the 2PCEP$^{\text{reg}}$ instance is positive iff the associated ReachUcs instance is. Hence 2PCEP$^{\text{reg}}$ reduces to ReachUcs.

The same association of an UCS with $f_1, g_1, f_2, g_2, \mathcal{A}_1, \mathcal{A}_2$ shows that 2PCEP$^{\omega\text{-reg}}$ reduces to RecReachUcs.

Indeed, an infinite solution σ_1, σ_2 in some ω-regular languages R_1 and R_2, can be used to build an infinite run of the UCS that visit infinitely many configurations $\langle q^2_{\text{final}}, q^1_i, x_i, y_i\rangle$ with some $q^2_{\text{final}} \in F_2$, and infinitely many configurations $\langle q^2_i, q^1_{\text{final}}, x'_i, y'_i\rangle$ with some $q^1_{\text{final}} \in F_1$. Using Lemma 6.5.4, this run can be reordered into a run visiting infinitely many configurations $\langle q^2_{\text{final}}, q^1_{\text{final}}, x''_i, y''_i\rangle$, showing the RecReachUcs instance is positive.

Reciprocally, from an infinite run of the UCS that visits infinitely many configurations of the form $\langle q^2_{\text{final}}, q^1_{\text{final}}, x''_i, y''_i\rangle$, one extracts two solutions σ_1, σ_2 that show that the 2PCEP$^{\omega\text{-reg}}$ instance is positive.
□

Lemma 6.5.7 ReachUcs *reduces to* 2PCEP^{reg}*, and* RecReachUcs *to* 2PCEP$^{\omega\text{-}reg}$*.*

Proof. Consider an ReachUcs instance with some UCS
$S = (Q_1, Q_2, \mathtt{M}, \{\mathtt{r}, \mathtt{l}\}, \Delta_1, \Delta_2)$, some initial states $q^1_{\text{init}}, q^2_{\text{init}}$, and some sets of final states F_1, F_2.

With this instance, we associate a 2PCEP$^{\text{reg}}$ instance where $\Sigma_1 \stackrel{\text{def}}{=} \Delta_2$ and $\Sigma_2 \stackrel{\text{def}}{=} \Delta_1$ are the set of rules. Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ for R_1 and R_2 are obtained from the control graph of the receiver (resp., the sender) in the obvious way. (Note that we extract FSA's from an ReachUcs instance, and Büchi automata from an RecReachUcs instance.) The morphisms are defined in the obvious way:

$$f_1(\delta) \stackrel{\text{def}}{=} x \text{ and } g_1(\delta) \stackrel{\text{def}}{=} y \text{ for } \delta = q \xrightarrow{\mathtt{r}?x\ \mathtt{l}?y} r \text{ in } \Delta_2,$$
$$f_2(\delta) \stackrel{\text{def}}{=} x \text{ and } g_2(\delta) \stackrel{\text{def}}{=} y \text{ for } \delta = q \xrightarrow{\mathtt{r}!x\ \mathtt{l}!y} r \text{ in } \Delta_1.$$

□

from 2PCEP$^{\text{reg}}$ to PEP$^{\text{reg}}_{\text{dir}}$

We consider a 2PCEP instance f_1, g_1, f_2, g_2 where we assume that the morphisms are short, i.e., f_i and g_i can be seen as having type $(\Sigma_i \cup \{\epsilon\}) \to (\Gamma_\cup\{\epsilon\})$. For 2PCEP$^{\text{reg}}$ and 2PCEP$^{\omega\text{-reg}}$, and thanks to the possibility offered

by the regular constraints, this assumption is no loss of generality, as can be easily proved using the techniques from section 4.1.1.

Let $\Sigma \stackrel{\text{def}}{=} (\Sigma_1 \cup \{\epsilon\}) \times (\Sigma_2 \cup \{\epsilon\})$ and define $X \subseteq \Sigma$ by

$$(i, j) \in X \text{ if and only if } f_1(i) = f_2(j).$$

Then $(i_1, j_1).(i_2, j_2) \ldots (i_n, j_n) \in X^*$ implies that $f_1(i_1.i_2 \ldots i_n) = f_2(j_1.j_2 \ldots j_n)$. Reciprocally, if $f_1(\sigma_1) = f_2(\sigma_2)$, then σ_1 and σ_2 can be decomposed under the form $\sigma_1 = i_1.i_2 \ldots i_n$ and $\sigma_2 = j_1.j_2 \ldots j_n$ such that $(i_k, j_k) \in X$ for $k = 1, \ldots, n$. Observe that in this decomposition, $n \geq |\sigma_i|$ is possible since $i_k = \epsilon$ or $j_k = \epsilon$ (or both) is allowed.

Now define projection morphisms $h_1 : \Sigma^* \to \Sigma_1^*$ and $h_2 : \Sigma^* \to \Sigma_2^*$ in the obvious way, and let $u, v : \Sigma^* \to \Gamma^*$ be two morphisms given by $u \stackrel{\text{def}}{=} g_1 \circ h_1$ and $v \stackrel{\text{def}}{=} g_2 \circ h_2$. Then $u_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)} \sqsubseteq v_{(i_1,j_1).(i_2,j_2)\ldots(i_n,j_n)}$ if and only if $g_1(i_1.i_2 \ldots i_n) \sqsubseteq g_2(j_1.j_2 \ldots j_n)$.

Finally, the $\mathsf{2PCEP}^{\text{reg}}$ instance with regular constraints R_1, R_2 translates into an equivalent $\mathsf{PEP}^{\text{reg}}$ instance, with morphisms u and v as above, and with constraint

$$R \stackrel{\text{def}}{=} X^* \cap {h_1}^{-1}(R_1) \cap {h_2}^{-1}(R_2),$$

which is regular. Similarly, the $\mathsf{2PCEP}^{\omega\text{-reg}}$ instance with ω-regular constraints R_1, R_2 translates into an equivalent $\mathsf{PEP}^{\omega\text{-reg}}$ instance, with same morphisms u and v, and with constraint

$$R \stackrel{\text{def}}{=} X^\omega \cap {h_1}^{-1}(R_1) \cap {h_2}^{-1}(R_2),$$

which is ω-regular.

Chapter 7

Direct $\mathsf{PEP}^{\mathrm{reg}}$ algorithm

In this chapter, we give a direct proof of decidability of $\mathsf{PEP}^{\mathrm{reg}}$. For sake of simplicity, no complexity result is shown: this result does not rely on the complex miniaturisation results.

However, the main interest of presenting this proof is to introduce *blockers languages*. This turned out to be an useful notion with many good properties, which are the subject of the next chapter.

Theorem 7.0.8 $\mathsf{PEP}^{\mathrm{reg}}$ *is decidable.*

7.1 Blocking and stable families

In the rest of this section, we assume a given $\mathsf{PEP}^{\mathrm{reg}}$ instance made of $u, v : \Sigma^* \to \Gamma^*$ and $R \subseteq \Sigma^*$. Let $\mathcal{L}(R)$ be the residual languages of R. We consider some $\mathcal{L}(R)$-indexed families of languages in Γ^*:

Definition 7.1.1 (Blocking family) *An $\mathcal{L}(R)$-indexed family $(A_L, B_L)_{L \in \mathcal{L}(R)}$ of languages in Γ^* is a* blocking family *if for all $L \in \mathcal{L}(R)$:*

$$\sigma \in L \text{ and } \alpha \in A_L \text{ imply } \alpha u_\sigma \not\sqsubseteq v_\sigma, \tag{B1}$$

$$\sigma \in L \text{ and } \beta \in B_L \text{ imply } u_\sigma \not\sqsubseteq \beta v_\sigma. \tag{B2}$$

The terminology "blocking" comes from the fact that the α prefix "blocks" solutions in L to $\alpha.u_\sigma \sqsubseteq v_\sigma$. For B_L, the situation is dual: adding $\beta \in B_L$ is not enough to allow solutions in L to $u_\sigma \sqsubseteq \beta.v_\sigma$.

There is a largest blocking family, called the *blocker* languages, or blocker family, $(X_L, Y_L)_{L \in \mathcal{L}(R)}$, given by:

$$X_L \stackrel{\text{def}}{=} \{\alpha \in \Gamma^* \mid \forall \sigma \in L, \alpha u_\sigma \not\sqsubseteq v_\sigma\}, \tag{B3}$$

$$Y_L \stackrel{\text{def}}{=} \{\beta \in \Gamma^* \mid \forall \sigma \in L, u_\sigma \not\sqsubseteq \beta v_\sigma\}. \tag{B4}$$

A blocking family provides information about the absence of solutions to several variants of our $\mathsf{PEP}^{\mathrm{reg}}$ instance. For example, the u, v, R instance itself is positive iff $\epsilon \notin X_R$ iff $\epsilon \notin Y_R$.

For proving that a given family is blocking, we use a criterion called "stability".

Before defining stability, we need some new subword combinatorics notions.

When $x \not\sqsubseteq y$, we decompose x as a concatenation $x = x_1x_2$ such that x_1 is the *longest* prefix of x with $x_1 \sqsubseteq y$. We call x_1 the "*matched prefix*" and x_2 the "*unmatched suffix*". We use $x \ominus y$ to denote the unmatched suffix. For example $\underline{aa}bcabc \ominus \underline{ba}\underline{ca} = bcabc$. Note that $x \ominus y$ is only defined when $x \not\sqsubseteq y$ (hence $x \ominus y \neq \epsilon$).

When $x \sqsubseteq y$, we decompose y as a concatenation $y = y_1y_2$ such that y_1 is the *shortest* prefix of y with $x \sqsubseteq y_1$. We call y_1 the "*used prefix*" and y_2 the "*available suffix*". We use $y \oslash x$ to denote the available suffix. For example, $a\underline{bc}\underline{a}bc \oslash \underline{ba} = bc$. Note that $y \oslash x$ is only defined when $x \sqsubseteq y$.

Definition 7.1.2 (Stable family) *An $\mathcal{L}(R)$-indexed family $(A_L, B_L)_{L\in\mathcal{L}(R)}$ of languages is* stable *iff, for all $L \in \mathcal{L}(R)$:*

1. *$A_L \subseteq \Gamma^*$ is upward-closed and $B_L \subseteq \Gamma^*$ is downward-closed,*
2. *if $\epsilon \in L$, then $\epsilon \notin A_L \cup B_L$,*
3. *for all $i \in \Sigma$ and $\alpha \in A_L$:*
   (a) *if $\alpha.u_i \sqsubseteq v_i$ then $v_i \oslash \alpha.u_i \in B_{L^{-1}i}$,*
   (b) *if $\alpha.u_i \not\sqsubseteq v_i$ then $(\alpha.u_i) \ominus v_i \in A_{L^{-1}i}$,*
4. *for all $i \in \Sigma$ and $\beta \in B_L$:*
   (a) *if $u_i \sqsubseteq \beta.v_i$ then $(\beta.v_i) \oslash u_i \in B_{L^{-1}i}$,*
   (b) *if $u_i \not\sqsubseteq \beta.v_i$ then $u_i \ominus \beta.v_i \in A_{L^{-1}i}$.*

Recall that A_L and B_L, being respectively upward- and downward-closed, must be regular languages. Observe also that $\epsilon \in B_L$ iff $B_L \neq \emptyset$, while $\epsilon \in A_L$ iff $A_L = \Gamma^*$.

First recall this simple fact frequently used when studying $\mathsf{PEP}^{\mathrm{reg}}$.

Fact 7.1.3 (proof in appendix A.1) *1. If $xy \sqsubseteq z$, then there exists a factorization $z = z_1z_2$ of z such that $x \sqsubseteq z_1$ and $y \sqsubseteq z_2$.*
2. If $x \sqsubseteq yz$, then there exists a factorization $x = x_1x_2$ of x such that $x_1 \sqsubseteq y$ and $x_2 \sqsubseteq z$.

However, this fact only works one way. For deeper analyses, we shall need the following more powerful tool.

Lemma 7.1.4 (Decomposition Lemma, proof in appendix A.4)

$$u.w \sqsubseteq v.t \text{ if and only if } \begin{cases} & u \sqsubseteq v \text{ and } w \sqsubseteq v \oslash u.t \\ \text{or} & u \not\sqsubseteq v \text{ and } u \ominus v.w \sqsubseteq t. \end{cases}$$

Proposition 7.1.5 (Soundness) *A stable family is a blocking family.*

Proof. Assume that $(A_L, B_L)_{L\in\mathcal{L}(R)}$ is stable. We prove that it satisfies (B1) and (B2) by induction on the length of σ.

Base case: $\sigma = \epsilon$. Hence $u_\sigma = v_\sigma = \epsilon$. Assuming $\alpha u_\sigma \sqsubseteq v_\sigma$ requires $\alpha = \epsilon$ but if $\sigma \in L$, stability implies that $\epsilon \notin A_L$. $\sigma \in L$ also implies that B_L is empty so that $u_\sigma \not\sqsubseteq \beta v_\sigma$ is vacuously true.

Inductive case: assume that σ is some $i.\rho$ with $i \in \Sigma$ and $\rho \in \Sigma^*$. Recall that $\sigma \in L$ iff $\rho \in L^{-1}i$.

Let $\alpha \in A_L$. If $\alpha u_i \sqsubseteq v_i$, then $v_i \oslash \alpha u_i \in B_{L^{-1}i}$ by stability. Hence $u_\rho \not\sqsubseteq (v_i \oslash \alpha u_i)v_\rho$ by ind. hyp. Then $\alpha u_\sigma = \alpha u_i u_\rho \not\sqsubseteq v_i v_\rho = v_\sigma$ by Lemma 7.1.4. If, on the other hand, $\alpha u_i \not\sqsubseteq v_i$, then $(\alpha u_i) \ominus v_i \in A_{L^{-1}i}$ by stability, hence $(\alpha u_i) \ominus v_i u_\rho \not\sqsubseteq v_\rho$ by ind. hyp., entailing $\alpha u_\sigma \not\sqsubseteq v_\sigma$ by Lemma 7.1.4.

For $\beta \in B_L$ the reasoning is similar. If $u_i \sqsubseteq \beta v_i$, then $(\beta v_i) \oslash u_i \in B_{L^{-1}i}$ by stability, hence $u_\rho \not\sqsubseteq (\beta v_i) \oslash u_i v_\rho$ by ind. hyp., hence $u_\sigma = u_i u_\rho \not\sqsubseteq \beta v_i v_\rho = \beta v_\sigma$ by Lemma 7.1.4. If, on the other hand, $u_i \not\sqsubseteq \beta v_i$, then $u_i \ominus \beta v_i \in A_{L^{-1}i}$ by stability, hence $u_i \ominus \beta v_i u_\rho \not\sqsubseteq v_\rho$ by ind. hyp., hence $u_\sigma \not\sqsubseteq \beta v_\sigma$.

□

The criterion is also sufficient:

Proposition 7.1.6 (Completeness) *The blocker family $(X_L, Y_L)_{L\in\mathcal{L}(R)}$ is stable.*

Proof. Clearly, as defined by (B3) and (B4) and for any $L \in \mathcal{L}(R)$, X_L is upward-closed and Y_L is downward-closed. Similarly, $\epsilon \notin X_L$ and $\epsilon \notin Y_L$ when $\epsilon \in L$.

It remains to check conditions *3* and *4* for stability. We consider four cases:

3a Assume that $\alpha u_i \sqsubseteq v_i$ for some i in Σ and some α in some X_L. If, by way of contradiction, we assume that $v_i \oslash \alpha.u_i \notin Y_{L^{-1}i}$ then, by (B4), there is some $\rho \in L^{-1}i$ such that $u_\rho \sqsubseteq v_i \oslash \alpha.u_i v_\rho$. Thus $\alpha u_i u_\rho \sqsubseteq v_i v_\rho$ by Lemma 7.1.4, i.e., $\alpha u_\sigma \sqsubseteq v_\sigma$ writing σ for $i.\rho$. But, since $\sigma \in L$, this contradicts $\alpha \in X_L$.

4a A similar reasoning applies if we assume that $u_i \sqsubseteq \beta v_i$ for some i in Σ and some β in some Y_L while $(\beta v_i) \oslash u_i \notin Y_{L^{-1}i}$: we derive from (B4) that $u_\rho \sqsubseteq (\beta v_i) \oslash u_i v_\rho$ for some $\rho \in L^{-1}i$. Hence $u_i u_\rho \sqsubseteq \beta v_i v_\rho$ by Lemma 7.1.4, a contradiction since $i.\rho \in L$.

3b If we assume that $\alpha u_i \not\sqsubseteq v_i$ for $\alpha \in X_L$ and $(\alpha u_i) \ominus v_i \notin X_{L^{-1}i}$ then, by (B3), there is some $\rho \in L^{-1}i$ s.t. $(\alpha u_i) \ominus v_i u_\rho \sqsubseteq v_\rho$. Then $\alpha u_i u_\rho \sqsubseteq v_i v_\rho$ by Lemma 7.1.4, a contradiction since $i.\rho \in L$.

4b Similarly, assuming that $u_i \not\sqsubseteq \beta v_i$ while $u_i \ominus \beta v_i \notin A_{L^{-1}i}$, we derive $(u_i \ominus \beta v_i)u_\rho \sqsubseteq v_i v_\rho$, i.e., $u_i u_\rho \sqsubseteq \beta v_i v_\rho$, another contradiction.

□

7.2 Computability

Lemma 7.2.1 *Let $v \in \Gamma^*$ be a word, and $\mathcal{A}$ a NFA recognizing some regular language $L \subseteq \Gamma^*$. Then $L \ominus v \stackrel{def}{=} \{u \ominus v \mid u \in L\}$ is regular and a NFA for it can be built from $\mathcal{A}$.*

Proof.[Sketch] For some u of the form u_1u_2, $u \ominus v = u_2$ $(= u^{-1}u_1)$ if $u_1 \sqsubseteq v$ and either $u_2 = \epsilon$ or u_2 is some au_3 and $u_1 a \not\sqsubseteq v$. Hence $L \ominus v$ contains all $L^{-1}u_1$ for $u_1 \sqsubseteq v$ such that $v \oslash u_1 = \epsilon$, and all $(L \cap u_1 a\Gamma^*)^{-1}u_1$ for $u_1 \sqsubseteq v$ and a such that $u_1 a \not\sqsubseteq v$. This is a finite union of languages derived from L by regularity-preserving operations like quotient or intersection. □

Lemma 7.2.2 *Let $v \in \Gamma^*$ be a word, and $\mathcal{A}$ a NFA recognizing some regular language $L \subseteq \Gamma^*$. Then $L \oslash v \stackrel{def}{=} \{u \oslash v \mid u \in L \text{ and } v \sqsubseteq u\}$ is regular and a NFA for it can be built from $\mathcal{A}$.*

Proof.[Sketch] Assume that v is some $a_1.a_2 \dots a_n$ and $u = u_1u_2$. Then $u \oslash v = u_2$ $(= u^{-1}u_1)$ iff $u_1 \in V$ for V defined by the following regular expression:

$$(\Gamma \setminus \{a_1\})^* a_1 (\Gamma \setminus \{a_1, a_2\})^* a_2 (\Gamma \setminus \{a_2, a_3\})^* \dots a_{n-1} (\Gamma \setminus \{a_{n-1}, a_n\})^* a_n.$$

Hence $L \oslash v = L^{-1}V$ can be obtained by right-quotienting L with a regular language. □

Proposition 7.2.3 (Stability is decidable) *It is decidable whether an $\mathcal{L}(R)$-indexed family $(A_L, B_L)_{L \in \mathcal{L}(R)}$ of regular languages is a stable family.*

Proof. We can assume that the A_L and B_L are given by DFA's. Conditions *1* and *2* of stability are easy to check.

For a given $i \in \Sigma$ and $L \in \mathcal{L}(R)$, checking condition *3a* needs only consider α's that are shorter than v_i, which is easily done.

Checking condition *3b* is trickier. One way to do it is to consider the set of all α's such that $\alpha u_i \not\sqsubseteq v_i$. This is a regular set that can be obtained effectively. Then the set of all corresponding $(\alpha u_i) \ominus v_i$ is also regular and effective (Lemma 7.2.1) so that we can check that it is included in $A_{L^{-1}i}$.

For condition *4a*, and given some $L \in \mathcal{L}(R)$ and some $i \in \Sigma$, the set of all β's such that $u_i \sqsubseteq \beta v_i$ is regular and effective. One can then compute the corresponding set of all $(\beta v_i) \oslash u_i$, again regular and effective (Lemma 7.2.2), and check inclusion in $B_{L^{-1}i}$. The complement set of all β's such that $u_i \not\sqsubseteq \beta v_i$ is also regular and effective, and one easily derives the corresponding $u_i \ominus \beta v_i$'s (a finite set of suffixes of u_i), hence checking condition *4b*. □

Proof.[of Theorem 7.0.8] Since $\mathsf{PEP}^{\mathrm{reg}}$ is r.e., it is sufficient to prove that it is also co-r.e. For this we observe that, by Propositions 7.1.5 and 7.1.6, a $\mathsf{PEP}^{\mathrm{reg}}$ instance is negative if, and only if, there exists a stable family $(A_L, B_L)_{L \in \mathcal{L}(R)}$ satisfying $\epsilon \in A_R$. One can effectively enumerate all families $(A_L, B_L)_{L \in \mathcal{L}(R)}$ of regular languages and check whether they are stable (Proposition 7.2.3) and have $\epsilon \in A_R$. If the $\mathsf{PEP}^{\mathrm{reg}}$ instance is negative, this procedure will eventually terminate, e.g., when it considers the blocker family. □

We remark that, when the above procedure terminates in the case of a negative instance, it is not guaranteed that the stable family it has found is indeed the blocker family. In fact, there is no way to tell that a stable family is the blocker family as will be seen in section 8.3.

Chapter 8

Languages of PEP blockers

From the direct algorithm to $\mathsf{PEP}^{\mathrm{reg}}$, we will principally remember the notion of blockers and coblockers as they open a whole range of possible problems. We will now explore some of the immediate ones on blocker/coblockers set or their complements: comparisons with regular languages and finiteness.

In this chapter, we will always consider a generic PEP instance given by some $u, v : \Sigma^* \to \Gamma^*$.

Blockers and coblockers. Recall the definition of blocker and coblocker sets as defined in the previous chapter.

Write Sol_L for the set $\{\sigma \in L \mid u_\sigma \sqsubseteq v_\sigma\}$ of solutions in some *constraint* language $L \subseteq \Sigma^*$ and define:

$$X_L \stackrel{\text{def}}{=} \{\alpha \in \Gamma^* \mid \forall \sigma \in L, \alpha.u_\sigma \not\sqsubseteq v_\sigma\}, \qquad \text{(left } L\text{-blockers)}$$

$$X'_L \stackrel{\text{def}}{=} \{\alpha \in \Gamma^* \mid \forall \sigma \in L, u_\sigma.\alpha \not\sqsubseteq v_\sigma\}, \qquad \text{(right } L\text{-blockers)}$$

$$Y_L \stackrel{\text{def}}{=} \{\beta \in \Gamma^* \mid \forall \sigma \in L, u_\sigma \not\sqsubseteq \beta.v_\sigma\}, \qquad \text{(left } L\text{-coblockers)}$$

$$Y'_L \stackrel{\text{def}}{=} \{\beta \in \Gamma^* \mid \forall \sigma \in L, u_\sigma \not\sqsubseteq v_\sigma.\beta\}. \qquad \text{(right } L\text{-coblockers)}$$

Right blockers and coblockers are defined for sake of completeness. We will not elaborate on those, it is equivalent to consider left blockers on mirror problems.

A key observation is that, in order to decide whether Sol_L is empty or not, it is simpler to reason about blocker and coblocker sets. Rather than considering what are the solutions, the blocker and coblocker sets provide information on what latitude is allowed/required by the solutions, in particular by the most permissive ones. The decision algorithm presented in chapter 7 elaborate on the particular case where we asks for the presence of ϵ in blockers languages.

$$Sol_L = \emptyset \text{ iff } \epsilon \in X_L \text{ iff } \epsilon \in X'_L \text{ iff } \epsilon \in Y_L \text{ iff } \epsilon \in Y'_L. \tag{8.1}$$

Working with blocker sets rather than solutions sets has two main advantages:

- First, blocker and coblocker sets behave smoothly as a function of the constraint set L. This allows compositional reasoning w.r.t. L. For instance, assume L is the product (concatenation) of two languages: $L = L_1.L_2$. Clearly Sol_L contains $Sol_{L_1}.Sol_{L_2}$. However the containment is strict in general, and it is not possible to express Sol_L as a function of Sol_{L_1} and Sol_{L_2}. By contrast, (see App. 8.6)

$$X_{L_1.L_2} = \Gamma^* \text{ iff } \left(X'_{L_1} \cup Y_{L_2}\right) \cap \left(Y'_{L_1} \cup X_{L_2}\right) = \Gamma^*. \tag{8.2}$$

- Second, blocker and coblocker sets are always regular languages, unlike the Sol_L sets (illustrated in the next chapter). This makes them easier to handle algorithmically, representing them via FSA's or regular expressions. In particular, compositional reasoning as exemplified in Equation (8.2) can easily be turned into simple and effective algorithms.

We will consider the computability of the blocker and coblocker sets X_R and Y_R for R a regular constraint language. We prove that blocker sets are not computable[1] while, quite unexpectedly, coblocker sets are computable. Concerning blocker sets, and since they cannot be computed, we consider decision problems that are weaker than computability, e.g., whether a blocker set is empty, infinite, whether is it contained in ("safety"), or contains ("cosafety"), a given set. A summary of the results of this chapter will be found in Fig. 8.1.

Outline of the chapter. Section 8.1 formally introduces the problems we address. Then Section 8.2 shows how to compute coblocker sets, while Section 8.3 considers what can be computed on blocker sets. The undecidability results in that section are proved by a reduction from lossy counter machines described in Section 8.4.

8.1 Blockers and coblockers

Recall that, for a regular constraint set $R \subseteq \Sigma^*$, X_R is upward-closed and Y_R is downward-closed. Hence both are regular.

For blocker and coblocker sets, we consider questions that range in generality from just checking one α for membership, to computing the whole set.

[1]Here, and in the rest of the chapter, we say informally that regular sets like X_L are "*computable*" when we really mean that an index for them can be computed uniformly from an index for L.

Definition 8.1.1 (Decision problems for blocker and coblocker sets) *We consider questions where one is given two morphisms $u, v : \Sigma^* \to \Gamma^*$ and a regular language $R \subseteq \Sigma^*$ as inputs, with possibly some additional input in the form of a word $\alpha \in \Gamma^*$, or a regular "safe" set $S \subseteq \Gamma^*$.*

- Blockers_Membership*: does $\alpha \in X_R$?*
- Blockers_Emptiness*: does $X_R = \emptyset$?*
- Blockers_Universality*: does $X_R = \Gamma^*$?*
- Blockers_Safety*: does $X_R \subseteq S$?*
- Blockers_Cosafety*: does $S \subseteq X_R$?*
- Blockers_Finiteness*: is X_R finite?*
- Blockers_Cofiniteness*: is X_R cofinite?, i.e., is $\Gamma^* \smallsetminus X_R$ finite?*

The same decision problems CoBlockers_Membership, CoBlockers_Safety, *..., are defined for coblocker sets.*

Finally, Blockers_Computable *and* CoBlockers_Computable *ask one to* compute *a representation of X_R (resp., Y_R) under the form of a regular expression or a FSA. (These are not decision problems).*

Remark 8.1.2 *The restriction to* regular *safe sets S is a natural assumption that is both expressive and tractable. However, in our setting where blocker and coblocker sets are upward-closed (resp., downward-closed), the expressive power is even larger. Indeed, for any L, $X_R \subseteq L$ iff $X_R \subseteq S$ where S is the upward-closure of L. Thus, and since the upward-closure of L is always regular, our positive results automatically apply to any class of safe sets for which the upward and downward closures can be effectively computed (e.g., context-free languages [Lee78]).*

Remark 8.1.3 (Relations among problems) Safety *is a general problem that subsumes* Emptiness *and* Membership. Cosafety *subsumes* Universality *and (non-)*Membership. Blockers_Universality *reduces to* Blockers_Membership *since $X_R = \Gamma^*$ iff $\epsilon \in X_R$.* CoBlockers_Universality *is trivial since $Y_R = \Gamma^*$ iff $R = \emptyset$.* Finiteness *and* Cofiniteness *are natural counting questions.* Finiteness *coincides with* Emptiness *for blocker sets (assuming Γ is not empty) and more generally for all upward-closed sets (*Cofiniteness *and* Universality *coincide for downward-closed sets in general, and coblocker sets in particular).*

There are no other obvious reductions between the above decision problems (e.g., Finiteness *and* Cofiniteness *are in general unrelated).*

Regarding computability of the blocker and coblocker sets, observe that since these sets are regular, the decidability of Safety *and* Cosafety *would entail their computability (see also Section 8.2). Conversely, all the decision problems listed above can easily be answered from an FSA description of the sets. Hence our decision problems can be seen as different special cases of the general* Blockers_Computable *and* CoBlockers_Computable *problems.*

	Blockers	Coblockers
Membership	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.2)	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.6)
Safety	undecidable (Theo. 8.3.3)	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.6)
Cosafety	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.2)	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.6)
Emptiness	undecidable (Theo. 8.3.3)	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.6)
Universality	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.2)	trivial
Finiteness	undecidable (Theo. 8.3.3)	$\mathsf{F}_{\omega^\omega}$-complete (Coro. 8.2.6)
Cofiniteness	undecidable (Theo. 8.3.2)	trivial
Computable	no	yes (Coro. 8.2.6)

Figure 8.1: Computability for blocker and coblocker sets. See Remark 8.1.4 about complexity.

Remark 8.1.4 (On lower bound of blocker and coblocker problems) *All the non-trivial problems listed in Def. 8.1.1 are more general than* $\mathsf{PEP}^{\mathrm{reg}}$*. This was made precise in Remark 8.1.3 except for* CoBlockers_Finiteness*, but it is easy to provide a reduction from* CoBlockers_Emptiness *to* CoBlockers_Finiteness*: add one extra symbol to* Γ*, ensuring that* Y_R *is finite iff it is empty. Hence all the above problems are at least as hard as* $\mathsf{PEP}^{\mathrm{reg}}$*, i.e.* $\mathsf{F}_{\omega^\omega}$*-hard.*

8.2 Upper bound results

8.2.1 On blockers sets

We start with the computability results. They can be obtained via reductions to $\mathsf{PEP}^{\mathrm{reg}}$:

Lemma 8.2.1 Blockers_Cosafety *many-one reduce to (the complement of)* $\mathsf{PEP}^{\mathrm{reg}}$*.*

Proof. with u, v, R and S we associate a $\mathsf{PEP}^{\mathrm{reg}}$ instance $u', v' : \Sigma'^* \to \Gamma^*$ and a regular constraint $R' \subseteq \Sigma'^*$. Assume w.l.o.g. that Σ and Γ are disjoint alphabets and let $\Sigma' \stackrel{\text{def}}{=} \Sigma \cup \Gamma$. u' and v' are extensions of u and v with $u'(\gamma) = \gamma$ and $v'(\gamma) = \epsilon$ for all $\gamma \in \Gamma$. Finally let $R' \stackrel{\text{def}}{=} S.R$, this is indeed a regular subset of Σ'^*.

Now, u', v', R' is a positive $\mathsf{PEP}^{\mathrm{reg}}$ instance iff $u'_x \sqsubseteq v'_x$ for some $x \in R'$, iff $u'_{\alpha y} \sqsubseteq v'_{\alpha y}$ for some $\alpha \in S$ and some $y \in R$, iff $u'_\alpha.u'_y \sqsubseteq v'_\alpha.v'_y$, iff $\alpha.u_y \sqsubseteq v_y$ for some $\alpha \in S$ and y, i.e., iff some $\alpha \in S$ is not in X_R, i.e., $S \not\subseteq X_R$. □

Since $\mathsf{PEP}^{\mathrm{reg}}$ is decidable, and thanks to Remark 8.1.4, Lemma 8.2.1 entails:

Corollary 8.2.2 Blockers_Cosafety, Blockers_Universality *and* Blockers_Membership *are* $\mathsf{F}_{\omega^\omega}$*-complete.*

8.2.2 On coblockers sets

Lemma 8.2.3 (Elimination Lemma, proof in App. A.1) *If* $xw \sqsubseteq y$ *and* $x' \sqsubseteq wy'$ *then* $xx' \sqsubseteq yy'$.

Lemma 8.2.4 (proof in App. A.3) $x \not\sqsubseteq y$ *and* $xx' \sqsubseteq yy'$ *imply* $(x \ominus y)x' \sqsubseteq y'$.

let be $s = \max_{i\in\Sigma}(|u_i|, |v_i|)$ and n be the size of the syntactic congruence $\sim_R$ of R, $r = s.n$.

Lemma 8.2.5 *The size of the biggest element of the basis of* $\Gamma^* \setminus Y'_R$ *is bounded by* $F_{\omega^\omega}(f(|\Gamma|, r))$*, for some primitive recursive function* f.

We choose to look at Y'_R instead of Y_R only to allow us to use our results on $\ominus$. The result is naturally also valid for Y_R. Let write $B(|\Gamma|, r)$ for the bound $F_{\omega^\omega}(f(|\Gamma|, r))$.

Proof. Let β be an element of the basis of $\Gamma^* \setminus Y'_R$. then there is some shortest $\sigma \in R$ such that $u_\sigma \sqsubseteq v_\sigma.\beta$. β being an element of the basis it is the smallest such word, so $\beta = u_\sigma \ominus v_\sigma$.

Let σ_i be the prefix of σ such that $u_{\sigma_i} \not\sqsubseteq v_{\sigma_i}$ and let $w_i = u_{\sigma_i} \ominus v_{\sigma_i}$. Notice that $(w_i)_i$ is controlled by $Succ^s$.

To exhibit a contradiction, suppose that $|\beta| > B(|\Gamma|, r)$. Then, $(w_i)_i$ is a long enough controlled sequence of words such that it is n-bad. Hence there are $i < j$ such that $\sigma_i \sim_R \sigma_j$ and $w_i \sqsubseteq w_j$.

Let x be the prefix of σ such that $\sigma = \sigma_j.x$. Knowing that $u_{\sigma_j} \not\sqsubseteq v_{\sigma_j}$ and $u_{\sigma_j}.u_x \sqsubseteq v_{\sigma_j}.v_x.\beta$, we can apply Lemma 8.2.4, which gives us that $u_{\sigma_j} \ominus v_{\sigma_j}.u_x \sqsubseteq v_y.\beta$. Then $w_i.u_x \sqsubseteq v_y.\beta$ and by Lemma 8.2.3, we obtain that $u_{\sigma_i}.u_x \sqsubseteq v_{\sigma_i}.v_x.\beta$ and $\sigma_i.x \in R$, which contradict the assumption of minimality of σ. □

Corollary 8.2.6 *1.* Y_R *and* Y'_R *are computable in time* $F_{\omega^\omega}(g(|\Gamma|, r))$ *for some primitive recursive function* g.
2. CoBlockers_Membership, CoBlockers_Emptiness, CoBlockers_Safety, CoBlockers_Cosafety, CoBlockers_Finiteness *are* $\mathsf{F}_{\omega^\omega}$*-complete.*

Proof. 1. Lemma 8.2.5 tells us that in order to compute a finite basis of Y_R, it is sufficient to do this on words smaller than $B(|\Gamma|, r)$. A simple brute force algorithm can do this in time $O(2^{|\Gamma|^{B(|\Gamma|,r)}})$, hence there is some g primitive recursive such that Y_R is computable in time $F_{\omega^\omega}(g(|\Gamma|, r))$.
2. Knowing a finite representation of Y_R as a FSA yields directly an algorithm to those problems in time at worst exponential in the sizes of the

automatons of Y_R and S. The completeness comes from all those problems being more general than $\mathsf{PEP}^{\mathrm{reg}}$ (See Remark 8.1.4). □

8.3 Blocker sets are not computable

It is not possible to effectively compute the blocker sets X_R from given u, v, R, even though X_R is known to be regular. This is shown with Lemma 8.3.1, our main negative result (proved in Section 8.4):

Lemma 8.3.1 Blockers_Cofiniteness *is* Σ_1^0*-hard and* Blockers_Emptiness *is* Π_1^0*-hard.*

With Lemma 8.3.1, we are in a position to prove all the undecidability results in Fig. 8.1:

Theorem 8.3.2 Blockers_Cofiniteness *is* Σ_1^0*-complete.*

Proof.[Sketch] Membership in Σ_1^0 can be seen by writing the cofiniteness of X_R under the form $\exists n \in \mathbb{N}, \Gamma^{\geq n} \subseteq X_R$ and relying on the decidability of Blockers_Cosafety (Coro. 8.2.2). □ □

Theorem 8.3.3 Blockers_Safety*,* Blockers_Emptiness *and* Blockers_Finiteness *are* Π_1^0*-complete.*

Proof. The Π_1^0-hardness of Blockers_Emptiness (Lemma 8.3.1) also applies to Blockers_Finiteness (since the two problems coincide) and Blockers_Safety (a more general problem), see Remark 8.1.3.

For upper bounds, we observe that Blockers_Safety (hence also Blockers_Emptiness) is in Π_1^0 since it can be written under the form $\forall \alpha \in \Gamma^*, (\alpha \in S \vee \alpha \notin X_R)$ (recall that $\alpha \notin X_R$ is decidable). □ □

8.4 Lossy counter machines

Lossy counter machines or, for short, *LCM*'s, were introduced by R. Mayr [May03b]. They are a variant of Minsky counter machines (with zero-test, increments and decrements) where counters are *lossy*, i.e., they may decrease non-deterministically. We only give a streamlined presentation of LCM's here and refer to [May03b, Sch10] for more details.

Let $M = (Q, C, \Delta, q_{\mathrm{init}})$ be a Minsky counter machine with finite set of control states $Q \ni q_{\mathrm{init}}$, finite set of counters C, and finite set of transitions rules Δ. Four counters are sufficient for our purposes so we fix $C = \{c_1, c_2, c_3, c_4\}$. A configuration of M is some $\tau = (q, n_1, n_2, n_3, n_4) \in$

$Conf(M) \stackrel{\text{def}}{=} Q \times \mathbb{N}^4$, with *size*, denoted $|\tau|$, being $n_1 + n_2 + n_3 + n_4$. We (partially) order $Conf(M)$ with

$$(q, n_1, n_2, n_3, n_4) \leq (q', n_1', n_2', n_3', n_4') \stackrel{\text{def}}{\Leftrightarrow} q = q' \wedge n_1 \leq n_1' \wedge \cdots \wedge n_4 \leq n_4'.$$

An initial state $q_{\text{init}} \in Q$ is fixed, and the initial configuration is $\tau_{\text{init}} \stackrel{\text{def}}{=} (q_{\text{init}}, 0, 0, 0, 0)$. Observe that the only way to have $\tau \leq \tau_{\text{init}}$ is with $\tau = \tau_{\text{init}}$.

A transition rule δ is a directed edge between states of M, labeled by an operation $op \in OP \stackrel{\text{def}}{=} C \times \{\texttt{++}, \texttt{--}, \texttt{=0?}\}$, and denoted (q, op, q'). The rules in Δ give rise to two different transition relations between configurations. First, steps $\tau \stackrel{\delta}{\rightarrow} \tau'$ are defined in the expected way. Formally, with $\delta = (q_1, op, q_2)$, there is a step $(q, n_1, n_2, n_3, n_4) \stackrel{\delta}{\rightarrow} (q', n_1', n_2', n_3', n_4')$ if, and only if, the following three conditions are satisfied:
1. $q_1 = q$ and $q_2 = q'$;
2. op is some c_k++ or c_k-- or c_k=0?, and $n_i' = n_i$ for all $i \neq k$;
3. if op is c_k++ then $n_k' = n_k + 1$; if op is c_k-- then $n_k' = n_k - 1$; if op is c_k=0? then $0 = n_k = n_k'$.

These so-called *perfect steps* describe the operational semantics of M when its counters are not assumed to be lossy. Then a second operational semantics, with transitions denoted $\tau \stackrel{\delta}{\rightarrow}_{\text{sl}} \tau'$, is derived[2] in the following way:

$$\tau \stackrel{\delta}{\rightarrow}_{\text{sl}} \tau' \stackrel{\text{def}}{\Leftrightarrow} \tau \stackrel{\delta}{\rightarrow} \tau'' \text{ for some } \tau'' \geq \tau'. \tag{8.3}$$

These *lossy steps* describe the behavior of M when its counters are assumed to be lossy. In the usual way, the δ superscript on transitions is omitted when irrelevant. *Lossy runs*, denoted $\tau_0 \stackrel{*}{\rightarrow}_{\text{sl}} \tau_n$, are sequences of chained lossy steps $\tau_0 \rightarrow_{\text{sl}} \tau_1 \rightarrow_{\text{sl}} \cdots \tau_n$. We write $Reach_{\text{lossy}}(M)$ for the set of configurations that can be reached via lossy runs of M, starting from τ_{init}.

We rely on known undecidability results on LCM's and use the following two problems:

LCM_Infinite: the question whether $Reach_{\text{lossy}}(M)$ is infinite, for a given LCM M;

LCM_Unbounded_Counter: the question whether $Reach_{\text{lossy}}(M)$ contains configurations with arbitrarily large values for the first counter c_1.

These two problems are a variant of one another, and they are easily seen to be inter-reducible. The following theorem is from [May03b, Sch10]:

Theorem 8.4.1 LCM_Infinite *and* LCM_Unbounded_Counter *are* Π_1^0*-complete.*

[2]Lossy steps could also be defined *directly* without deriving them from perfect steps, but the indirect definition is very convenient as it permits reasoning simultaneously on both kinds of steps for the same counter machine.

8.4.1 From lossy counters to Post-embedding

With a LCM $M = (Q, C, \Delta, q_{\text{init}})$ we associate a PEP instance $u, v : \Sigma^* \to \Gamma^*$ that will be used in three different reductions (with different constraint languages $R_1, R_2, R_3 \subseteq \Sigma^*$). Here $\Gamma \stackrel{\text{def}}{=} Q \cup C$ is used to encode the configurations of M: a configuration $\tau = (q, n_1, n_2, n_3, n_4)$ is encoded by the word $c_1^{n_1} c_2^{n_2} c_3^{n_3} c_4^{n_4} q$, denoted $\lceil \tau \rceil$. Observe that $\lceil \tau \rceil \sqsubseteq \lceil \tau' \rceil$ iff $\tau \leq \tau'$.

We further let $\Sigma \stackrel{\text{def}}{=} \Gamma \cup \Delta \cup OP \cup \overline{Q} \cup \overline{C}$ where $\overline{Q} = \{\overline{q} \mid q \in Q\}$ and $\overline{C} = \{\overline{c}_1, \overline{c}_2, \overline{c}_3, \overline{c}_4\}$ are copies of Q and C, with new symbols obtained by overlining the original symbols from $Q \cup C$. We define two morphisms $u, v : \Sigma^* \to \Gamma^*$ with

$$u((q, op, q')) \stackrel{\text{def}}{=} q, \quad v((q, op, q')) \stackrel{\text{def}}{=} q', \qquad u(\overline{c}_i) \stackrel{\text{def}}{=} c_i, \qquad v(\overline{c}_i) \stackrel{\text{def}}{=} c_i,$$
$$u(c_i\texttt{++}) \stackrel{\text{def}}{=} \epsilon, \qquad v(c_i\texttt{++}) \stackrel{\text{def}}{=} c_i, \quad u(c_i\texttt{--}) \stackrel{\text{def}}{=} c_i, \quad v(c_i\texttt{--}) \stackrel{\text{def}}{=} \epsilon.$$

How u and v evaluate on the rest of Σ will be defined later when it becomes relevant.

With every transition rule $\delta = (q, op, q')$ in Δ, we associate a language $R_\delta \subseteq \Sigma^*$ given via the following regular expressions:

$$R_\delta \stackrel{\text{def}}{=} \begin{cases} \overline{c}_1{}^* \cdots \overline{c}_{k-1}{}^* \cdot op \cdot \overline{c}_k{}^* \cdots \overline{c}_4{}^* \cdot \delta & \text{if } op \text{ is } c_k\texttt{++} \text{ or } c_k\texttt{--}, \\ \overline{c}_1{}^* \cdots \overline{c}_{k-1}{}^* \cdot \overline{c}_{k+1}{}^* \cdots \overline{c}_4{}^* \cdot \delta & \text{if } op \text{ is } c_k\texttt{=0?}. \end{cases}$$

These definitions ensure that, when $x \in R_\delta$, u_x and v_x are the encodings of related configurations. We let the reader check that the following more precise statement holds:

Lemma 8.4.2
1. If $x \in R_\delta$, then $u_x = \lceil \tau \rceil$ and $v_x = \lceil \tau' \rceil$ for some configurations τ, τ' such that $\tau \xrightarrow{\delta} \tau'$.
2. Reciprocally, if $\tau \xrightarrow{\delta} \tau'$, then $\lceil \tau \rceil = u_x$ and $\lceil \tau' \rceil = v_x$ for some (unique) $x \in R_\delta$.

We further define $R_\Delta \stackrel{\text{def}}{=} \bigcup_{\delta \in \Delta} R_\delta$ and $R_M \stackrel{\text{def}}{=} (R_\Delta)^*$: these languages are regular.

Lemma 8.4.3 *Let $\alpha \in \Gamma^*$. If $u_x.\alpha \sqsubseteq \lceil \tau_{init} \rceil . v_x$ for some $x \in R_M$, then $\alpha \sqsubseteq \lceil \tau \rceil$ for some $\tau \in Reach_{lossy}(M)$.*

Proof. We assume $\alpha \neq \epsilon$ and $x \neq \epsilon$, otherwise $\alpha \sqsubseteq \lceil \tau_{\text{init}} \rceil$ trivially. Thus $x \in R_M$ must be of the form $x = x_1 \dots x_n$ with $n > 0$ and $x_i \in R_\Delta$ for all $i = 1, \dots, n$. By Lemma 8.4.2, u_x is some $\lceil \tau_0 \rceil . \lceil \tau_1 \rceil \dots \lceil \tau_{n-1} \rceil$ and v_x is some $\lceil \tau'_1 \rceil . \lceil \tau'_2 \rceil \dots \lceil \tau'_n \rceil$ such that, for all $i = 1, \dots, n$, $\tau_{i-1} \to \tau'_i$ is a perfect step of M.

We now use the assumption that $u_x.\alpha \sqsubseteq \lceil \tau_{\text{init}} \rceil.v_x$. Since $\alpha \neq \epsilon$, u_x embeds into a strict prefix, denoted w, of $\lceil \tau_{\text{init}} \rceil.v_x$. Note that u_x contains $n > 0$ symbols from Q and ends with one of them, while w has at most n (it is shorter than $\lceil \tau_{\text{init}} \rceil.v_x$ that has $n+1$ symbols from Q and ends with one of them). Hence w necessarily has n symbols from Q and $u_x.\alpha \sqsubseteq \lceil \tau_{\text{init}} \rceil.v_x$ can be decomposed as $\lceil \tau_i \rceil \sqsubseteq \lceil \tau_i' \rceil$ (i.e., $\tau_i \leq \tau_i'$) for all $i = 1, \ldots, n-1$, with also $\lceil \tau_0 \rceil \sqsubseteq \lceil \tau_{\text{init}} \rceil$ (hence $\tau_0 = \tau_{\text{init}}$) and $\alpha \sqsubseteq \lceil \tau_n' \rceil$. Combining with $\tau_{i-1} \rightarrow \tau_i'$ we deduce $\tau_{i-1} \rightarrow_{\text{sl}} \tau_i$ for $i = 1, \ldots, n-1$. Finally $\tau_{\text{init}} = \tau_0 \rightarrow_{\text{sl}} \tau_1 \cdots \rightarrow_{\text{sl}} \tau_{n-1} \rightarrow \tau_n'$ is a lossy run of M, so that $\tau_n' \in \mathit{Reach}_{\text{lossy}}(M)$. □ □

There is a converse to Lemma 8.4.3:

Lemma 8.4.4 *If $\tau \in \mathit{Reach}_{\mathit{lossy}}(M)$, there exists some $x \in R_M$ such that $u_x.\lceil \tau \rceil \sqsubseteq \lceil \tau_{\mathit{init}} \rceil.v_x$.*

Proof. Since $\tau \in \mathit{Reach}_{\text{lossy}}(M)$ there exists a lossy run $\tau_{\text{init}} = \tau_0 \rightarrow_{\text{sl}} \tau_1 \rightarrow_{\text{sl}} \cdots \tau_n = \tau$. We show, by induction on $i = 0, 1, \ldots, n$, that $u_{x_i}.\lceil \tau_i \rceil \sqsubseteq \lceil \tau_{\text{init}} \rceil.v_{x_i}$ for some $x_i \in R_M$.

The base case, $i = 0$, is dealt with $x_0 = \epsilon$ since $\tau_0 = \tau_{\text{init}}$.

For the case $i > 0$, we know by ind. hyp. that there is some $x_{i-1} \in R_M$ with

$$u_{x_{i-1}}.\lceil \tau_{i-1} \rceil \sqsubseteq \lceil \tau_{\text{init}} \rceil.v_{x_{i-1}}. \tag{8.4}$$

The lossy step $\tau_{i-1} \rightarrow_{\text{sl}} \tau_i$ implies the existence of a perfect step $\tau_{i-1} \rightarrow \tau'$ with $\tau' \geq \tau_i$ (Equation (8.3)). Thus $\lceil \tau_{i-1} \rceil = u_y$ and $\lceil \tau' \rceil = v_y$ for some $y \in R_\Delta$ (Lemma 8.4.2).

From $\tau_i \leq \tau'$, we deduce

$$u_y.\lceil \tau_i \rceil \sqsubseteq \lceil \tau_{i-1} \rceil.v_y. \tag{8.5}$$

We now put together Equations (8.4) and (8.5). The Elimination Lemma yields

$$u_{x_{i-1}}.u_y.\lceil \tau_i \rceil \sqsubseteq \lceil \tau_{\text{init}} \rceil.v_{x_{i-1}}.v_y, \tag{8.6}$$

so that setting $x_i \stackrel{\text{def}}{=} x_{i-1}.y$ concludes our proof. We observe that $x_i \in R_M$ since $x_{i-1} \in R_M$ and $y \in R_\Delta$. □ □

8.4.2 Reducing LCM_Infinite and LCM_Unbounded_Counter to blockers problems

For the next step in the reduction, we extend u and v on $Q \cup C$ $(= \Gamma)$ with

$$u(\gamma) \stackrel{\text{def}}{=} \pi_1(\gamma) = \begin{cases} c_1 & \text{if } \gamma = c_1, \\ \epsilon & \text{if } \gamma \in \Gamma \setminus \{c_1\}, \end{cases} \qquad v(\gamma) \stackrel{\text{def}}{=} \gamma \text{ for all } \gamma \in \Gamma.$$

When $\alpha \in \Gamma^*$, we shall write $\pi_1(\alpha)$ rather than u_α to emphasize the fact that u only retains the c_1 symbols of α and erases the rest. Below, we rely on a few obvious properties of this erasing morphism, such as $\pi_1(\alpha) \sqsubseteq \alpha$, or $\pi_1(\alpha\beta) = \pi_1(\beta\alpha)$, and in particular the following:

Fact 8.4.5 *For all $\beta \in \Gamma^*$ and $x, y \in \Sigma^*$, $x.c_1.\pi_1(\beta) \sqsubseteq y.\beta$ implies $x.c_1 \sqsubseteq y$.*

Finally, we let $R_1 \stackrel{\text{def}}{=} q_{\text{init}}.R_M$ and $R_2 \stackrel{\text{def}}{=} R_1.\Gamma^*$. This provides two different reductions, with properties captured by Lemmas 8.4.6 and 8.4.8.

Lemma 8.4.6 *Let $\alpha \in \Gamma^*$. The following are equivalent:*
(1) $\alpha \notin X'_{R_1}$,
(2) there exists $x \in R_1$ such that $u_x.\alpha \sqsubseteq v_x$,
(3) there exists $\tau \in Reach_{lossy}(M)$ such that $\alpha \sqsubseteq \lceil\tau\rceil$.

Proof.[Sketch] (1) $\Leftrightarrow$ (2) by definition of X'_{R_1}. Then, given the definitions of R_1, u and v, Lemma 8.4.3 shows "(2) $\Rightarrow$ (3)" (note that $u(q_{\text{init}}) = \epsilon$ and $v(q_{\text{init}}) = q_{\text{init}} = \lceil\tau_{\text{init}}\rceil$). Finally, Lemma 8.4.4 shows "(3) $\Rightarrow$ (2)". □ □
In particular, X'_{R_1} is cofinite iff M does not satisfy LCM_Infinite.

Corollary 8.4.7 Blockers_Cofiniteness *is Σ^0_1-hard.*

Lemma 8.4.8 *Let $\alpha \in \Gamma^*$. The following are equivalent:*
(1) $\alpha \notin X'_{R_2}$,
(2) there exists $y \in R_2$ such that $u_y.\alpha \sqsubseteq v_y$,
(3) there exists $\tau \in Reach_{lossy}(M)$ such that $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$.

Proof. (1) $\Leftrightarrow$ (2) by definition of X'_{R_2}.
(3) $\Rightarrow$ (2): Assume $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$ for some $\tau \in Reach_{\text{lossy}}(M)$. Then, $\pi_1(\alpha) \sqsubseteq \lceil\tau\rceil$ so that, by Lemma 8.4.6, there exists some $x \in R_1$ with $u_x.\pi_1(\alpha) \sqsubseteq v_x$. Appending α to the right yields $u_x.\pi_1(\alpha).\alpha = u_x.u_\alpha.\alpha \sqsubseteq v_x.\alpha = v_x.v_\alpha$. Letting $y \stackrel{\text{def}}{=} x.\alpha$ ($\in R_2$) proves (2).
(2) $\Rightarrow$ (3): Assume $u_y.\alpha \sqsubseteq v_y$ for some $y \in R_2$ of the form $x.\beta$ with $x \in R_1$ and $\beta \in \Gamma^*$. We assume $\pi_1(\alpha) \neq \epsilon$ since otherwise $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau_{\text{init}}\rceil)$ holds trivially. From $u_y.\alpha \sqsubseteq v_y$, we deduce

$$u_x.\pi_1(\alpha).\pi_1(\beta) = u_x.\pi_1(\beta).\pi_1(\alpha) = u_y.\pi_1(\alpha) \sqsubseteq u_y.\alpha \sqsubseteq v_y = v_x.v_\beta = v_x.\beta.$$

From $u_x.\pi_1(\alpha).\pi_1(\beta) \sqsubseteq v_x.\beta$, one deduces $u_x.\pi_1(\alpha) \sqsubseteq v_x$ (using Fact 8.4.5 and the assumption that $\pi_1(\alpha) \neq \epsilon$). Thus there exists a $\tau \in Reach_{\text{lossy}}(M)$ with $\pi_1(\alpha) \sqsubseteq \lceil\tau\rceil$ (Lemma 8.4.3), hence $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$. □ □
In other words, $\alpha \notin X'_{R_2}$ iff there is a reachable configuration where the c_1 counter is larger than, or equal to, the number of c_1 symbols in α. Thus $X'_{R_2} = \emptyset$ iff M satisfies LCM_Unbounded_Counter.

Corollary 8.4.9 Blockers_Emptiness *is Π^0_1-hard.*

8.5 Regularity of Post-embedding languages is undecidable

As an aside, the reduction from LCM's can be used to prove Theo. 8.5.2 below. The regularity problem for Post-embedding languages is a natural question since Sol_R is not always regular and, as will be shown in next chapter, since comparisons with a regular S are possible:

Theorem 8.5.1 (Proof at 9.2.2) *The questions, for $S \subseteq \Sigma^*$ a regular language, whether $S \subseteq Sol_R$, and whether $Sol_R \subseteq S$, are decidable.*

Theorem 8.5.2 *The question whether, for $u, v : \Sigma^* \to \Gamma^*$ and a regular $R \subseteq \Sigma^*$, Sol_R is a regular language, is Σ_1^0-complete.*

In this section we prove one half of Theorem 8.5.2, i.e., that the regularity of Sol_R is Σ_1^0-hard. The other half, membership in Σ_1^0, is a consequence of Theorem 8.5.1.

We consider the reduction from LCM_Infinite to PEP built in Section 8.4.1 and further extend u and v on $\overline{Q}$ with $u(\overline{q}) = q$ and $v(\overline{q}) = \epsilon$ for each $\overline{q} \in \overline{Q}$. We further define $R_3 = q_{\text{init}}.R_M.\overline{Q}$. In this framework, the following holds:

Lemma 8.5.3 *If $Reach_{lossy}(M)$ is finite, then Sol_{R_3} is regular.*

Proof.[Sketch] Any $x \in R_3$ has the form $q_{\text{init}}.x_1.x_2 \dots x_n.\overline{q}$ for some $q \in Q$ and some $x_1, x_2, \dots, x_n \in R_\Delta$. As seen in the proof of Lemma 8.4.3, such an x belongs to Sol_{R_3} if, and only if, there exists a lossy run

$$(\tau_{\text{init}} =)\tau_0 \xrightarrow{\delta_1}_{\text{sl}} \tau_1 \xrightarrow{\delta_2}_{\text{sl}} \cdots \xrightarrow{\delta_n}_{\text{sl}} \tau_n \qquad (*)$$

with $x_i \in R_{\delta_i}$, $u_{x_i} = \lceil \tau_{i-1} \rceil$ (and where the control state of τ_n is q). The assumption that $Reach_{\text{lossy}}(M)$ is finite implies that the set of lossy runs in (*), when viewed as sequences σ of the form $(\tau_0, \delta_1) \dots (\tau_{n-1}, \delta_n)$ over the (finite!) alphabet $Reach_{\text{lossy}}(M) \times \Delta$, is a regular language, as is the set of paths of any finite graph. Since there is a bijective correspondence between the x_i's and the pairs (τ_{i-1}, δ_i) (see Lemma 8.4.2), the set of all $x_1 \dots x_n$ that correspond to lossy runs is regular too, hence also Sol_{R_3}. □ □

We can prove a reciprocal of Lemma 8.5.3 if we restrict ourselves to *deflatable* counter machines. Formally, a counter machine M is deflatable if it contains among its transition rules, the so-called "deflating" rules $q \xrightarrow{c_i--} q$ for all states $q \in Q$ and counters $c_i \in C$.

Lemma 8.5.4 *If $Reach_{lossy}(M)$ is infinite and M is deflatable, then Sol_{R_3} is not regular.*

Proof.[Sketch] For the proof, we use the projection morphism $\pi_\Delta : \Sigma^* \to \Delta^*$ that erases all symbols not in Δ (recall that, in our reduction from LCM's to PEP, the set of rules Δ is a sub-alphabet of Σ) and we show that $\pi_\Delta(Sol_{R_3})$ is not regular, which is sufficient since morphisms preserve regularity.

Now, since $Reach_{\text{lossy}}(M)$ is infinite, for every $N \in \mathbb{N}$ there exists a reachable configuration τ_N having size N. From τ_N, N deflating steps are possible and not more. Thus, for each $N \in \mathbb{N}$, there is a lossy run of the form

$$\tau_0 \xrightarrow{\delta_{N,1}}_{\text{sl}} \cdots \xrightarrow{\delta_{N,k_N}}_{\text{sl}} \tau_N \underbrace{\xrightarrow{\text{defl}}_{\text{sl}} \cdots \xrightarrow{\text{defl}}_{\text{sl}}}_{N \text{ deflating steps}} \tau'_N.$$

With this lossy run one associates a word $y_N \in R_3$, exactly as in the proof of Lemma 8.5.3. Now $\pi_\Delta(y_N)$ is $\delta_{N,1} \dots \delta_{N,k_N}(\text{defl})^N$, i.e., some $Y_N(\text{defl})^N$ (for simplifying purposes, we assume π_Δ further projects all different deflating rules to a single one called just "defl").

If $\pi_\Delta(Sol_{R_3})$ is regular, the pumping lemma for regular language implies that, for N large enough, if $\pi_\Delta(Sol_{R_3})$ contains $Y_N(\text{defl})^N$, it also contains $Y_N(\text{defl})^N(\text{defl}^m)^*$ for some $m > 0$. But this is clearly impossible since it would imply the existence of lossy runs starting with the same k_N steps and ending with arbitrarily many deflating steps.

Hence neither $\pi_\Delta(Sol_{R_3})$, nor Sol_{R_3}, are regular. □ □

We conclude by observing that the restriction to deflatable counter machines is no loss of generality. Deflating rules mimic losses in counters, hence any counter machine can be turned into a deflatable one that has essentially the same behavior as long as one only considers the lossy semantics. In particular, the original machine and its deflatable version have exactly the same reachable configurations (via lossy runs).

Therefore, Lemmas 8.5.3 and 8.5.4 show that LCM_Infinite reduces to the question whether the solutions of a PEP^{reg} instance is a regular language. Hence the regularity of Sol_R is Σ_1^0-hard as announced.

8.6 Appendix

$$X_{L_1.L_2} = \Gamma^* \text{ iff } \left(X'_{L_1} \cup Y_{L_2}\right) \cap \left(Y'_{L_1} \cup X_{L_2}\right) = \Gamma^*. \tag{8.7}$$

The proof is clearer if the equation is written contrapositionally, under the form:

$$X_{L_1.L_2} \neq \Gamma^* \text{ iff } \left[X'_{L_1} \cup Y_{L_2} \neq \Gamma^* \text{ or } Y'_{L_1} \cup X_{L_2} \neq \Gamma^*\right]. \tag{8.2'}$$

Proof. ($\Leftarrow$:) assume that there exists some $\alpha \in \Gamma^*$ that does not belong to $X'_{L_1} \cup Y_{L_2}$ (the case $\alpha \notin Y'_{L_1} \cup X_{L_2}$ is symmetric). Hence $\alpha \notin X'_{L_1}$ and $\alpha \notin Y_{L_2}$. Therefore $u_{x_1}.\alpha \sqsubseteq v_{x_1}$ for some $x_1 \in L_1$, and $u_{x_2} \sqsubseteq \alpha.v_{x_2}$ for some

$x_2 \in L_2$. We deduce $u_{x_1} u_{x_2} \sqsubseteq v_{x_1} v_{x_2}$ (by the Elimination Lemma A.1.2). Hence taking $x = x_1 x_2$ shows $\epsilon \notin X_{L_1.L_2}$ and then $X_{L_1.L_2} \neq \Gamma^*$.

($\Rightarrow$:) If $X_{L_1.L_2} \neq \Gamma^*$ then, in particular, $\epsilon \notin X_{L_1.L_2}$ (since blocker sets are upward-closed) and there exists some $x \in L_1.L_2$ with $u_x \sqsubseteq v_x$. Writing x under the form $x = x_1 x_2$ with $x_1 \in L_1$ and $x_2 \in L_2$, we deduce $u_{x_1}.u_{x_2} \sqsubseteq v_{x_1}.v_{x_2}$. Thus, by Lemma 7.1.4 (Decomposition Lemma), there exists $w \in \Gamma^*$ such that either $u_{x_1}.w \sqsubseteq v_{x_1}$ and $u_{x_2} \sqsubseteq w.v_{x_2}$, or $u_{x_1} \sqsubseteq v_{x_1}.w$ and $w.u_{x_2} \sqsubseteq v_{x_2}$. In the first case, $w \notin X'_{L_1}$ and $w \notin Y_{L_2}$. In the second case $w \notin Y'_{L_1}$ and $w \notin X_{L_2}$. Thus $X'_{L_1} \cup Y_{L_2} \neq \Gamma^*$ or $Y'_{L_1} \cup X_{L_2} \neq \Gamma^*$. $\square$

Chapter 9

Languages of PEP solutions

In this chapter, we will study $PE(u,v) \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid u(x) \sqsubseteq v(x)\}$, the Post Embedding language.

Regular constraints and the set of PEP-solutions. The decidability of $\mathsf{PEP}^{\text{reg}}$ can be restated under the following form: it is decidable, given two morphisms $u, v : \Sigma^* \to \Gamma^*$ and a regular language $R \subseteq \Sigma^*$, whether the following holds:

$$\exists x \in R : u(x) \sqsubseteq v(x). \qquad \text{(Existence)}$$

In other words, one can decide whether $R \cap PE(u,v) \neq \emptyset$. However, this problem has very high complexity.

In this chapter, we prove the decidability of the following questions:

$$\forall x \in R : u(x) \sqsubseteq v(x), \qquad \text{(Universality)}$$
$$\exists^{\infty} x \in R : u(x) \sqsubseteq v(x), \qquad \text{(Infinity)}$$
$$\neg\exists^{\infty} x \in R : u(x) \not\sqsubseteq v(x). \qquad \text{(Cofiniteness)}$$

"Universality" asks whether all words in R are solutions. "Infinity" asks whether R contains infinitely many solutions x, while dually "Cofiniteness" asks whether all but finitely many $x \in R$ are solutions. Equivalently, these questions ask whether $R \subseteq PE(u,v)$, whether $R \cap PE(u,v) =_a \emptyset$, and whether $R \smallsetminus PE(u,v) =_a \emptyset$, writing $S =_a S'$ to denote the "quasi-equality" of two sets, i.e., equality up to a finite subset. As a consequence of these decidability results we can compute the number of words in R that are (respectively, that are not) solutions.

These results are obtained with the help of two pumping lemmas, one for sets of solutions and one for sets of "antisolutions", i.e., words x such that $u(x) \not\sqsubseteq v(x)$. These pumping lemmas are the more technically involved developments of this chapter. Proving them relies on two kinds of techniques: (1) combinatorics of words in presence of the subword relation and associated

operations, and (2) a miniaturisation of Higman's Lemma that gives effective bounds on the length of bad sequences.

On complexity. Aiming at simplicity, our main decidability proofs do not come with explicit statements regarding the computational complexity of the associated problems. The decidability proofs can be turned into deterministic algorithms with complexity in $\mathsf{F}_{\omega^\omega}$. Regarding lower bounds, it is clear that "Infinity" is at least as hard as $\mathsf{PEP}^{\mathrm{reg}}$. We do not know if the same lower bound holds for "Universality" and "Cofiniteness".

Outline of the chapter. Section 9.1 deals with combinatorics on words with subwords. Section 9.2 proves the decidability of comparisons with regular sets. Then our pumping lemma is stated in Section 9.3 and used in Section 9.4 for deciding finiteness, counting, and quasi-regular questions. Sections 9.5 and 9.6 prove the two halves of the pumping lemma.

9.1 Composing, decomposing, and iterating words and subwords

This section is devoted to the subword ordering and the way it interacts with concatenations and factorizations. It proves a few basic results, e.g., Lemma 9.1.7, that we have been unable to find in the technical literature [Lot83, Lot02]. All missing proofs can be found in App. A.

9.1.1 Available suffixes

Recall that, when $x \sqsubseteq y$, the "used prefix" is the shortest prefix y_1 of y such that $x \sqsubseteq y_1$. Then, writing $y = y_1y_2$, what remains, i.e., y_2, is called the "available suffix" and denoted $y \oslash x$. For example, $a\underline{bc}\underline{a}bc \oslash \underline{ba} = bc$. Note that $y \oslash x$ is only defined when $x \sqsubseteq y$.

Lemma 9.1.1 *$x \sqsubseteq y$ and $x' \sqsubseteq (y \oslash x)y'$ imply $xx' \sqsubseteq yy'$.*

Corollary 9.1.2 *$x \sqsubseteq y$ implies $x(y \oslash x) \sqsubseteq y$.*

Lemma 9.1.3 *$x \sqsubseteq y$ and $xx' \sqsubseteq yy'$ imply $x' \sqsubseteq (y \oslash x)y'$.*

9.1.2 Unmatched suffixes

Recall that, when $x \not\sqsubseteq y$, the "matched prefix" is the longest prefix x_1 of x s.t. $x_1 \sqsubseteq y$. Then, writing $x = x_1x_2$, what remains, i.e., x_2, is called the "unmatched suffix" and denoted $x \ominus y$. For example $\underline{aa}bcabc \ominus b\underline{a}c\underline{a} = bcabc$. Note that $x \ominus y$ is only defined when $x \not\sqsubseteq y$ (hence $x \ominus y \neq \epsilon$).

Lemma 9.1.4 *$x \not\sqsubseteq y$ and $xx' \not\sqsubseteq yy'$ imply $[(x \ominus y)x'] \ominus y' = xx' \ominus yy'$.*

Corollary 9.1.5 *$x \not\sqsubseteq y$ and $xx' \not\sqsubseteq yy'$ imply $(x \ominus y)x' \not\sqsubseteq y'$.*

Lemma 9.1.6 *$x \not\sqsubseteq y$ and $xx' \sqsubseteq yy'$ imply $(x \ominus y)x' \sqsubseteq y'$.*

9.1.3 Iterating factors

Lemma 9.1.7 *$xy \sqsubseteq yz$ if, and only if, $x^k y \sqsubseteq yz^k$ for all $k \in \mathbb{N}$.*

Lemma 9.1.8 *Assume $x \not\sqsubseteq y$, $xz \not\sqsubseteq yt$, and $x \ominus y \sqsubseteq xz \ominus yt$. Then for all $k \in \mathbb{N}$:*

$$xz^k \not\sqsubseteq yt^k. \tag{Z_k}$$

Furthermore, if we let $r_k \stackrel{def}{=} xz^k \ominus yt^k$, then for all $k \in \mathbb{N}$:

$$r_0 \sqsubseteq r_k \sqsubseteq r_{k+1}. \tag{R_k}$$

9.2 Regular properties of sets of PEP solutions

Given two morphisms $u, v : \Sigma^* \to \Gamma^*$, a word $x \in \Sigma^*$ is called a "*solution*" (of Post's Embedding Problem) when $u_x \sqsubseteq v_x$. Otherwise it is an "*antisolution*". We let $PE(u, v)$ denote the set of solutions (for given u and v). Note that ϵ is always a solution.

We consider questions where we are given a PEP instance u, v with $u, v : \Sigma^* \to \Gamma^*$ and a regular language $R \subseteq \Sigma^*$. The considered problems are
PEP_Inclusion: does $PE(u, v) \subseteq R$?
PEP_Containment: does $PE(u, v) \supseteq R$?
PEP_Equality: does $PE(u, v) = R$?

It is tempting to compare $PE(u, v)$ with another Post-embedding set, however:

Theorem 9.2.1 *The questions "does $PE(u, v) \cap PE(u', v') = \{\epsilon\}$?" and "does $PE(u, v) \subseteq PE(u', v')$?" are Π_1^0-complete.*

Proof. Π_1^0-hardness can be shown directly by reduction from PCP. For the first question, simply let $u' = v$ and $v' = u$. Then a common solution has $u_x \sqsubseteq v_x = u'_x \sqsubseteq v'_x = u_x$, i.e., $u_x = v_x$.

For the second question we use a more subtle encoding: assume w.l.o.g. that Γ contains two distinct symbols a, b and that $u_x \neq \epsilon$ when $x \neq \epsilon$. Let now $u'_x \stackrel{\text{def}}{=} (ab)^{|u_x|}$ and $v'_x \stackrel{\text{def}}{=} (ba)^{|v_x|}$. Thus $u'_x \sqsubseteq v'_x$ if, and only if, $x = \epsilon$ or $|u_x| < |v_x|$. Finally, $PE(u, v) \setminus PE(u', v')$ contains the non-trivial PCP solutions. □ □

Theorem 9.2.2 PEP_Inclusion, PEP_Containment *and* PEP_Equality *are decidable.*

Note that, while comparisons with a regular language are decidable, regularity itself is undecidable, at least in the more general form stated here:

Proposition 9.2.3 (Regularity is undecidable 8.5) *The question "is* $R \cap PE(u, v)$ *a regular language?" is* Σ_1^0*-complete.*

The remainder of this section proves Theorem 9.2.2.

We first observe that PEP_Inclusion and $\mathsf{PEP}^{\mathrm{reg}}$ are inter-reducible since (u, v, R) is a positive instance for PEP_Inclusion if, and only if, $(u, v, \Sigma^* \setminus R)$ is a negative instance for $\mathsf{PEP}^{\mathrm{reg}}$. Hence the decidability of PEP_Inclusion follows from the decidability of $\mathsf{PEP}^{\mathrm{reg}}$.

For the decidability of PEP_Containment (and then of PEP_Equality), we fix an instance (u, v, R).

For a word $x \in \Sigma^*$, we say that x is *good* if $u_x \sqsubseteq v_x$ and then we let $w_x \stackrel{\text{def}}{=} v_x \oslash u_x$, otherwise it is *bad* and then we let $r_x \stackrel{\text{def}}{=} u_x \ominus v_x$. We say that x is *alive* if $xy \in R$ for some y, otherwise it is *dead*. Finally, we write $|R|$ for the number of states of a FSA for R, and let $L \stackrel{\text{def}}{=} K_v \times |R|$ be a *size threshold* (more details in the proof of Lemma 9.2.5).

A word x is a *cut-off* if, and only if, one of the following conditions holds:

dead cut-off: x is dead;

subsumption cut-off: there exists a strict prefix x' of x such that $x' \sim_R x$, and either

1. both x and x' are good, with $w_{x'} \sqsubseteq w_x$,
2. or both x and x' are bad, with $r_x \sqsubseteq r_{x'}$;

big cut-off: x is alive, bad and $|r_x| > L$.

Let $T \subseteq \Sigma^*$ be the set of all words that do not have a cut-off as a (strict) prefix. T is prefix-closed and can be seen as a tree.

Lemma 9.2.4 *T is finite.*

Proof. We show that T, seen as a tree, has no infinite branch. Hence, and since it is finitely branching, it is finite (Kőnig's Lemma).

Assume, by way of contradiction, that T has an infinite branch labeled by some $x_0, x_1, x_2, \ldots$ (and recall that every x_i is a prefix of all the x_{i+k}'s). We show that one of the x_i must be a cut-off, which contradicts the assumption.

Since the syntactic congruence $\sim_R$ has finite index, there exists an infinite subsequence $x_0, x_1, x_2, \ldots$ (renumbered for convenience) of $\sim_R$-equivalent x_i's. If infinitely many of the x_i's are good, one of them must be a subsumption cut-off since, by Higman's Lemma, the infinite sequence of the w_{x_i}'s (for

good x_i's) must have some $w_{x'} \sqsubseteq w_x$. If only finitely many of the x_i's are good, then infinitely many of them are bad and either some r_{x_i} has size larger than L (hence x_i is a big cut-off), or all r_{x_i}'s have size at most L, hence belong to a finite set $\Gamma^{\leq L}$, and two of them must be equal (hence there must be a subsumption cut-off). □

With the next two lemmas, we show that T contains enough information to decide whether $R \subseteq PE(u, v)$.

Lemma 9.2.5 *If T contains a big cut-off, then $R \not\subseteq PE(u, v)$.*

Proof. Assume x is a big cut-off (i.e., is alive, bad, and with $|r_x| > L$) in T. It is alive so $xy \in R$ for some y. We pick the smallest such y, ensuring that $|y| < |R|$ (the number of states of an FSA for R). Since x is bad, we know that $u_x \not\sqsubseteq v_x$. Note that $|v_y| \leq K_v \times |y| \leq K_v \times |R| \leq L$ so that $|v_y| < |r_x|$ and, consequently, $r_x \not\sqsubseteq v_y$. Thus, and since $r_x = u_x \ominus v_x$, applying Lemma 9.1.6 contrapositively gives $u_x \not\sqsubseteq v_x v_y$ and, *a fortiori*, $u_{xy} \not\sqsubseteq v_{xy}$. Finally $xy \notin PE(u, v)$. Since $xy \in R$, we conclude $R \not\subseteq PE(u, v)$. □

There is a reciprocal.

Lemma 9.2.6 *Assume that T has no big cut-offs and that $(R \cap T) \subseteq PE(u, v)$. Then $R \subseteq PE(u, v)$.*

Proof. Consider some $x \in R$: we show that $u_x \sqsubseteq v_x$ by induction on the size of x. If $x \in T$ then $x \in (R \cap T) \subseteq PE(u, v)$ and we are done. If $x \notin T$, then a prefix of x is a cut-off. This cannot be a big cut-off (we assumed T has none) or a dead cut-off (the prefix is alive since $x \in R$). Hence this is a subsumption cut-off, caused by one of its prefixes. Finally, x can be written under the form $x = x_1x_2x_3$ with x_1x_2 the subsumption cut-off, and x_1 the prefix justifying the subsumption. We know $x_2 \neq \epsilon$ (x_1 is a strict prefix of the cut-off) and $x_1 \sim_R x_1x_2$. Hence $x_1x_3 \in R$ (since $x_1x_2x_3 \in R$) and $u_{x_1x_3} \sqsubseteq v_{x_1x_3}$ by induction hypothesis.

There are now two cases, depending on what kind of subsumption is at hand.

1. If x_1 is good then $u_{x_1} \sqsubseteq v_{x_1}$. Combining with $u_{x_1x_3} \sqsubseteq v_{x_1x_3}$ entails $u_{x_3} \sqsubseteq w_{x_1}v_{x_3}$ (Lemma 9.1.3). From $w_{x_1} \sqsubseteq w_{x_1x_2}$ (condition for subsumption) we deduce $u_{x_3} \sqsubseteq w_{x_1x_2}v_{x_3}$. Combining with $u_{x_1x_2} \sqsubseteq v_{x_1x_2}$ (x_1x_2 too is good), Lemma 9.1.1 yields $u_{x_1x_2}u_{x_3} \sqsubseteq v_{x_1x_2}v_{x_3}$.

2. If x_1 is bad, then $u_{x_1x_3} \sqsubseteq v_{x_1x_3}$ and $u_{x_1} \not\sqsubseteq v_{x_1}$ entail $r_{x_1}u_{x_3} \sqsubseteq v_{x_3}$ (Lemma 9.1.6). From $r_{x_1x_2} \sqsubseteq r_{x_1}$ (condition for subsumption) we deduce $r_{x_1x_2}u_{x_3} \sqsubseteq v_{x_3}$. Combined with $u_{x_1x_2} \not\sqsubseteq v_{x_1x_2}$ (x_1x_2 too is bad), applying Coro. 9.1.5 contrapositively yields $u_{x_1x_2}u_{x_3} \sqsubseteq v_{x_1x_2}v_{x_3}$.

In both cases we proved that $x_1x_2x_3 \in PE(u, v)$ as requested. □

We can now prove the decidability of PEP_Containment: the tree T can be built effectively starting from the root since it is easy to see whether a

word is a cut-off. The construction terminates thanks to Lemma 9.2.4. Once T is at hand, Lemmas 9.2.5 and 9.2.6 gives an effective criterion for deciding whether $R \subseteq PE(u, v)$: it is enough to check that T has no big cut-off and that all the words $x \in T$ satisfy $u_x \sqsubseteq v_x$ or do not belong to R.

9.3 Pumpable solutions and antisolutions

Let $u, v : \Sigma^* \to \Gamma^*$ be a given PEP instance.

Definition 9.3.1 *A triple of words $(x, y, z) \in \Sigma^*$ with $y \neq \epsilon$ is a* pumpable solution *if $xy^kz \in PE(u, v)$ for all $k \in \mathbb{N}$.*
It is a pumpable antisolution *if $xy^kz \notin PE(u, v)$ for all $k \in \mathbb{N}$.*

In other words, a pumpable solution denotes an infinite subset of $PE(u, v)$ of the form xy^*z, while a pumpable antisolution denotes an infinite subset of its complement. Our interest in pumpable solutions and antisolutions is that they provide simple witnesses proving that $PE(u, v)$ (or its complement) is infinite.

We observe that these witnesses are effective:

Proposition 9.3.2 (Decidability of pumpability) *It is decidable whether (x, y, z) is a pumpable solution, and also whether it is a pumpable antisolution.*

Proof. Checking that (x, y, z) is a pumpable solution reduces to the PEP_Containment problem, while checking that it is not a pumpable antisolution reduces to the $\mathsf{PEP}^{\mathrm{reg}}$ problem (or, equivalently, PEP_Inclusion). □ □

We can now state our main technical result. Here (and below) we speak loosely of "a pumpable solution", when we mean "*the language denoted by* a pumpable solution".

Lemma 9.3.3 (Pumping Lemma) *Let $R \subseteq \Sigma^*$ be a regular language.*
1. If $R \cap PE(u, v)$ is infinite, it contains a pumpable solution.
2. If $R \smallsetminus PE(u, v)$ is infinite, it contains a pumpable antisolution.

Section 9.5 is devoted to a proof of the Pumping Lemma for solutions, while Section 9.6 proves the Pumping Lemma for antisolutions. Without waiting for that, we list the main consequences on our questions.

9.4 Quasi-regular properties and counting properties

For two languages L, L, we say that L is *quasi-included* in L', written $L \subseteq_a L'$, when $L \setminus L'$ is finite, and that they are *quasi-equal*, written $L =_a L'$, when $L \subseteq_a L'$ and $L' \subseteq_a L$.

We consider the following questions, where we are given a PEP instance u, v and a regular $R \subseteq \Sigma^*$:
PEP_Quasi_Inclusion: does $PE(u,v) \subseteq_a R$?
PEP_Quasi_Containment: does $PE(u,v) \supseteq_a R$?
PEP_Quasi_Equality: does $PE(u,v) =_a R$?

Theorem 9.4.1 PEP_Quasi_Inclusion, PEP_Quasi_Containment *and* PEP_Quasi_Equality *are decidable.*

Proof. We start with PEP_Quasi_Inclusion. This problem is co-r.e. since when $PE(u,v) \setminus R$ is infinite, there is a pumpable solution in $\Sigma^* \setminus R$ (Pumping Lemma) that can be guessed and checked (Prop. 9.3.2). It is also r.e. since $PE(u,v) \subseteq_a R$ iff there is a finite language $F \subseteq \Sigma^*$ s.t. $PE(u,v) \subseteq R \cup F$, which can be checked (Theo. 9.2.2) since $R \cup F$ is a regular language. Thus PEP_Quasi_Inclusion, being r.e. and co-r.e., is decidable.

We use the same reasoning to show that PEP_Quasi_Containment is decidable. Then PEP_Quasi_Equality is obviously decidable as well. □ □

We also consider counting questions where the answer is a number in $\mathbb{N} \cup \{\omega\}$:
PEP_NbSol: what is the cardinality of $R \cap PE(u,v)$?
PEP_NbAntisol: what is the cardinality $R \setminus PE(u,v)$?

Theorem 9.4.2 PEP_NbSol *and* PEP_NbAntisol *are decidable (more precisely, the associated counting functions are recursive).*

Proof. We start with PEP_NbSol. We can first check whether the cardinality of $R \cap PE(u,v)$ is finite by deciding whether $PE(u,v) \subseteq_a (\Sigma^* \setminus R)$ (using the decidability of PEP_Quasi_Inclusion). If we find that the cardinality is infinite, we are done. Otherwise we can enumerate all words in R and check whether they are solutions. At any given stage during this enumeration, we can check whether the current set F of already found solutions is complete by deciding whether $PE(u,v) \cap (R \setminus F) = \emptyset$ (using the decidability of PEP_Inclusion). We are bound to eventually find a complete set since we only started enumerating solutions in R knowing there are finitely many of them.

The same method works for PEP_NbAntisol, this times using the decidability of PEP_Containment and PEP_Quasi_Containment. □ □

9.5 Pumping in long solutions

We start with a sufficient condition for pumpability of solutions.

Definition 9.5.1 *A triple $x, y, z \in \Sigma^*$ with $y \neq \epsilon$ is* positive *if the following four conditions are satisfied:*

$$u_x \sqsubseteq v_x, \quad \text{(C1)} \qquad u_x u_y \sqsubseteq v_x v_y, \quad \text{(C2)}$$
$$u_x u_y u_z \sqsubseteq v_x v_y v_z, \quad \text{(C3)} \qquad (v_x \oslash u_x) \sqsubseteq (v_x v_y \oslash u_x u_y). \quad \text{(C4)}$$

Lemma 9.5.2 *If (x, y, z) is positive then (x, y, yz) is a pumpable solution.*

Proof. Assume that (x, y, z) is positive, so that (C1–4) hold. Write shortly w for $v_x \oslash u_x$ and w' for $v_{xy} \oslash u_{xy}$. From (C1) and the definition of w, Coro. 9.1.2 yields:

$$u_x w \sqsubseteq v_x. \quad \text{(C5)}$$

From (C2), it further yields $u_x u_y w' \sqsubseteq v_x v_y$, from which (C4) entails:

$$u_x u_y w \sqsubseteq v_x v_y. \quad \text{(C6)}$$

Applying Lemma 9.1.3 on (C1) and (C3) (respectively on (C1) and (C6)) yields:

$$u_y u_z \sqsubseteq w v_y v_z, \quad \text{(C7)} \qquad u_y w \sqsubseteq w v_y. \quad \text{(C7')}$$

Applying Lemma 9.1.7 on (C7') gives

$$u_{y^k} w = (u_y)^k w \sqsubseteq w (v_y)^k = w v_{y^k} \text{ for all } k \in \mathbb{N}. \quad \text{(C8)}$$

With (C5) and (C8), Lemma 9.1.1 entails

$$u_x u_{y^k} w \sqsubseteq v_x v_{y^k} \text{ for all } k \in \mathbb{N}. \quad \text{(C9)}$$

With (C7) and (C9), it then entails

$$u_x u_{y^k} u_{yz} \sqsubseteq v_x v_{y^k} v_{yz} \text{ for all } k \in \mathbb{N}, \quad \text{(C10)}$$

which just states that (x, y, yz) is a pumpable solution. □ □

We now let n_R denote the number of equivalence classes induced by $\sim_R$ (Section 2.1.2). Finally, we let H_u and H_v denote, respectively, $H(n_R + 1, K_u, |\Gamma|)$ and $H(n_R+1, K_v, |\Gamma|)$. Recall that, by definition of the H function (Lemma 2.1.5), any K_u-controlled sequence of at least H_u Γ-words is $(n_R + 1)$-good.

Lemma 9.5.3 *If R contains a solution $\sigma \in PE(u, v)$ of length $|\sigma| \geq 2H_v$ then it contains a pumpable solution.*

(Observe that this will entail, as a corollary, the first half of the Pumping Lemma since, if $R \cap PE(u,v)$ is infinite, it contains solutions σ of arbitrarily large length.)

Proof. Let $\sigma \in PE(u,v)$ be a solution of length L: σ has $L+1$ prefixes $x_0, x_1, \ldots, x_L$. We consider the subsequence $x_{i_1}, x_{i_2}, \ldots x_{i_l}$ of all prefixes of σ that satisfy $u_{x_{i_j}} \sqsubseteq v_{x_{i_j}}$ (called *good prefixes*) and split the proof in three main steps.

1. We show, by induction over j, that the sequence $\left(v_{x_{i_j}} \oslash u_{x_{i_j}}\right)_{j=1,..,l}$ is K_v-controlled, i.e., writing w_j for $v_{x_{i_j}} \oslash u_{x_{i_j}}$, that $|w_j| \leq j \times K_v$ for all $j = 1, \ldots, l$. The base case is obvious since $i_1 = 0$ and $w_1 = \epsilon$. For the inductive case, we consider $j > 0$ so that $x_{i_j} = x_{i_j-1}.a$ for some $a \in \Sigma$ (the i_j-th letter in σ). If $u_{x_{i_j-1}} \sqsubseteq v_{x_{i_j-1}}$ (hence $i_{(j-1)} = (i_j) - 1$) then $w_j = v_{x_{i_j}} \oslash u_{x_{i_j}}$ is $(v_{x_{i_j-1}}.v_a) \oslash (u_{x_{i_j-1}}.u_a)$ which cannot be longer than $(v_{x_{i_j-1}}.v_a) \oslash u_{x_{i_j-1}}$, itself not longer than $(v_{x_{i_j-1}} \oslash u_{x_{i_j-1}}).v_a$. Thus $|w_j| \leq |w_{j-1}| + K_v$ and we conclude with the induction hypothesis. If on the other hand $u_{x_{i_j-1}} \not\sqsubseteq v_{x_{i_j-1}}$, then w_j is a suffix of v_a hence $|w_j| \leq K_v$.

2a. Assume now that $l \geq H_v$. Then, using Lemma 2.1.5, we conclude that there is a further subsequence $(x_{i_{j_r}})_{r=0,\ldots,n_R}$ of n_R+1 prefixes of σ such that $w_{j_0} \sqsubseteq w_{j_1} \sqsubseteq \cdots \sqsubseteq w_{j_{n_R}}$. Since n_R is the index of $\sim_R$, we deduce that there exists two such prefixes $x_{i_{j_p}}$ (shortly, x) and $x_{i_{j_{p'}}}$ (shortly, x') with $x \sim_R x'$. If we write x' under the form xy (NB: $y \neq \epsilon$) and σ under the form xyz, we have found a positive triple (x,y,z). Then Lemma 9.5.2 applies and shows that xy^*yz is a pumpable solution. Finally, since $x \sim_R xy$, we know that xy^*yz is a subset of R.

2b. Observe that if a prefix x_i of $\sigma = x_i.y_i$ is not good, then $\widetilde{y_i}$ is a good prefix of the solution $\widetilde{\sigma} \in PE(\widetilde{u}, \widetilde{v})$ of the mirror PEP problem. Hence if σ has $l < H_v$ good prefixes, $\widetilde{\sigma}$ has $l' \geq 2H_v - l > H_v$ good ones. Then the mirror problem falls in case 2a above (we note that $\sim_R$, n_R, and K_v do not have to be adjusted when mirroring). We deduce that there is a pumpable solution in $\widetilde{R} \cap PE(\widetilde{u}, \widetilde{v})$, whose mirror is a pumpable solution in $R \cap PE(u,v)$. □

9.6 Pumping in long antisolutions

As with pumpable solutions, there is a sufficient condition for pumpability of antisolutions.

Definition 9.6.1 *A triple $x, y, z \in \Sigma^*$ with $y \neq \epsilon$ is* negative *if the following four conditions are satisfied:*

$$u_x \not\sqsubseteq v_x, \quad \text{(D1)} \qquad u_x u_y \not\sqsubseteq v_x v_y, \quad \text{(D2)}$$

$$u_x u_z \not\sqsubseteq v_x v_z \quad \text{(D3)} \qquad u_x \ominus v_x \sqsubseteq u_{xy} \ominus v_{xy} \quad \text{(D4)}$$

Lemma 9.6.2 *If (x, y, z) is negative then (x, y, z) is a pumpable antisolution.*

Proof. Assume that (x, y, z) is negative, so that (D1–4) hold. Write shortly r for $u_x \ominus v_x$ and r' for $u_{xy} \ominus v_{xy}$. With (D1), (D2) and (D4), Lemma 9.1.8 applies and yields

$$u_{xy^k} \not\sqsubseteq v_{xy^k} \text{ for all } k \in \mathbb{N}, \tag{D5}$$

with furthermore

$$u_{xy^k} \ominus v_{xy^k} \sqsubseteq u_{xy^{k+1}} \ominus v_{xy^{k+1}}. \tag{D6}$$

On the other hand, (D1) and (D3) entail $ru_z \not\sqsubseteq v_z$ by Coro. 9.1.5, hence $(u_{xy^k} \ominus v_{xy^k})u_z \not\sqsubseteq v_z$ by (D6). We deduce that $u_{xy^k z} \not\sqsubseteq v_{xy^k z}$. □ □

Lemma 9.6.3 *If R contains an antisolution $\sigma \notin PE(u, v)$ of length $|\sigma| \geq 2H_u$ then it contains a pumpable antisolution.*

(As a corollary, we obtain the second half of the Pumping Lemma.)
Proof.[Sketch] We proceed as with Lemma 9.5.3. Write L for $|\sigma|$, and $x_0, x_1, \ldots, x_L$ for the prefixes of σ. Consider the subsequence $x_{i_1}, x_{i_2}, \ldots x_{i_l}$ of all *bad prefixes* of σ, i.e., such that $u_{x_{i_j}} \not\sqsubseteq v_{x_{i_j}}$ and define $r_j = u_{x_{i_j}} \ominus v_{x_{i_j}}$. The sequence $(r_j)_{j=1,\ldots,l}$ is K_u-controlled.

If $l \geq H_u$, we find two positions $1 \leq p < p' \leq l$ such that $x_{i_{j_p}} \sim_R x_{i_{j_{p'}}}$ and $r_{j_p} \sqsubseteq r_{j_{p'}}$, so that, writing x for $x_{i_{j_p}}$, x' for $x_{i_{j_{p'}}}$, writing x' under the form xy, and σ under the form xyz, we can apply Lemma 9.6.2 and deduce that (x, y, z) is a pumpable antisolution. Furthermore xy^*z is a subset of R since $xyz = \sigma \in R$ and $xy \sim_R x$.

Observe that if a prefix x_i is not bad, then, writing σ under the form $x_i y_i$, $\widetilde{y_i}$ is a bad prefix of the antisolution $\widetilde{\sigma} \notin PE(\widetilde{u}, \widetilde{v})$ of the mirror problem. Thus, if $l < H_u$, then $\widetilde{\sigma}$ has $\geq H_u$ bad prefixes in the mirror problem. Hence $\widetilde{R} \smallsetminus PE(\widetilde{u}.\widetilde{v})$ contains a pumpable antisolution, whose mirror is a pumpable antisolution in $R \cap PE(u, v)$. □ □

Remark 9.6.4 *Lemmas 9.5.3 and 9.6.3 show that one can strengthen the statement of the Pumping Lemma. Rather than assuming that $R \cap PE(u, v)$ (respectively, $R \smallsetminus PE(u, v)$) is infinite, we only need to assume that they contain a large enough element.* □

9.7 Concluding remarks

The decidability of the Regular Post Embedding Problem means that one can find out whether the inequation $u(x) \sqsubseteq v(x)$ has a solution in a given

regular R. In this chapter, we investigated more general questions pertaining to the set of solutions $PE(u, v)$. We developed new techniques showing how one can decide regular questions (does $PE(u, v)$ contain, or is it included in, a given R?), finiteness and quasi-regular questions (does $PE(u, v)$ satisfy a regular constraint except perhaps for finitely many elements?), and counting questions (how many elements in some R are — or are not — solutions?).

It is not clear how to go beyond these positive results. One direction is suggested by the pumping lemmas we developed here. These lemmas have applications beyond the finiteness problems we considered. For example, they are useful in the study of the expressive power of $\mathsf{PEP}^{\mathrm{reg}}$-languages, i.e., languages of the form $R \cap PE(u, v)$ for some R, u, v. For example, using the pumping lemma we can show that $L_0 \stackrel{\text{def}}{=} \{a^n b^n \mid n \in \mathbb{N}\}$ is not a $\mathsf{PEP}^{\mathrm{reg}}$-language. Now, and since $L_1 \stackrel{\text{def}}{=} \{a^n b^{n+m} \mid n, m \in \mathbb{N}\}$ and $L_2 \stackrel{\text{def}}{=} \{a^{n+m} b^n \mid n, m \in \mathbb{N}\}$ clearly are $\mathsf{PEP}^{\mathrm{reg}}$-languages, we conclude that $\mathsf{PEP}^{\mathrm{reg}}$-languages are not closed under intersection!

Chapter 10

Conclusion

We tackled the study of the complexity class of lossy channel system problems ($\mathsf{F}_{\omega^\omega}$) and obtained two main results.
— **1**: we give a precise characterization of the reachability problem on LCS's as bounded Turing machine. The main obstacle to solve that problem opened for more that 10 years was to find the right bound from proof theory folklore. We wish that our presentation make the bridge between the two domains natural to the verification community.
— **2**: We defined the Regular Post Embedding Problem ($\mathsf{PEP}^{\mathrm{reg}}$), an abstract problem complete for $\mathsf{F}_{\omega^\omega}$. It's simple definition only rely on basic notion of language theory and summarize the essential properties that make a problem complete for $\mathsf{F}_{\omega^\omega}$. It is suitable for easy definition of variants as shown by our exploration of the more natural ones.
With these result, we think that we obtained a solid basis for the study of that complexity class.

further

- As we now start to understand $\mathsf{PEP}^{\mathrm{reg}}$, its link and difference with $\mathsf{ReachLcs}$, we now have new possibilities. We looked at some natural extensions and variants of $\mathsf{PEP}^{\mathrm{reg}}$, but many other versions are possible.
  - For instance $\mathsf{PEP}^{\mathrm{reg}}$ is the question, whether $\exists x \in R : u(x) \sqsubseteq v(x)$. There could be an interesting decidable logic summarizing and extending all our results on languages of solutions.
  - Are there languages classes that could be used as constraint to have simpler problems ?

  LCS's have been used as a base problem to show that problems are not primitive recursive. We think that $\mathsf{PEP}^{\mathrm{reg}}$ or $\mathsf{PEP}^{\mathrm{reg}}_{\mathrm{dir}}$ are better for that role. Such reduction should be written to emphasis it. Moreover, since there was no upper bound on $\mathsf{ReachLcs}$, only lower bound were shown using those reductions. Some of the problems harder that $\mathsf{ReachLcs}$ could be shown equivalent. That would need new reductions where $\mathsf{PEP}^{\mathrm{reg}}$ could prove useful.

- Cichon and Tahhan Bittar's proof on Higman's lemma could be made more precise. Indeed, the function bounding the length of bad sequence is $F_{\omega^\omega} \circ p$ for some p left implicit but primitive recursive, which is very loose. We can suppose, from some results of de Jongh and Parikh [dJP77], showing the order type of $\sqsubseteq$, that the function p could be the identity. There is an undergoing similar work on the Dickson's lemma, which from simplifying the proof went to make it more precise. It would be also interesting to have such a proof that don't need background in proof theory to understand.

- Our work on LCS can be seen as a link between the word data structure ordered with subword and the ordinal ω^ω. Such a correspondence could be made on other data type, like multiset, or extend the results to be able to compose data-types. For instance this would allow to directly characterize systems working on words of tuples of integers.

Bibliography

[AB09] Mohamed Faouzi Atig and Ahmed Bouajjani. On the reachability problem for dynamic networks of concurrent pushdown systems. In *RP*, pages 1–2, 2009.

[ABBM10] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18, 2010.

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV '98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998.

[ABRS05] 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.

[ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. Trex: A tool for reachability analysis of complex systems. In *CAV*, pages 368–372, 2001.

[ACBJ04] 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.

[ADO+08] P. A. Abdulla, J. Deneux, J. Ouaknine, K. Quaas, and J. Worrell. Universality analysis for one-clock timed automata. *Fundamenta Informaticae*, 89(4):419–450, 2008.

[AJ93] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. 8th IEEE Symp. Logic in Computer Science (LICS '93), Montreal, Canada, June 1993*, pages 160–170. IEEE Comp. Soc. Press, 1993.

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

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

[AK95] Parosh Aziz Abdulla and Mats Kindahl. Decidability of simulation and bisimulation between lossy channel systems and finite state systems (extended abstract). In *CONCUR*, pages 333–347, 1995.

[BBS06] C. Baier, N. Bertrand, and Ph. Schnoebelen. On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In *Proc. LPAR 2006*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 347–361. Springer, 2006.

[BBS07] C. Baier, N. Bertrand, and Ph. Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Transactions on Computational Logic*, 9(1), 2007.

[BS03] N. Bertrand and Ph. Schnoebelen. Model checking lossy channels systems is probably decidable. In *Proc. 6th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS 2003), Warsaw, Poland, Apr. 2003*, volume 2620 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2003.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CFP96] 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.

[Cic07] E. A. Cichon. Personal communication, December 2007.

[Clo86] P. Clote. On the finite containment problem for Petri nets. *Theoretical Computer Science*, 43(1):99–105, 1986.

[CS10] P. Chambart and Ph. Schnoebelen. Toward a compositional theory of leftist grammars and transformations. In *Proc. FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2010.

[CT98] E. A. Cichon and E. Tahhan Bittar. Ordinal recursive bounds for Higman's theorem. *Theoretical Computer Science*, 201(1–2):63–84, 1998.

[dJP77] D. H. J. de Jongh and R. Parikh. Well-partial orderings and hierarchies. *Indag. Math.*, 39(3):195–207, 1977.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[FFSS10] Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermann and primitive-recursive bounds with Dickson's lemma. Research Report cs.LO/1007.2989, Computing Research Repository, July 2010.

[FG09a] Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In Susanne Albers and Jean-Yves Marion, editors, *Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS'09)*, volume 3 of *Leibniz International Proceedings in Informatics*, pages 433–444, Freiburg, Germany, February 2009. Leibniz-Zentrum für Informatik.

[FG09b] Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: Complete WSTS. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, and Wolfgang Thomas, editors, *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lecture Notes in Computer Science*, pages 188–199, Rhodes, Greece, July 2009. Springer.

[Fin85] A. Finkel. Une généralisation des théorèmes de Higman et de Simon aux mots infinis. *Theoretical Computer Science*, 38(1):137–142, 1985.

[Fin87] Alain Finkel. A generalization of the procedure of Karp and Miller to well structured transition system. In Thomas Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508, Karlsruhe, Germany, July 1987. Springer-Verlag.

[Fin94] A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7(3):129–135, 1994.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[FW98] M. V. Fairtlough and S. S. Wainer. Hierarchies of provably recursive functions. In S. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic*, chapter 3, pages 149–207. North-Holland, 1998.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford Univ. Press, 1995.

[GKWZ06] D. Gabelaia, A. Kurucz, F. Wolter, and M. Zakharyaschev. Nonprimitive recursive decidability of products of modal logics with expanding domains. *Annals of Pure and Applied Logic*, 142(1–3):245–268, 2006.

[Hai69] L. H. Haines. On free monoids partially ordered by embedding. *J. Combinatorial Theory*, 76:94–98, 1969.

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

[HMK+05] J. G. Henriksen, M. Mukund, K. N. Kumar, M. A. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.

[Jan01] Petr Jancar. Nonprimitive recursive complexity and undecidability for petri net equivalences. *Theor. Comput. Sci.*, 256(1-2):23–30, 2001.

[Jur08] T. Jurdziński. Leftist grammars are nonprimitive recursive. In *Proc. ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2008.

[Kre52] G. Kreisel. On the interpretation of the nonfinitist proofs, ii. *The Journal of Symbolic Logic*, 17:43–58, 1952.

[Kru72] Joseph B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Comb. Theory, Ser. A*, 13(3):297–305, 1972.

[Lee78] J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.

[LM04] M. Lohrey and A. Muscholl. Bounded MSC communication. *Information and Computation*, 189(2):160–181, 2004.

[LNO+08] R. Lazić, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundamenta Informaticae*, 88(3):251–274, 2008.

[Lot83] M. Lothaire, editor. *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Cambridge Univ. Press, 1983.

[Lot02] M. Lothaire, editor. *Algebraic combinatorics on words*, volume 90 of *Encyclopedia of Mathematics and Its Applications*. Cambridge Univ. Press, 2002.

[LW08] S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Computational Logic*, 9(2), 2008. To appear.

[May03a] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, 2003.

[May03b] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, 2003.

[McA84] K. McAloon. Petri nets and large finite sets. *Theoretical Computer Science*, 32(1–2):173–183, 1984.

[MM81] Ernst W. Mayr and Albert R. Meyer. The complexity of the finite containment problem for petri nets. *J. ACM*, 28(3):561–576, 1981.

[Mül85] H. Müller. Weak Petri net computers for Ackermann functions. *Elektronische Informationsverarbeitung und Kybernetik*, 21(4/5):236–246, 1985.

[Odi92] P. Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers (Studies in Logic and the Foundations of Mathematics)*. North Holland, new ed edition, February 1992.

[OW07] J. Ouaknine and J. Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Comp. Science*, 3(1):1–27, 2007.

[Pś5] R. Péter. Konstruktion nichtrekursiver funktionen. *Math. Ann.*, 111:42–60, 1935.

[Rob65] J.W. Robbin. *Subrecursive hierarchies*. Ph.d. thesis, Princeton University, 1965.

[Ros84] H. E. Rose. *Subrecursion: Functions and Hierarchies*, volume 9 of *Oxford Logic Guides*. Oxford Univ. Press, 1984.

[Sch02] Ph. Schnoebelen. Verifying lossy channel systems has non-primitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[Sch10] Ph. Schnoebelen. Lossy counter machines: A survey. In *Proc. RP 2010*, Lecture Notes in Computer Science. Springer, 2010.

[Tou97] H. Touzet. *Propriétés combinatoires pour la terminaison de systèmes des réécriture*. Thèse de doctorat, Université de Nancy 1, France, September 1997.

[Tou02] H. Touzet. A characterisation of multiply recursive functions with higman's lemma. *Information and Computation*, 178:534–544, 2002.

[Wai70] S. S. Wainer. A classification of the ordinal recursive functions. *Arch. math. Logik Grundlag.*, 13(3–4):136–153, 1970.

[Wai72] S. S. Wainer. Ordinal recursion, and a refinement of the extended grzegorczyk hierarchy. *The Journal of Symbolic Logic*, 37(2):281–292, 1972.

Appendix A

Combinatorics on subwords

Here are gathered all proofs of subword combinatorics results we use in the document.

A.1 Basics

It will be convenient to recall the following obvious facts:

Fact A.1.1 (Splitting)
1. If $xy \sqsubseteq z$ then there exists a factorization $z = z'z''$ of z such that $x \sqsubseteq z'$ and $y \sqsubseteq z''$.
2. If $x \sqsubseteq yz$ then there exists a factorization $x = x'x''$ of x such that $x' \sqsubseteq y$ and $x'' \sqsubseteq z$.

Proof. To prove this simple result, we need to use the definition of $\sqsubseteq$ given in section 6.1.1.
1. If $xy \sqsubseteq z$ then there exists an order-preserving injective map $h : \{1, \ldots, |xy|\} \to \{1, \ldots, |z|\}$ such that $xy_i = z_{h(i)}$ for all $i = 1, \ldots, |xy|$. If x or y is empty, then that holds trivially, so suppose it is not the case. Then $h' = h$ is an order-preserving injective map on $\{1, \ldots, |x|\} \to \{1, \ldots, h(|x|)\}$ and $h'' = \lambda a.(h(|x| + a))$ is one on $\{|x| + 1, \ldots, |xy|\} \to \{h(|x|) + 1, \ldots, |z|\}$ such that $x_i = z_{h'(i)}$ and $y_i = z_{h''(i)}$. To conclude, we just need to take z' as the prefix of length $h(|x|)$ of z and z'' as the remaining suffix.
2. If $x \sqsubseteq yz$ then there exists an order-preserving injective map $h : \{1, \ldots, |x|\} \to \{1, \ldots, |yz|\}$ such that $x_i = yz_{h(i)}$ for all $i = 1, \ldots, |x|$. If $h(|x|) \leq |y|$ or $h(1) > |y|$ then that holds trivially, so suppose it is not the case. Then there exists n such that $h(n) \leq |y|$ and $h(n + 1) > |y|$. Then $h' = h$ is an order-preserving injective map on $\{1, \ldots, n\} \to \{1, \ldots, h(|y|)\}$ and $h'' = \lambda a.(h(a) - h(n))$ is one on $\{n + 1, \ldots, |x|\} \to \{|y| + 1, \ldots, |yz|\}$ such that $x_i = y_{h'(i)}$ and $x_i = z_{h''(i)}$ when defined. To conclude, we just need to take x' as the prefix of length n of x and x'' as the remaining suffix.
□

Now, we won't have to use that kind of order-preserving injective map, the other lemmas only rely on Fact A.1.1.

Lemma A.1.2 (Elimination Lemma) *If $xw \sqsubseteq y$ and $x' \sqsubseteq wy'$ then $xx' \sqsubseteq yy'$.*

Proof. By Fact A.1.1 there exist factorizations $y = y_1.y_2$ and $x' = x'_1.x'_2$ such that $x \sqsubseteq y_1$, $w \sqsubseteq y_2$, $x'_1 \sqsubseteq w$ and $x'_2 \sqsubseteq y'$. One concludes with $xx' = xx'_1x'_2 \sqsubseteq y_1wy' \sqsubseteq y_1y_2y' = yy'$. □

Lemma A.1.3 (Mirror Elimination Lemma) *If $x \sqsubseteq yw$ and $wx' \sqsubseteq y'$ then $xx' \sqsubseteq yy'$.*

Proof. Mirroring the assumptions gives $\widetilde{x} \sqsubseteq \widetilde{w}\widetilde{y}$ and $\widetilde{x'}\widetilde{w} \sqsubseteq \widetilde{y'}$. Then Lemma A.1.2 applies, yielding $\widetilde{x'}\widetilde{x} \sqsubseteq \widetilde{y'}\widetilde{y}$. Mirroring again gives $xx' \sqsubseteq yy'$. □

A.2 Available suffixes

Recall that, when $x \sqsubseteq y$, the "used prefix" is the shortest prefix y_1 of y such that $x \sqsubseteq y_1$. Then, writing $y = y_1y_2$, what remains, i.e., y_2, is called the "available suffix" and denoted $y \oslash x$.

Fact A.2.1 (Monotonicity)
1. If $x \sqsubseteq y$, then $(yz) \oslash x = (y \oslash x)z$.
2. If $xx' \sqsubseteq y$, then $y \oslash (xx') = (y \oslash x) \oslash x'$.

Lemma A.2.2 *$xz \sqsubseteq y$ implies $z \sqsubseteq y \oslash x$.*

Proof. If $xz \sqsubseteq y$ then $x \sqsubseteq y'$ and $z \sqsubseteq y''$ for a factorization $y'y''$ of y (Fact A.1.1). Since y_1 is the shortest prefix with $x \sqsubseteq y_1$, it is a prefix of y', hence y'' is a suffix of y_2. Hence $z \sqsubseteq y_2 = y \oslash x$. □

Lemma A.2.3 *$x \sqsubseteq y$ and $x' \sqsubseteq (y \oslash x)y'$ imply $xx' \sqsubseteq yy'$.*

Proof. Let $x' = x'_1x'_2$ be a factorization with $x'_1 \sqsubseteq y \oslash x$ and $x'_2 \sqsubseteq y'$. Lemma A.2.2 gives $xx'_1 \sqsubseteq y$. Concatenating with $x'_2 \sqsubseteq y'$ concludes since $xx' = (xx'_1)x'_2$. □

Corollary A.2.4 *$x \sqsubseteq y$ implies $x(y \oslash x) \sqsubseteq y$.*

Lemma A.2.5 *$x \sqsubseteq y$ and $xx' \sqsubseteq yy'$ imply $x' \sqsubseteq (y \oslash x)y'$.*

Proof. From $xx' \sqsubseteq yy'$, Lemma A.2.2 gives $x' \sqsubseteq yy' \oslash x$. But $yy' \oslash x = (y \oslash x)y'$ since $x \sqsubseteq y$ (Fact A.2.1). □

A.3 Unmatched suffixes

Recall that, when $x \not\sqsubseteq y$, the "matched prefix" is the longest prefix x_1 of x s.t. $x_1 \sqsubseteq y$. Then, writing $x = x_1x_2$, what remains, i.e., x_2, is called the "unmatched suffix" and denoted $x \ominus y$.

The following is immediate from the definition:

Fact A.3.1 *If $x \not\sqsubseteq yz$ then $x \ominus (yz) = (x \ominus y) \ominus z$.*

Lemma A.3.2 *Assume $x \not\sqsubseteq y$. Then $x \ominus y \sqsubseteq z$ implies $x \sqsubseteq yz$.*

Proof. In other words, assume $x_1 \sqsubseteq y$ and $x_2 \sqsubseteq z$ and conclude $x = x_1x_2 \sqsubseteq yz$. □

Reciprocally:

Lemma A.3.3 *Assume $x \not\sqsubseteq y$. Then $x \sqsubseteq yz$ implies $x \ominus y \sqsubseteq z$.*

Proof. If $x \sqsubseteq yz$ then $x' \sqsubseteq y$ and $x'' \sqsubseteq z$ for a factorization $x'x''$ of x (Fact A.1.1). However, if $x \not\sqsubseteq y$, then $x = x_1x_2$ where $x_2 = x \ominus y$ and x_1 is is the longest prefix of x with $x_1 \sqsubseteq z$, ensuring that x' is a prefix of x_1, hence that x_2 is a suffix of x''. Finally, $x \ominus y = x_2 \sqsubseteq x'' \sqsubseteq z$. □

Lemma A.3.4 *$x \not\sqsubseteq y$ implies $(xx') \ominus y = (x \ominus y)x'$.*

Proof. Since $x \not\sqsubseteq y$, the prefixes of x that embed in y are exactly the prefixes of xx' that embed in y, hence their longest matched prefixes coincide. The unmatched suffixes are x_2 for $x \ominus y$ and x_2x' for $(xx') \ominus y$. □

Lemma A.3.5 *$x \not\sqsubseteq y$ and $xx' \not\sqsubseteq yy'$ imply $[(x \ominus y)x'] \ominus y' = xx' \ominus yy'$.*

Proof. By applying Lemma A.3.4: $(x \ominus y)x' = (xx') \ominus y$ and Fact A.3.1: $[(xx') \ominus y] \ominus y' = (xx') \ominus (yy')$. □

Corollary A.3.6 *$x \not\sqsubseteq y$ and $xx' \not\sqsubseteq yy'$ imply $(x \ominus y)x' \not\sqsubseteq y'$.*

Lemma A.3.7 *$x \not\sqsubseteq y$ and $xx' \sqsubseteq yy'$ imply $(x \ominus y)x' \sqsubseteq y'$.*

Proof. If $x \not\sqsubseteq y$ then $x = x_1x_2$ with $x_1 \sqsubseteq y$ the matched prefix and $x_2 = x \ominus y$. If $xx' \sqsubseteq yy'$ then there is a factorisation $xx' = zz'$ with $z \sqsubseteq y$ and $z' \sqsubseteq y'$ (Fact A.1.1). Hence z is a prefix of x_1 (Lemma A.3.4) so that x_2x' is a suffix of z'. We conclude since $x_2x' = (x \ominus y)x'$. □

A.4 Decomposition

Lemma A.4.1

$$u.w \sqsubseteq v.t \text{ if and only if } \begin{cases} u \sqsubseteq v \text{ and } w \sqsubseteq v \oslash u.t \\ \text{or} \quad u \not\sqsubseteq v \text{ and } u \ominus v.w \sqsubseteq t. \end{cases}$$

Proof. Assume $xx' \sqsubseteq yy'$. Then there exists a factorization $yy' = zz'$ of yy' such that $x \sqsubseteq z$ and $x' \sqsubseteq z'$ (Fact A.1.1). Since $yy' = zz'$, either z is a prefix of y or z' is a suffix of y'. In the first case, we let the interpolant w be given by writing $y = zw$, so that $z' = wy'$. Now, from $x \sqsubseteq z$ and $x' \sqsubseteq z'$, we deduce the required $xw \sqsubseteq y$ and $x' \sqsubseteq wy'$. In the second case, a mirror reasoning gives $x \sqsubseteq yw$ and $wx' \sqsubseteq y'$ for w obtained by writing $y' = wz'$. □

A.5 Iterating factors

Lemma A.5.1 *For all words x, y, z:*

$$xy \sqsubseteq yz \text{ if, and only if, } x^k y \sqsubseteq yz^k \text{ for all } k \in \mathbb{N}.$$

Proof. We only need to prove the "⇒" direction. This is done by induction on the length of y. The cases where $y = \epsilon$ or $x = \epsilon$ or $k = 0$ are obvious, so we assume that $|y|$, $|x|$ and k are strictly positive. There are now two cases:
1. If $x \sqsubseteq y$, we consider a factorization $y = y_1 y_2$ (e.g., $y_2 = y \oslash x$ is convenient) with $x \sqsubseteq y_1$ (hence $x^k \sqsubseteq y_1^k$) and $y \sqsubseteq y_2 z$. Since $|y_2| < |y|$ (because $x \neq \epsilon$ and hence $y_1 \neq \epsilon$), the induction hypothesis applies and from $y_1 y_2 = y \sqsubseteq y_2 z$ one gets $y_1^k y_2 \sqsubseteq y_2 z^k$. Now $x^k y \sqsubseteq y_1^k y = y_1 y_1^k y_2 \sqsubseteq y_1 y_2 z^k = y z^k$.
2. If $x \not\sqsubseteq y$, we write $x = x_1 x_2$ with $x_2 = x \ominus y$. Thus $x_1 \sqsubseteq y$ and $x_2 y \sqsubseteq z$. Thus there exists a factorization $z = z_1 z_2$ s.t. $x_2 \sqsubseteq z_1$ (entailing $x \sqsubseteq y z_1$) and $y \sqsubseteq z_2$. Now $x^k y \sqsubseteq (y z_1)^k z_2 = y z_1 (y z_1)^{k-1} z_2 \sqsubseteq y z_1 (z_2 z_1)^{k-1} z_2 = y z^k$.
□

Lemma A.5.2 *Assume $x \not\sqsubseteq y$, $xz \not\sqsubseteq yt$, and $x \ominus y \sqsubseteq xz \ominus yt$. Then for all $k \in \mathbb{N}$:*

$$xz^k \not\sqsubseteq yt^k. \tag{Z_k}$$

Furthermore, if we let $r_k \stackrel{def}{=} xz^k \ominus yt^k$, then for all $k \in \mathbb{N}$:

$$r_0 \sqsubseteq r_k \sqsubseteq r_{k+1}. \tag{R_k}$$

Proof. The hypothesis for the Lemma are that (Z_0), (Z_1) and (R_0) hold. We prove, by induction on k, that (Z_k) and (R_{k-1}) imply (Z_{k+1}) and (R_k).

Proof of (Z_{k+1}): applying Coro. A.3.6 on (Z_0) and (Z_1) yields $r_0 z \not\sqsubseteq t$, hence a fortiori $r_k z \not\sqsubseteq t$ using (R_{k-1}). Combining with (Z_k) and applying Lemma A.3.7 contrapositively entails $xz^k z \not\sqsubseteq yt^k t$, i.e., ($\mathrm{Z}_{k+1}$).

Proof of (R_k): r_{k+1} is $xz^{k+1} \ominus yt^{k+1}$. By Lemma A.3.5, this is $[(xz^k \ominus yt^k)z] \ominus t$, i.e., $r_k z \ominus t$. From (R_{k-1}) we get $r_{k-1} z \ominus t \sqsubseteq r_k z \ominus t$. However $r_{k-1} z \ominus t = r_k$ (Lemma A.3.5). Finally $r_k \sqsubseteq r_{k+1}$. □