Subwords : automata, embedding problems, and verification

Prateek Karandikar

Subwords: automata, embedding problems, and verification

Prateek Karandikar

Chennai Mathematical Institute, Chennai

and

Laboratoire Spécification et Vérification,
École Normale Supérieure, Cachan

supervised by

Narayan Kumar
CMI, Chennai

Philippe Schnoebelen
LSV, CNRS, & ENS Cachan

Doctoral thesis in computer science

Chennai Mathematical Institute
École Doctorale Sciences Pratiques, ENS Cachan

12 February 2015

Members of the jury:
Narayan Kumar (advisor)
Ranko Lazic (examiner)
Madhavan Mukund
Jean-François Raskin
Philippe Schnoebelen (advisor)
Marc Zeitoun (examiner)

Abstract

The increasing use of software and automated systems has made it important to ensure their correct behaviour. Bugs can not only have significant financial costs, but also catastrophic consequences in mission-critical systems. Testing against a variety of environments and inputs helps with discovering bugs, but cannot guarantee their absence. Formal verification is the technique that establishes correctness of a system or a mathematical model of the system with respect to properties expressed in a formal language.

Regular model checking is a common technique for verification of infinite-state systems - it represents infinite sets of configurations symbolically in a finite manner and manipulates them using these representations. Regular model checking for lossy channel systems (LCS) brings up basic automata-theoretic questions concerning the subword relation on words which models the lossiness of the channels. We address these state complexity and decision problems, and also solve a long-standing problem involving the index of the Simon's piecewise-testability congruence.

The reachability problem for lossy channel systems, though decidable, has very high complexity (it is $\mathsf{F}_{\omega^\omega}$-complete), well beyond primitive-recursive. In recent times several problems with this complexity have been discovered, for example in the fields of verification of weak memory models and metric temporal logic. The Post Embedding Problem (PEP) is an algebraic abstraction of the reachability problem on LCS, with the same complexity, and is our champion for a "master" problem for the class $\mathsf{F}_{\omega^\omega}$. We provide a common generalization of two known variants of PEP and give a simpler proof of decidability. This allows us to extend the unidirectional channel system (UCS) model with simple channel tests while having decidable reachability.

Contents

Chapter 1

Introduction

The increasing use of software and automated systems has made it important to ensure their correct behaviour. Bugs can not only have significant financial costs, but also catastrophic consequences in mission-critical systems. Testing against a variety of environments and inputs helps with discovering bugs, but cannot guarantee their absence. Concurrency, for example, can introduce very subtle and hard-to-discover bugs, and with large complex systems it is impossible to evaluate by hand whether testing was comprehensive. Formal verification is the technique that establishes correctness of a system or a mathematical model of the system with respect to properties expressed in a formal language. See [64] for an economic perspective on bugs, testing, and verification and [82] for a survey of formal verification as applied to hardware design.

Early work on sequential programs worked directly with the program as the system with properties expressed in very powerful logics, often extensions of first-order logic or even higher-order logic, tailored to express properties of programs. Establishing the correctness was through proofs in a proof system (for example, Floyd-Hoare logic [61, 49]). One of the drawbacks of this approach is that it is not automatic and requires advanced skills in mathematical reasoning making it difficult to use in practice. Further it does not extend easily to handle concurrent programs or reactive programs. The latter are nonterminating programs that typically provide some service such as device drivers or daemons in an operating system as well as programs that run on embedded controllers.

In a landmark paper [101], Pnueli proposed the use of temporal logic for specifying and verifying properties of programs, including concurrent, reactive programs. Many properties useful in practice, such as safety properties (of the form "⟨*something undesirable*⟩ will never happen") and liveness properties (of the form"⟨*something desirable*⟩ will eventually happen"), are expressed naturally in temporal logic. The use of temporal logic to express properties has been one of the drivers of formal verification over the last few decades and this is because, for many interesting classes of programs, these properties can be checked automatically.

1.1 Model checking

Model checking [34, 41] is the problem of deciding, given a model of a system, whether it satisfies a given specification. The nature of the system and specification can vary and depends on the application in question. Model checking is often used for verifying correctness properties of finite-state systems, the properties typically being expressed in a suitable variant of temporal logic. Tools such as SPIN[1] [65] and SMV[2] have been successful in practice for temporal logic verification.

Models of hardware systems tend to be finite-state, which make them amenable to model-checking-based approaches to verification. Software systems, on the other hand, tend to be infinite-state, thus harder to verify. Even a sequential recursive program with a finite data domain corresponds to an infinite-state transition system because of the unbounded stack size. Infinite sets of states can be handled symbolically, by formulae, or by constraints, or in particular by automata as in regular model checking. Even the finitary aspects of the state space can result in a very large number of states. To delay the effects of this state explosion in verification, it is usually necessary to work on abstractions of the model. Abstract interpretation is a standard approach for developing abstractions of complex programs [38]. As a simple example, one may abstract out an integer variable by remembering only its sign (positive, zero, or negative). This may result in a loss of precision: the sum of a positive and a negative number could be either positive or negative. In practice the abstraction used depends on both the nature of the programs and properties in question. One approach is to start with a very coarse abstraction and iteratively refine it as needed – this is known as counterexample-guided abstraction refinement (CEGAR) and is used by Microsoft's tool SLAM[3].

Regular model checking [83, 22, 120, 9] works with infinite subsets of the state space by representing them finitely and manipulating them via their finite descriptions. The finite descriptions are "regular", defined via automata, regular expressions, semilinear sets, Presburger formulae, or similar constructs with strong effectiveness and closure properties. Manna and Pnueli verify parameterized programs in [91] by replacing finiteness with finite representability. In [21], regular model checking is applied to pushdown systems, using alternating finite automata to represent regular sets of configurations. It gives an algorithm to compute, given a representation of a set of configurations, a representation of the set of all predecessors of this set. This is applied to model-check pushdown systems against temporal logic specifications.

Another example of regular model checking is the reachability analysis of lossy channel systems (LCS, see section 1.2), where upward-closed sets of words (a subclass of regular languages) are used to represent sets of configurations and compute the predecessor sets. Often such procedures to compute predecessor sets work by iterating the procedure to compute

[1] `http://spinroot.com/spin/whatispin.html`
[2] `http://www.cs.cmu.edu/~modelcheck/smv.html`
[3] `http://research.microsoft.com/en-us/projects/slam/`

one-step predecessors until the complete set of predecessors is found, that is, till we reach a fixed-point of the one-step predecessor operation. Thus termination of fixed-point computations is a central concern with regular model checking. Well-structured transition systems (see section 1.3) are one setting where termination is guaranteed. Where guaranteed termination is not available, one may be content with over- or under- approximations. Another concern is basic data structures for representing regular sets, algorithms for implementing basic operations on these sets, and for deciding properties such as universality, inclusion, and so on. These algorithms and data structures would form basic building blocks of regular model checking algorithms.

1.2 Channel systems

Many systems with otherwise finite-state components have infinitely many configurations as a result of unbounded communication channels. These channels are queues of messages which can grow to arbitrary size. We call such systems *channel systems*. These have been used for protocol modelling [117], analysis [104, 25, 88, 97, 107, 44], and synthesis [122, 50, 105, 92]. These systems have also been used as an abstract Turing-complete model of computation, in the form of Post's tag systems [102]. A typical problem of interest is reachability, which asks given two configurations of the system, whether the system can reach the second, starting from the first. A procedure to decide reachability is useful to decide safety properties, that is, to check if the system can ever reach a bad or unsafe configuration. As a result of Turing-completeness, reachability and other problems on general channel systems are undecidable. Finkel discusses several approaches to deal with this in [46]. One such is to require that every incoming message can always be "absorbed", that is, consumed and ignored. At least for the purposes of reachability, this is equivalent to a relaxation which allows channels to nondeterministically lose messages, as studied by Abdulla and Jonsson in [7]. They show that reachability is decidable for these *lossy channel systems*. This can also be seen as a consequence of Pachl's result from [97]. The lossiness models unreliability in channels, and can also be seen as an overapproximation of the behaviours of the system with reliable channels. Other kinds of unreliability such as duplication and insertion errors, and their combinations with each other and message losses are considered in [26]. Unreliability has been considered in other models of computation too, such as noisy Turing machines [11].

Lossy channel systems are a "weak" model of computation, but can compute exactly in certain restricted settings, for example with space bounded by F_{ω^ω} [30], or with probabilistic losses [3]. Channel systems play a central role in some areas of program and system verification (see [26, 4, 87, 13]) and also provide decidable automata models for problems on Real-Time and Metric Temporal Logic, modal logics, and data logics [6, 95, 86, 85, 89, 16]. An example of a channel system is shown in Figure 1.1.

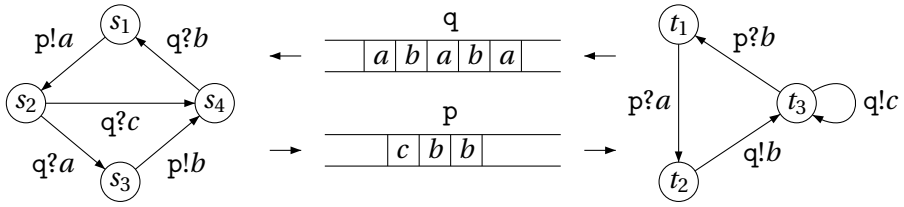


Figure 1.1: A snapshot of a channel system with two components and two channels

Several real world protocols have been modelled using lossy channel systems. An example is the Alternating Bit Protocol (ABP) [17], which allows a sender and a receiver to strictly alternate messages and acknowledgements by using parity tags, even when the channels are lossy. The High Level Data Link Control (HDLC) procedures are another example [69]. In [2], the Bounded Retransmission Protocol (BRP) is modelled as a lossy channel system and verified using a tool LCS developed by the authors. The BRP is a data link protocol which can be seen as an extended version of the ABP.

LCSes have also been used as a decidable model for solving other problems. The verification of finite-state concurrent programs running under weak or relaxed memory models is achieved by reducing the problem to reachability for LCS in [12]. An example of a weak memory model is one where in a single thread of execution, a write to a variable may be postponed past a read from a different variable. In a multithreaded context, this can introduce new program behaviours, making analysis nontrivial.

1.3 Well-structured transition systems

Valk and Jantzen had observed in 1985 [118] that "monotonicity" was key to many decision procedures on Petri nets and vector addition systems. Finkel distilled the key monotonicity, well-quasi-order, finiteness, an decidability properties that make the Karp-Miller coverability tree algorithm [80] work, and used these ideas to extend the Karp-Miller result to "well-structured transition systems" (WSTS), introduced by him [43, 45]. A WSTS is a transition system with a well-quasi-order[4] between the configurations which is compatible with the transitions. Lossy channel systems and vector addition systems are typical examples. Finkel applied the WSTS theory to lossy channel systems where messages can be ignored [46]. In the meanwhile, Abdulla and Jonsson proposed a model of lossy channel systems which models lossiness more directly, and a backward exploration regular model checking algorithm for reachability [7]. In [4] they use loop acceleration for forward analysis of lossy channel systems, and implement it in a tool TREX. Both the backward and forward analysis approaches

[4] section 2.2

are examples of regular model checking. In [48] Finkel and Schnoebelen streamline and unify notions of WSTS with a cleaner, simpler definition which clearly separates structure and effectiveness issues, leading to greater clarity and simpler algorithms.

Recent developments in the theory of WSTS include the study of comparative expressiveness [5], and the completion technique for forward-chaining [47]. When it comes to applications, many new WSTS models have been introduced, in distributed computing, verification, and other fields. These use well-quasi-orderings based on trees, sequences of vectors, and graphs (see references in [113]), while vectors of natural numbers and words with the subword relation were the staple of traditional examples. A survey of the history of well-structured transition systems is found in [114] and [1]. The complexity analysis of WSTS models and algorithms is another recent development, borrowing nontrivial concepts from proof theory and ordinal analysis [30, 76, 55, 115].

1.4 Fast-growing complexity, LCS, and UCS

The reachability problem on LCS, though decidable, has astronomically high complexity, in the complexity class $\mathsf{F}_{\omega^\omega}$, well above primitive-recursive functions. Even specifying this complexity needs nontrivial definitions of ordinal-recursive hierarchies such as the fast-growing hierarchy[76]. Such hierarchies have been used to classify recursive functions in terms of their growth rate. Proof theory is used to measure the complexity of recursive functions based on the logical complexity of proofs that they terminate in [42], and the slow-growing and fast-growing hierarchies arise naturally. In [103] fast-growing functions are used to measure strengths of axiom systems, for example producing a first-order truth of natural numbers not provable in Peano arithmetic. However, the use of fast-growing functions to actually characterize the complexity of problems is more recent [30, 55, 111, 76].

In recent years many problems have been found with the same complexity as reachability for LCS [96, 12, 89]. The quest to find a "master problem" for this class, much like SAT is for NP, led to the definition of the Post Embedding Problem (PEP) [27]. The problem is shown decidable and a connection with LCSes is established in [27]. In chapter 5 we introduce a more general problem and show its decidability using simpler techniques. The $\mathsf{F}_{\omega^\omega}$ complexity for reachability in LCS is shown in [30], and the bound carries over to PEP.

LCSes have been used to show complexity lower bounds for other problems. For example, lower bounds for the satisfiability and model checking problems for metric temporal logic are shown in [96] using channel systems with insertion errors, which are just lossy channel systems running in reverse.

The decidability of reachability in systems with both lossy and reliable channels has been considered in [28]. The addition of reliable channels rapidly leads to undecidability. [28] classifies network topologies by decidability of the corresponding reachability problem. The

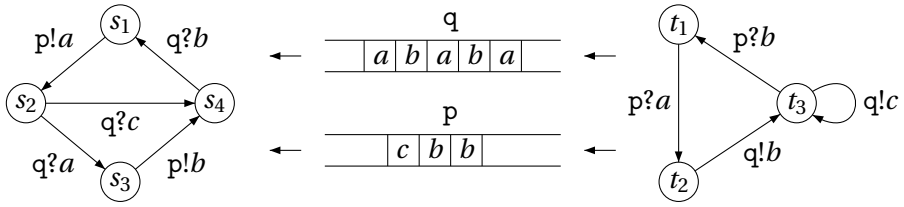


Figure 1.2: A snapshot of a unidirectional channel system with two components and two channels

two minimal topologies which result in decidability are the ones with no reliable channels (this is just LCS), and the topology consisting of two components, Sender and Receiver, and two parallel channels from Sender to Receiver, one reliable and one lossy. An example of such a "unidirectional channel system" (UCS) is shown in Figure 1.2, if we consider one of the two channels to be reliable and the other to be lossy. Another motivation for studying unidirectional channel systems is that they arise naturally in the connection between PEP and LCS in [27].

In channel systems, *channel tests* are guard transitions which can be taken only when the contents of a given channel satisfy a given constraint. Channel tests do not pose a challenge to the verification of LCSes as LCSes with tests can be simulated with LCSes without tests. Channel tests are convenient in practice, when describing protocols, but they are omitted for simplification when analyzing the model mathematically. The situation is different with UCSes as channel tests are a nontrivial addition to the model. Tests can be used to implement limited "backward" communication from Receiver to Sender, and can express new behaviours which cannot be simulated without tests. We investigate channel tests in chapter 6.

1.5 Our contributions

In the context of regular model checking for lossy channel systems, upward and downward closed sets of configurations and their representations are fundamental. This brings up some basic automata-theoretic questions concerning the subword relation on words, both state complexity questions and decision problems. Surprisingly, these very basic questions have not been considered, or not in detail, in the relevant literature. Apart from our regular model checking motivations, applications also exist in data processing and bioinformatics [110, 15]. In chapter 3 we show tight state complexity bounds on closures, with both small and unbounded alphabets, introducing simpler examples and some new results. Our applications also suggest other questions, such as computing interiors, which we address in detail. In trying to solve these questions we encountered the question of estimating the index of Simon's congruence which relates two words iff they have the same subwords of length upto a

given threshold [108]. This was a long-standing open problem and even whether the index is asymptotically single- or double- exponential (or somewhere in between) was not known, in spite of the index being relevant to questions involving piecewise testable languages. We give asymptotically tight bounds on this index in chapter 4.

In chapter 5 we define a problem $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ which is a common generalization of both PEP and $\mathsf{PEP}_{\text{codir}}$ defined in [27], and show its decidability using techniques simpler than those used earlier. We also study further variants and extensions on both sides of the decidability frontier. In chapter 6 we consider UCSes with tests. We show the decidability of reachability with the addition of simple emptiness and nonemptiness tests, by a nontrivial sequence of reductions to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$. We also show that some further extensions and other kinds of tests easily lead to undecidability.

1.6 Acknowledgements

Chapter 3 is based on [75], co-authored with Niewerth and Schnoebelen, which is the full version of [79]. Chapter 4 is based on [74], co-authored with Kufleitner and Schnoebelen. Chapter 5 is based on [78], co-authored with Schnoebelen, which is the full version of [77]. Chapter 6 is based on [71], co-authored with Jančar and Schnoebelen, which is the full version of [70].

The author was partially funded by Tata Consultancy Services.

Chapter 2

Technical preliminaries

We introduce some technical preliminaries in this chapter.

For an integer n, we use $[n]$ to denote $\{m \in \mathbb{N} : 0 \leq m < n\}$. For Σ a finite alphabet, a (finite) *word* over Σ is a finite sequence $w = a_0 a_1 \dots a_{\ell-1}$ of elements of Σ. ℓ is called the *length* of w, also denoted as $|w|$. For $i \in [|w|]$, $w[i]$ denotes a_i. We use $w_1 w_2$ to denote the concatenation of w_1 with w_2 and ϵ for the empty word. The set of all finite words over Σ is denoted by Σ^*. This forms a monoid under concatenation, with ϵ as the identity. This is the free monoid over Σ. For a word $s = a_0 \dots a_{\ell-1}$, $\widetilde{s} \stackrel{\text{def}}{=} a_{\ell-1} \dots a_0$ is the mirrored word. The mirror of a language R is $\widetilde{R} \stackrel{\text{def}}{=} \{\widetilde{s} : s \in R\}$.

2.1 Subwords and embeddings

Definition 2.1. For words v, w, an *embedding* of v into w is a strictly increasing function $f : [|v|] \to [|w|]$ such that for all $i \in [|v|]$, $v[i] = w[f(i)]$.

Figure 2.1 shows two embeddings of $abca$ into $cabcbca$.

Definition 2.2. For words v, w, we say v is a *subword* of w if there exists an embedding of v into w. This is denoted as $v \sqsubseteq w$.

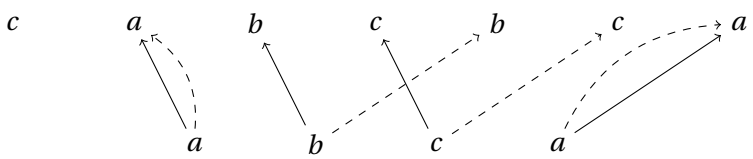


Figure 2.1: The solid and dashed lines show the leftmost and rightmost embedding respectively of $abca$ into $cabcbca$.

If v is a subword of w, there might be several embeddings of v into w, but there are two special ones: the leftmost and the rightmost embedding. If f_1 and f_2 are two embeddings of v into w, we say f_1 is to the left of f_2 (equivalently, f_2 is to the right of f_1) if for all i, $f_1(i) \leq f_2(i)$.

Lemma 2.3. If v is a subword of w, then there exists a unique *leftmost embedding* of v into w, that is, an embedding which is to the left of all embeddings of v into w. Similarly, there exists a unique *rightmost embedding* of v into w, that is, an embedding which is to the right of all embeddings of v into w.

In fact, a natural left-to-right greedy algorithm to test whether a given word is a subword of another given word will produce the leftmost embedding, if an embedding exists.

The subword relation $\sqsubseteq$ is a reflexive and transitive relation. It is compatible with concatenation in several ways, as we see below.

Lemma 2.4. For all words y, z, s, t, if $y \sqsubseteq s$ and $z \sqsubseteq t$, then $yz \sqsubseteq st$. Moreover, if $yz \sqsubseteq st$, then $y \sqsubseteq s$ *or* $z \sqsubseteq t$.

Definition 2.5. If $w = w_1 w_2$, then w_1 is a *prefix* of w and w_2 is a *suffix* of w.

A word of length n has exactly $n+1$ prefixes, one each of length $0, \ldots, n$, and similarly has exactly $n+1$ suffixes, one of every length.

Definition 2.6. If w_1 is a prefix of w, $w_1^{-1} w$ is defined to be the unique w_2 such that $w = w_1 w_2$, and is undefined otherwise. Similarly, if w_2 is a suffix of w, $w w_2^{-1}$ is defined to be the unique w_1 such that $w = w_1 w_2$, and is undefined otherwise.

For example, $(abb)^{-1} abbab = ab$.

The following four lemmas involve "greedy" factorizations and can all be proven by considering rightmost embeddings.

Lemma 2.7. If $yz \sqsubseteq st$ and $z \sqsubseteq t$ and x is the longest suffix of y such that $xz \sqsubseteq t$, then $yx^{-1} \sqsubseteq s$.

Lemma 2.8. If $yz \sqsubseteq st$ and $z \not\sqsubseteq t$ and x is the shortest prefix of z such that $x^{-1} z \sqsubseteq t$, then $yx \sqsubseteq s$.

Lemma 2.9. If $yz \sqsubseteq st$ and $z \sqsubseteq t$ and x is the longest prefix of t such that $z \sqsubseteq x^{-1} t$, then $y \sqsubseteq sx$.

Lemma 2.10. If $yz \sqsubseteq st$ and $z \not\sqsubseteq t$ and x is the shortest suffix of s such that $z \sqsubseteq xt$, then $y \sqsubseteq sx^{-1}$.

These lemmas combining subwords and iteration will be useful later:

Lemma 2.11. If $sx \sqsubseteq yt$ and $t \sqsubseteq s$, then $sx^k \sqsubseteq y^k t$ for all $k \geq 1$.

Proof. By induction on k. The statement holds for $k = 1$. Suppose it holds for $k = p$. Then

$$sx^{p+1} = sx^p x \sqsubseteq y^p tx \sqsubseteq y^p sx \sqsubseteq y^p yt = y^{p+1} t$$ □

Lemma 2.12. If $xs \sqsubseteq ty$ and $t \sqsubseteq s$, then $x^k s \sqsubseteq ty^k$ for all $k \geq 1$.

Proof. By induction on k. The statement holds for $k = 1$. Suppose it holds for $k = p$. Then

$$x^{p+1} s \sqsubseteq xx^p s \sqsubseteq xty^p \sqsubseteq xsy^p \sqsubseteq tyy^p = ty^{p+1}$$ □

2.2 Well-quasi-orders

Definition 2.13. $(X, \leq)$ is a *quasi-order* (qo) if $\leq$ is a reflexive transitive relation on X.

For a qo $(X, \leq)$, we use $x < y$ to denote "$x \leq y$ and $y \not\leq x$". A qo is said to be a *partial order* if it is antisymmetric, that is, for all x, y, $x \leq y$ and $y \leq x$ implies $x = y$. The equivalence associated with a qo $(X, \leq)$ is the relation $\sim$ given by $x \sim y$ iff $x \leq y$ and $y \leq x$.

We now give three definitions for the concept of a *well-quasi-order* (wqo), and show that they are equivalent.

Definition 2.14 (wqo1)**.** A qo $(X, \leq)$ is said to be a wqo1 if for every infinite sequence $x_1, x_2, \ldots$ from X, there exist $i < j$ such that $x_i \leq x_j$.

Definition 2.15 (wqo2)**.** A qo $(X, \leq)$ is said to be a wqo2 if for every infinite sequence $x_1, x_2, \ldots$ from X, there exist infinitely many indices $i_1 < i_2 < \ldots$ such that, for every k, $x_{i_k} \leq x_{i_{k+1}}$.

Definition 2.16 (wqo3)**.** A qo $(X, \leq)$ is said to be a wqo3 if there is no infinite strictly decreasing sequence from X and no infinite antichain in X (an antichain is a set of pairwise incomparable elements).

Lemma 2.17. A qo $(X, \leq)$ is a wqo1 iff it is a wqo2 iff it is a wqo3.

Proof. Clearly wqo2 implies wqo1 and wqo1 implies wqo3. We now show wqo3 implies wqo2, to complete the argument.

Let $x_1, x_2, \ldots$ be an infinite sequence from X. Let $S = \{i : \text{not } (\exists j > i\, x_i \leq x_j)\}$. If S is finite, one gets an infinite increasing subsequence starting from the index $1 + \max(S)$. Otherwise, S is infinite. Let $y_1, y_2, \ldots$ be the subsequence of $x_1, x_2, \ldots$ given by restricting to indices from S. that for all i, j with $i < j$, $y_i \not\leq y_j$.

We now repeat a similar argument. Define $x \# y$ to mean "(not $x \leq y$) and (not $y \leq x$)". Let $T = \{i : \forall j > i\ x_i \# x_j\}$. If T is infinite, we get a contradiction to wqo3. Otherwise, let $i_1 = 1 + \max(T)$. Then there exists $i_2 > i_1$ such that $y_{i_1} \leq y_{i_2}$ or $y_{i_2} \leq y_{i_1}$. But the former is not possible, so we have $y_{i_2} < y_{i_1}$. Continuing this way, we get a strictly decreasing sequence $y_{i_1} > y_{i_2} > y_{i_3} > \ldots$, which contradicts wqo3.

One can also prove this by a direct application of the infinite Ramsey theorem. □

Definition 2.18. A qo $(X, \leq)$ is said to be a *well-quasi-order* if it is any of wqo1, wqo2, or wqo3.

Every finite set with the equality relation is a wqo. One of the simplest infinite wqos is the set of natural numbers with the usual ordering: for any infinite sequence $n_1, n_2, \ldots$, let n_i be the smallest element which occurs in the sequence, then the indices i and $i+1$ form an increasing pair.

The class of wqos is closed under several operations, for example:

Lemma 2.19 (Dickson's Lemma)**.** If $(X_1, \leq_1)$ and $(X_2, \leq_2)$ are wqos, so is their product $(X_1 \times X_2, \leq_\times)$, where the product ordering is defined componentwise. That is, $(a, b) \leq_\times (c, d) \stackrel{\text{def}}{\Leftrightarrow} a \leq c$ and $b \leq d$.

Proof. Given an infinite sequence in $X_1 \times X_2$, we extract an infinite sequence which is increasing in the first component, and from that, an infinite sequence that is increasing in the second component. □

Thus $\mathbb{N}^k$ with the componentwise ordering is a wqo for every k. This is central to the analysis of Petri nets, vector addition systems, and counter machines.

For a qo $(X, \leq)$, its *sequence extension* is defined to be the qo $(X^*, \leq_*)$, where X^* is the set of all finite words over X, and $v \leq_* w$ iff there exists a strictly increasing $f : [|v|] \to [|w|]$ such that for all i, $v[i] \leq w[f(i)]$. In case $\leq$ is the equality relation, $\leq_*$ is simply the subword relation defined earlier (Definition 2.2).

Theorem 2.20 (Higman's Lemma, [60])**.** If $(X, \leq)$ is a wqo, so is $(X^*, \leq_*)$.

Proof. We say that an infinite sequence $w_1, w_2, \ldots$ of words in X^* is *bad* if it has no increasing pair, that is, no $i < j$ such that $w_i \leq_* w_j$. Our goal is to show there are no bad sequences (we are using wqo1 here). For the sake of contradiction, suppose there exists a bad sequence. We will try to identify the "smallest" bad sequence.

Let w_1 be a word of minimum length such that there exists a bad sequence starting from w_1 (this must exist). Having picked $w_1, w_2, \ldots, w_k$, a beginning of some bad sequence, pick w_{k+1} of smallest length such that there exists a bad sequence starting with $w_1, \ldots, w_{k+1}$. Thus we get an infinite sequence $w \stackrel{\text{def}}{=} w_1, w_2, \ldots$. For all $i < j$, since there exists a bad sequence beginning with $w_1, \ldots, w_j$, we have $w_i \not\leq_* w_j$, and thus the sequence w itself is bad.

Clearly no element of this sequence is ϵ, as ϵ cannot belong to a bad sequence. Write each w_i as $a_i v_i$, where $a_i \in X$ and $v_i \in X^*$. Since $(X, \leq)$ is a wqo, there exists an infinite sequence $i_1 < i_2 < \ldots$ such that $a_{i_1} \leq a_{i_2} \leq \ldots$ (we are using wqo2 here). Now let z_j be defined as follows:

$$z_j = \begin{cases} w_j & \text{if } j < i_1 \\ v_{i_{j-i_1+1}} & \text{otherwise} \end{cases}$$

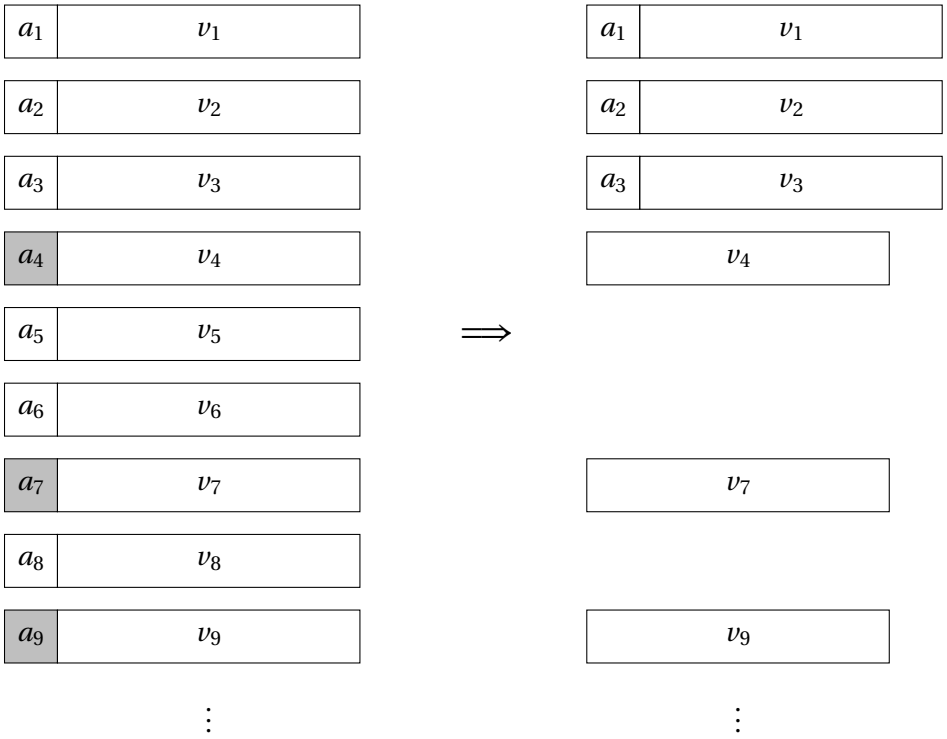


Figure 2.2: Getting a "smaller" bad sequence. The left and right columns show the sequences w and z respectively. In this example we have $a_4 \leq a_7 \leq a_9 \leq \ldots$.

An example of this construction is shown in Figure 2.2.

It is easy to see that the sequence $z_1, z_2, \ldots$ is a bad sequence. However, its first $i_1 - 1$ terms are the same as those of the sequence w, and $|z_{i_1}| = |w_{i_1}| - 1$, which contradicts the way w was chosen. □

A concise self-contained proof of Higman's Lemma is hard to find; the above proof is due to Nash-Williams [93]. Notice that the above proof relies on the equivalence of wqo1 and wqo2 established by Lemma 2.17. The most important consequence for us is the case when $(X, \leq)$ is $(\Sigma, =)$, a finite alphabet with the equality relation:

Corollary 2.21. For Σ a finite alphabet, the subword relation on Σ^* is a wqo.

Definition 2.22. For a qo $(X, \leq)$, a subset S of X is said to be *upward closed* if for all x, y, if $x \in X$ and $x \leq y$, then $y \in X$. Similarly, S is said to be *downward closed* if for all x, y with $x \in X$ and $y \leq x$, we have $y \in X$.

For example, in $\{a, b, c\}^*$ with the subword relation as the qo, the set of all words of length at least 7 is upward closed, while $(a + b)^*(c + \epsilon) + c^*$ is downward closed.

Definition 2.23. For a qo $(X, \leq)$ and $S \subseteq X$, the *upward closure* of S (denoted $\uparrow S$) is the smallest upward closed set which includes S and the *downward closure* of S (denoted $\downarrow S$) is the smallest downward closed set which includes S. That is,

$$\uparrow S = \{x : \exists y \in S \; y \leq x\}$$
$$\downarrow S = \{x : \exists y \in S \; x \leq y\}$$

For a singleton set, we may write $\uparrow x$ and $\downarrow x$ for $\uparrow \{x\}$ and $\downarrow \{x\}$.

The following lemma tells us that we can finitely represent upward closed sets in a wqo:

Lemma 2.24. Let $(X, \leq)$ be a wqo, and let $S \subseteq X$ be upward closed. Then there exists a finite set $F \subseteq S$ such that $\uparrow F = S$.

Proof. The idea is to consider all minimal elements of X. However, there might be several, even infinitely many, equivalent minimal elements, and we need to pick only one of them.

Formally, define $x \in S$ to be *minimal* if there does not exist y such that $y < x$. Let M be the set of minimal elements of S. Observe that if x is minimal and y is equivalent to x under $\sim$, the equivalence associated to $\leq$, then y is minimal too. Thus M is a union of equivalence classes of $\sim$. Let F be obtained by picking an element from each equivalence class. F is finite, otherwise we would have infinitely many incomparable elements in X, which is not possible. Clearly $\uparrow F \subseteq S$ since S is upward closed. For the reverse inclusion, consider any $s \in S$. If s is minimal, s is equivalent to some element of F, and hence in $\uparrow F$. Otherwise, there exists $s_1 \in S$ such that $s > s_1$. If s_1 is minimal, we are done, otherwise there exists s_2 such that $s > s_1 > s_2$. Continuing this way, this must eventually terminate, otherwise we would have an infinite strictly decreasing chain in X, which is not possible. □

The following lemma is often used to prove termination of algorithms (for example, in [48]) :

Lemma 2.25. In a wqo, there exists no strictly increasing (by set inclusion) sequence of upward closed sets.

Proof. For the sake of contradiction, let $U_1 \subsetneq U_2 \subsetneq U_3 \ldots$ be a strictly increasing sequence of upward closed sets. For each i, let x_i be an element of $U_{i+1} \setminus U_i$. There exists i, j such that $i < j$ and $x_i \leq x_j$. $x_j \notin U_j$, and since U_j is upward closed, $x_i \notin U_j$. But $x_i \in U_{i+1} \subseteq U_j$, since $i + 1 \leq j$. This is a contradiction. □

Chapter 3

State complexity and related questions

Quoting from [121], "*State complexity problems are a fundamental part of automata theory that has a long history. [...] However, many very basic questions, which perhaps should have been solved in the sixties and seventies, have not been considered or solved.*"

In this chapter, we are concerned with *(scattered) subwords* and the associated operations on regular languages: computing closures and interiors (see definitions in Section 3.1). Our motivations come from automatic verification of channel systems, see, e.g., [4, 53]. Other applications exist in data processing or bioinformatics [15]. Closures and interiors with respect to subwords and superwords are very basic operations, and the above quote certainly applies to them.

It has been known since [56] that $\downarrow L$ and $\uparrow L$, the downward closure and, respectively, the upward closure, of a language $L \subseteq \Sigma^*$, are regular for any L.

In 2009, Gruber *et al.* explicitly raised the issue of the state complexity of downward and upward closures of regular languages [52] (less explicit precursors exist, e.g. [19]). Given an n-state automaton A, constructing automata $A^{\downarrow}$ and $A^{\uparrow}$ for $\downarrow L(A)$ and $\uparrow L(A)$ respectively can be done by simply adding extra transitions to A. However, when A is a DFA, the resulting automaton is in general not deterministic, and its determinization may entail an exponential blowup in general. Gruber *et al.* proved a $2^{\Omega(\sqrt{n}\log n)}$ lower bound on the number of states of any DFA for $\downarrow L(A)$ or $\uparrow L(A)$, to be compared with the $2^n - 1$ upper bound that comes from the simple "closure followed by determinization" algorithm.

Okhotin improved on these results by showing an improved $2^{\frac{n}{2}-2}$ lower bound for $\downarrow L(A)$. He also established the precise state complexity of $\uparrow L(A)$ by showing a $2^{n-2}+1$ upper bound and proving its tightness [94].

All the above lower bounds assume an unbounded alphabet, and Okhotin in fact showed that his $2^{n-2}+1$ lower bound on state complexity for $\uparrow L(A)$ does not hold with fewer than $n-2$ distinct letters. He then considered the case of languages over a *fixed alphabet* with $|\Sigma| = 3$

letters, in which case he demonstrated exponential $2^{\sqrt{2n+30}-6}$ and $\frac{1}{5}4^{\sqrt{n/2}}n^{-\frac{3}{4}}$ lower bounds for $\downarrow L(A)$ and, respectively, $\uparrow L(A)$ [94]. The construction and the proof are quite involved, and they leave open the case where $|\Sigma| = 2$ (the 1-letter case is trivial). It turns out that, in the 2-letter case, Héam had previously proved an $\Omega(r^{\sqrt{n}})$ lower bound for $\uparrow L(A)$, here with $r = (\frac{1+\sqrt{5}}{2})^{\frac{1}{\sqrt{2}}}$ [57]. Regarding $\downarrow L(A)$, the question remains open whether it may require an exponential number of states even when $|\Sigma| = 2$.

Dual to closures are *interiors*. The *upward interior* and *downward interior* of a language L, denoted $\left(\uparrow\right)L$ and $\left(\downarrow\right)L$, are the largest upward-closed and, resp., downward-closed, sets included in L. Building closures and interiors are essential operations when reasoning with subwords, e.g., when model-checking lossy channel systems [18]. More generally, one may use closures as regular overapproximations of more complex languages (as in [54, 14]), and interiors can be used as regular underapproximations.

The state complexity of interiors has not yet been considered in the literature. When working with DFAs, complementation is essentially free so that computing interiors reduces to computing closures, thanks to duality. However, when working with NFAs, the simple complement+closure+complement algorithm only yields a quite large 2^{2^n} upper-bound on the number of states of an NFA for $\left(\uparrow\right)L(A)$ or $\left(\downarrow\right)L(A)$ —it actually yields DFAs— and one would like to improve on this, or to prove a matching lower bound.

Our contribution. Regarding closures with DFAs, we prove in section 3.2 a tight 2^{n-1} state complexity for downward closure and show that its tightness requires unbounded alphabets. In section 3.3 we prove an exponential lower bound on $\downarrow L(A)$ in the case of a two-letter alphabet, answering the open question raised above.

Regarding interiors on NFAs, we show in section 3.4 doubly-exponential lower bounds for downward and upward interiors, assuming an unbounded alphabet. We also provide improved upper bounds, lower than the naive 2^{2^n} but still doubly exponential. Table 3.1 shows a summary of the known results.

Finally, we provide in section 3.5 the computational complexity of basic decision problems for sets of subwords or superwords described by automata.

Related work. We already mentioned previous works on the closure of regular languages: it is also possible to compute closures by subwords or superwords for larger classes like context-free languages or Petri net languages, see [54, 14, 123] and the references therein for applications and some results on descriptive complexity.

Interiors and other duals of standard operations have the form "*complement, operation, complement*" and thus can be seen as special cases of the *combined operations* studied in [109] and following papers. Such duals have not yet been considered widely: we are only aware of [20] studying the dual of $L \mapsto \Sigma^* \cdot L$.

Table 3.1: A summary of the results on state complexity for closures and nondeterministic state complexity for interiors, where $\psi(n)$ ($\leq 2^{2^n}$) is the nth Dedekind's number, see subsection 3.4.1.

Operation	Unbounded alphabet	Fixed alphabet
$\uparrow L$ (DFA to DFA)	$= 2^{n-2}+1$ with $\lvert\Sigma\rvert \geq n-2$	$2^{\Omega(n^{1/2})}$ for $\lvert\Sigma\rvert = 2$
$\downarrow L$ (DFA to DFA)	$= 2^{n-1}$ with $\lvert\Sigma\rvert \geq n-1$	$2^{\Omega(n^{1/3})}$ for $\lvert\Sigma\rvert = 2$
$\curvearrowleft L$ (NFA to NFA)	$\geq 2^{2^{\lfloor (n-4)/3 \rfloor}}+1$ and $\leq \psi(n)$	
$\curvearrowright L$ (NFA to NFA)	$\geq 2^{2^{\lfloor (n-3)/2 \rfloor}}$ and $\leq \psi(n)$	

3.1 Preliminaries

Recall that for a language $L \subseteq \Sigma^*$, its downward closure is $\downarrow L \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid \exists y \in L : x \sqsubseteq y\}$, and its upward closure operation is $\uparrow L \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid \exists y \in L : y \sqsubseteq x\}$. Closures enjoy the following properties:

$$\downarrow\emptyset = \emptyset\,, \quad L \subseteq \downarrow L = \downarrow\downarrow L\,, \quad \downarrow\left(\bigcup_i L_i\right) = \bigcup_i \downarrow L_i\,, \quad \downarrow\left(\bigcap_i \downarrow L_i\right) = \bigcap_i \downarrow L_i\,,$$

and similarly for upward closures. A language L is *downward-closed* (or *upward-closed*) if $L = \downarrow L$ (respectively, if $L = \uparrow L$). Note that L is downward-closed if, and only if, $\Sigma^* \setminus L$ is upward-closed.

Upward-closed languages are also called *shuffle ideals* since they satisfy $L = \texttt{shuffle}(L, \Sigma^*)$. They correspond exactly to level $\frac{1}{2}$ of Straubing's hierarchy [99].

Since, by Higman's Lemma, any L has only finitely many minimal elements with respect to the subword ordering, one deduces that $\uparrow L$, and then $\downarrow L$, are regular for any L.

Effective construction of a finite-state automaton for $\downarrow L$ or $\uparrow L$ is easy when L is regular (see section 3.2), is possible when L is context-free [119, 37], and is not possible in general since this would allow deciding the emptiness of L.

The *upward interior* of L is $\curvearrowleft L \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid \uparrow x \subseteq L\}$. Its *downward interior* is $\curvearrowright L \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid \downarrow x \subseteq L\}$. Alternative characterizations are possible, e.g., by noting that $\curvearrowleft L$ (respectively, $\curvearrowright L$) is the largest upward-closed (respectively, downward-closed) language contained in L, or by using the following dualities:

$$\curvearrowright L = \Sigma^* \setminus \uparrow(\Sigma^* \setminus L)\,, \qquad \curvearrowleft L = \Sigma^* \setminus \downarrow(\Sigma^* \setminus L)\,. \tag{3.1}$$

If L is regular, one may compute automata for the interiors of L by combining complementations and closures as in Eq. (3.1).

State complexity When considering a finite automaton $A = (\Sigma, Q, \delta, I, F)$, we usually write n for $|Q|$ (the number of states), k for $|\Sigma|$ (the size of the alphabet), and $L(A)$ for the language recognized by A. For a regular language L, $n_{\mathrm{D}}(L)$ and $n_{\mathrm{N}}(L)$ denote the minimum number of states of a DFA (resp., an NFA) that accepts L. (We do not assume that DFAs are complete and this sometimes allows saving one (dead) state.) Obviously $n_{\mathrm{N}}(L) \leq n_{\mathrm{D}}(L)$ for any regular language. In cases where $n_{\mathrm{N}}(L) = n_{\mathrm{D}}(L)$ we may use $n_{\mathrm{N\&D}}(L)$ to denote the common value.

We now illustrate a well-known technique for proving lower bounds on $n_{\mathrm{N}}(L)$:

Lemma 3.1 (Extended fooling set technique, [51])**.** Let L be a regular language. Suppose that there exists a set of pairs of words $S = \{(x_i, y_i)\}_{1 \leq i \leq n}$, called a *fooling set*, such that for all i, j, $x_i\, y_i \in L$ and at least one of $x_i\, y_j$ and $x_j\, y_i$ is not in L. Then $n_{\mathrm{N}}(L) \geq n$.

Proof. Let M be an NFA for L. For each i, $x_i y_i \in L$, so M has an accepting path $* \xrightarrow{x_i} q_i \xrightarrow{y_i} *$ starting at some initial state and ending at some accepting state, for some state q_i. The states $q_1, q_2, \ldots, q_n$ are all distinct: indeed, if $q_i = q_j$ for $i \neq j$ then M has accepting paths for both $x_i y_j$ and $x_j y_i$, which contradicts the assumption. □

Lemma 3.2 (An application of the fooling set technique)**.** Fix a nonempty alphabet Σ and define the following languages:

$$U_\Sigma \stackrel{\text{def}}{=} \{x \mid \forall a \in \Sigma : \exists i : x[i] = a\}, \qquad U'_\Sigma \stackrel{\text{def}}{=} \{x \mid \forall a \in \Sigma : \exists i > 0 : x[i] = a\}, \tag{3.2}$$

$$V_\Sigma \stackrel{\text{def}}{=} \{x \mid \forall i \neq j : x[i] \neq x[j]\}. \tag{3.3}$$

Then $n_{\mathrm{N\&D}}(U_\Sigma) = n_{\mathrm{N\&D}}(V_\Sigma) = 2^{|\Sigma|}$ and $n_{\mathrm{N\&D}}(U'_\Sigma) = 2^{|\Sigma|} + 1$.

Note that U_Σ has all words where every letter in Σ appears at least once, U'_Σ has all nonempty words x where every letter in Σ appears at least once in the first proper suffix $x[1..]$ while V_Σ has all words where no letter appears twice. U_Σ and U'_Σ are upward-closed while V_Σ is downward-closed.

Proof. It can easily be observed that the upper bounds hold for $n_{\mathrm{D}}(..)$: one designs DFAs A_U and A_V for, respectively, U_Σ and V_Σ where each state is some subset $\Gamma \subseteq \Sigma$ that corresponds to the set of letters that have been read so far. In both automata, the initial state is $\emptyset$. In A_U, $\delta(\Gamma, a) = \Gamma \cup \{a\}$ and we accept when we reach the state Σ. In A_V, all states are accepting but $\delta(\Gamma, a) = \Gamma \cup \{a\}$ is only defined when $a \notin \Gamma$. A DFA for U'_Σ is obtained from A_U by adding one additional state to indicate that no letter has been read so far.

We now show the lower bounds for $n_{\mathrm{N}}(U_\Sigma)$ and $n_{\mathrm{N}}(V_\Sigma)$. With any $\Gamma \subseteq \Sigma$, we associate two words x_Γ and y_Γ, where x_Γ (respectively, y_Γ) has exactly one occurrence of each letter from Γ (respectively, each letter not in Γ). Then $x_\Gamma y_\Gamma$ is in U_Σ and V_Σ, while for any $\Delta \neq \Gamma$ one of $x_\Gamma y_\Delta$ and $x_\Delta y_\Gamma$ is not in U_Σ (and one is not in V_Σ). We may thus let $S = \{(x_\Gamma, y_\Gamma)\}_{\Gamma \subseteq \Sigma}$ be our fooling set and conclude with Lemma 3.1.

For U'_Σ our fooling set will be $S = \{(a\,x_\Gamma, y_\Gamma)\}_{\Gamma\subseteq\Sigma} \cup \{(\epsilon, a\,x_\Sigma)\}$ where a is a fixed letter from Σ. As above, $a\,x_\Gamma\,y_\Gamma$ is in U'_Σ, while for any $\Delta \neq \Gamma$ one of $a\,x_\Gamma\,y_\Delta$ and $a\,x_\Delta\,y_\Gamma$ is not in U'_Σ. Furthermore $\epsilon \cdot a\,x_\Sigma$ is in U'_Σ, while $\epsilon \cdot y_\Gamma$ is not in U'_Σ for any Γ. One concludes again with Lemma 3.1. □

In the following, we use $\Sigma_k \stackrel{\text{def}}{=} \{a_1, \ldots, a_k\}$ to denote a k-letter alphabet, and write U_k and V_k instead of U_{Σ_k} and V_{Σ_k}.

3.2 State complexity of closures

For a regular language L recognized by an NFA A, one may obtain NFAs for the upward and downward closures of L by simply adding transitions to A, without increasing the number of states. More precisely, an NFA $A^\uparrow$ for $\uparrow L$ is obtained by adding to A self-loops $q \xrightarrow{a} q$ for every state q of A and every letter $a \in \Sigma$. Similarly, an NFA $A^\downarrow$ for $\downarrow L$ is obtained by adding to A epsilon transitions $p \xrightarrow{\epsilon} q$ for every transition $p \to q$ of A (on any letter).

3.2.1 Deterministic automata for closures

If now L is recognized by a DFA or an NFA A and we want a DFA for $\uparrow L$ or for $\downarrow L$, we can start with the NFA $A^\uparrow$ or $A^\downarrow$ defined above and transform it into a DFA using the powerset construction. This shows that if L has an n-state DFA, then both its upward and downward closures have DFAs with at most $2^n - 1$ states.

It is actually possible to provide tighter upper bounds by taking advantage of specific features of $A^\uparrow$ and $A^\downarrow$.

Proposition 3.3 (State complexity of upward closure)**.** 1. For an n-state NFA A, $n_{\mathrm{D}}(\uparrow L(A)) \leq 2^{n-2} + 1$.
2. Furthermore, for any $n > 1$ there exists a language L_n with $n_{\mathrm{N\&D}}(L_n) = n$ and $n_{\mathrm{D}}(\uparrow L_n) = 2^{n-2} + 1$.

Proof. 1. Let $A = (\Sigma, Q, \delta, I, F)$ be an n-state NFA for $L = L(A)$. We assume that $I \cap F = \emptyset$ (and $I \neq \emptyset \neq F$) otherwise L contains ϵ (or is empty) and $\uparrow L$ is trivial.

Since $A^\uparrow$ has loops on all its states and for any letter, applying the powerset construction yields a DFA where $P \xrightarrow{a} P'$ implies $P \subseteq P'$, hence any state P reachable from I includes I. Furthermore, if P is accepting (i.e., $P \cap F \neq \emptyset$) and $P \xrightarrow{a} P'$, then P' is accepting too, hence all accepting states recognize exactly Σ^* and are equivalent. Then there can be at most $2^{|Q\setminus(I\cup F)|}$ states in the powerset automaton that are both reachable and not accepting. To this we add 1 for the accepting states since they are all equivalent. Finally $n_{\mathrm{D}}(\uparrow L) \leq 2^{n-2} + 1$ since $|I \cup F|$ is at least 2 as we observed.

2. To show that $2^{n-2}+1$ states are sometimes necessary, we assume $n > 2$ and define $L_n \stackrel{\text{def}}{=} E_{n-2}$ where

$$E_k \stackrel{\text{def}}{=} \{a\,a \mid a \in \Sigma_k\} = \{a_1\,a_1, \ldots, a_k\,a_k\}. \tag{3.4}$$

In other words, L_n contains all words consisting of two identical letters from $\Sigma = \Sigma_{n-2}$. The minimal DFA for L_n has n states, see Fig. 3.1. Now $\uparrow L_n = \{x \in \Sigma^{\geq 2} \mid \exists j > i : x[i] = x[j]\} = \bigcup_{a\in\Sigma} \Sigma^* \cdot a \cdot \Sigma^* \cdot a \cdot \Sigma^*$, i.e., $\uparrow L_n$ has all words in Σ^* where *some letter reappears*, i.e., $\uparrow L_n$ is the complement of V_{n-2} from Lemma 3.2. A DFA for $\uparrow L_n$ has to record all letters previously

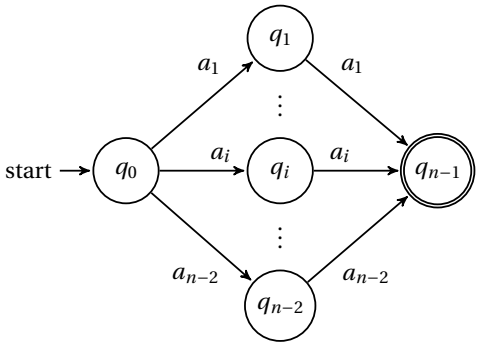


Figure 3.1: n-state DFA for $L_n = E_{n-2} = \{a_1\,a_1, a_2\,a_2, \ldots, a_{n-2}\,a_{n-2}\}$.

read in its non-accepting states, and has one accepting state: the minimal DFA has $2^{|\Sigma|}+1 = 2^{n-2}+1$ states. For obtaining $n_{\mathrm{N}}(L_n) = n$, $n_{\mathrm{N}}(L_n) \leq n$ is clear and we can infer $n_{\mathrm{N}}(L_n) \geq n$ just from $n_{\mathrm{D}}(\uparrow L_n) = 2^{n-2}+1$ and the first part of the proposition.

When $n = 2$, taking $L_2 = \{a\}$ over a 1-letter alphabet witnesses both $n_{\mathrm{D}}(L_n) = n = 2$ and $n_{\mathrm{D}}(\uparrow L_n) = 2^{n-2}+1 = 2$. Finally, the $2^{n-2}+1$ bound is tight even for the upward closure of DFAs. □

Remark 3.4. The above Proposition essentially reproduces Lemma 4.3 from [94] except that we do not assume that A is a DFA.

Proposition 3.5 (State complexity of downward closure)**.** 1. If A is an n-state NFA with only one initial state (in particular when A is a DFA) then $n_{\mathrm{D}}(\downarrow L(A)) \leq 2^{n-1}$.
2. Furthermore, for any $n > 1$ there exists a language L'_n with $n_{\mathrm{D}}(L'_n) = n$ and $n_{\mathrm{D}}(\downarrow L'_n) = 2^{n-1}$.

Proof. 1. We assume, without loss of generality, that all states in $A = (\Sigma, Q, \delta, \{q_{\text{init}}\}, F)$ are reachable from the single initial state. From A one derives an NFA $A^{\downarrow}$ for $\downarrow L(A)$ by adding ϵ-transitions to A. With these ϵ-transitions, the language recognized from a state $q \in Q$ is a subset of the language recognized from q_{init}. Hence, in the powerset automaton obtained by determinizing $A^{\downarrow}$, all states $P \subseteq Q$ that contain q_{init} are equivalent and recognize exactly $\downarrow L(A)$. There also are 2^{n-1} states in 2^Q that do not contain q_{init}. Thus $2^{n-1}+1$ bounds the number of non-equivalent states in the powerset automaton of $A^{\downarrow}$, and this includes a sink

state (namely $\emptyset \in 2^Q$) that will be omitted in the canonical minimal DFA for $\downarrow L(A)$.

2. To show that 2^{n-1} states are sometimes necessary, we assume $n > 1$ and let $L'_n \stackrel{\text{def}}{=} D_{n-1}$ where

$$D_k \stackrel{\text{def}}{=} \{x \in \Sigma_k^+ \mid \forall i > 0 : x[i] \neq x[0]\} = \bigcup_{a \in \Sigma_k} a \cdot \left(\Sigma_k \setminus \{a\}\right)^* . \tag{3.5}$$

In other words, L'_n has all words in Σ_{n-1}^+ where *the first letter does not reappear*.

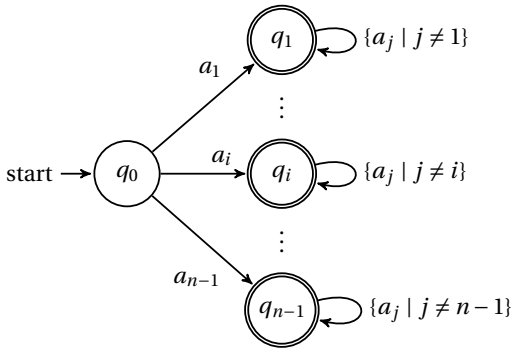


Figure 3.2: n-state DFA for $L'_n = D_{n-1} = \bigcup_{a \in \Sigma} a \cdot (\Sigma - \{a\})^*$ with $|\Sigma| = n - 1$.

The minimal DFA for L'_n has n states, see Figure 3.2. Now $\downarrow L'_n = \{x \mid \exists a \in \Sigma_{n-1} : \forall i \geq 1 : x[i] \neq a\}$, i.e., $\downarrow L'_n$ has all words x such that the first proper suffix $x[1..]$ does not use all letters. Equivalently $x \in \downarrow L'_n$ iff $x \in L'_n$ or x does not use all letters, i.e., $\downarrow L'_n$ is the union of L'_n and the complement of U_{n-1} from Lemma 3.2. The minimal DFA for $\downarrow L'_n$ just reads a first letter and then records all letters encountered after the first, hence needs exactly $2^{|\Sigma|}$ states. Thus 2^{n-1} states may be required for a DFA recognizing the downward closure of an n-state DFA. □

Remark 3.6. The condition of a single initial state in Proposition 3.5 cannot be lifted. In general $2^n - 1$ states may be required for a DFA recognizing the downward closure of an n-state NFA: the (downward-closed) language $\Sigma_n^* \setminus U_n$ of all words that do not use all letters is recognized by an n-state NFA (see Fig. 3.3) but its minimal DFA has $2^n - 1$ states.

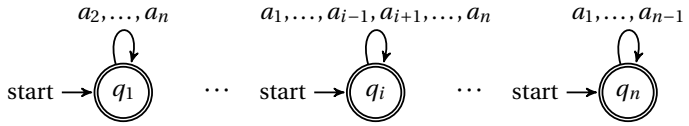


Figure 3.3: n-state NFA for $\Sigma_n^* \setminus U_n$.

3.2.2 State complexity of closures for languages over small alphabets

The language families $(L_n)_{n\in\mathbb{N}}$ and $(L'_n)_{n\in\mathbb{N}}$ used to prove that the upper bounds given in Propositions 3.3 and 3.5 are tight use linear-sized alphabets.

It is indeed known that the size of the alphabets matter for the state complexity of closure operations. In fact the automata witnessing tightness in Figs. 3.1 and 3.2 use the smallest possible alphabets. For example, Okhotin showed that the $2^{n-2}+1$ state complexity for $\uparrow L$ cannot be achieved with an alphabet of size smaller than $n-2$, see [94, Lemma 4.4].

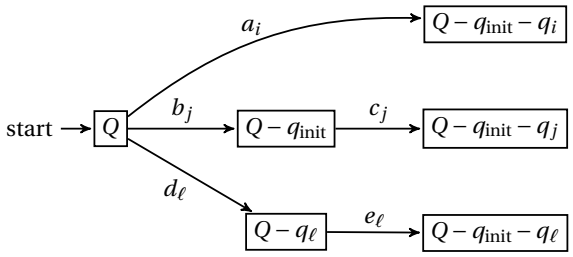
We now prove a similar result for downward closures:

Lemma 3.7. For $n>2$ let $A=(\Sigma,Q,\delta,\{q_{\text{init}}\},F)$ be an n-state NFA with a single initial state. If $|\Sigma|<n-1$ then $n_{\text{D}}(\downarrow L(A))<2^{n-1}$.

Proof. We assume that $n_{\text{D}}(\downarrow L(A))=2^{n-1}$ and deduce that $|\Sigma|\geq n-1$.

We write $Q=\{q_{\text{init}},q_1,\ldots,q_{n-1}\}$ to denote the states of A. As we saw with the proof of the first part of Proposition 3.5, the powerset automaton of $A^{\downarrow}$ can only have 2^{n-1} non-equivalent reachable states if all non-empty subsets of $Q\setminus\{q_{\text{init}}\}$ (written $Q-q_{\text{init}}$ for short) are reachable. Since $A^{\downarrow}$ has ϵ-transitions parallel to all transitions from A, every reachable state can be reached by ϵ transitions in $A^{\downarrow}$. Thus in the powerset automaton of $A^{\downarrow}$, we may take Q as the initial state. Then all edges $P\xrightarrow{a}P'$ in the powerset automaton satisfy $P\supseteq P'$. As a consequence, if $P\xrightarrow{x}P'$ for some $x\in\Sigma^*$ then in particular one can pick x with $|x|\leq|P\setminus P'|$.

Since every non-empty subset of $Q-q_{\text{init}}$ is reachable from Q there is in particular, for every $i=1,\ldots,n-1$, some x_i of length 1 or 2 such that $Q\xrightarrow{x_i}Q-q_{\text{init}}-q_i$. If we pick x_i of minimal



length then, for a given i, there are three possible cases (see picture): $x_i=a_i$ is a single letter (type 1), or x_i is some $b_i\,c_i$ with $Q\xrightarrow{b_i}Q-q_{\text{init}}\xrightarrow{c_i}Q-q_{\text{init}}-q_i$ (type 2), or x_i is some $d_i\,e_i$ with $Q\xrightarrow{d_i}Q-q_i\xrightarrow{e_i}Q-q_{\text{init}}-q_i$.

We now claim that the a_i's for type-1 states, the c_i's for type-2 states and the d_i's for type-3 states are all distinct, hence $|\Sigma|\geq n-1$.

Clearly the a_i's and the d_i's are pairwise distinct since they take Q to different states in the deterministic powerset automaton. Similarly, the c_i's are pairwise distinct, taking $Q-q_{\text{init}}$ to different states.

Assume now that $a_i = c_j$ for a type-1 q_i and a type-2 q_j. Then $Q - q_{\text{init}} \xrightarrow{c_j} Q - q_{\text{init}} - q_j$ and $Q \xrightarrow{a_i(=c_j)} Q - q_{\text{init}} - q_i$ contradict the monotonicity of $P \mapsto \delta(P,x)$ in the powerset automaton (since $Q - q_{\text{init}} - q_j$ and $Q - q_{\text{init}} - q_i$ are incomparable sets) . Similarly, assuming $d_\ell = c_j$ leads to $Q - q_{\text{init}} \xrightarrow{c_j} Q - q_{\text{init}} - q_j$ and $Q \xrightarrow{c_j(=d_\ell)} Q - q_\ell$, again contradicting monotonicity. Thus we can associate a distinct letter with each state $q_1, \ldots, q_{n-1}$, which concludes the proof. □

In view of the above results, the main question is whether, *in the case of a fixed alphabet*, exponential lower bounds still apply for the state complexity of upward and downward closures with DFAs as both input and output. The 1-letter case is degenerate since then both $n_{\text{D}}(\uparrow L)$ and $n_{\text{D}}(\downarrow L)$ are $\leq n_{\text{D}}(L)$. In the 3-letter case, exponential lower bounds for upward and downward closures were shown by Okhotin [94].

In the critical 2-letter case, say $\Sigma = \{a, b\}$, an exponential lower bound for upward closure was shown by Héam with the following witness: For $n > 0$, let $L''_n = \{a^i\, b\, a^{2j}\, b\, a^i \mid i + j + 1 = n\}$. Then $n_{\text{D}}(L''_n) = (n+1)^2$, while $n_{\text{D}}(\uparrow L''_n) \geq \frac{1}{7}(\frac{1+\sqrt{5}}{2})^n$ for $n \geq 4$ [57, Prop. 5.11]. Regarding downward closures for languages over a binary alphabet, the question was left open and we answer it in the next section.

3.3 Exponential state complexity of closures in the 2-letter case

In this section we show an exponential lower bound for the state complexity of downward closure in the case of a two-letter alphabet. Interestingly, the same languages can also serve as hard case for upward closure (but it gives weaker bounds than in [57]).

Theorem 3.8 (State complexity of closures with $|\Sigma| = 2$)**.** The state complexity of downward closure for languages over a binary alphabet is in $2^{\Omega(n^{1/3})}$. The same result holds for upward closure.

We now prove the theorem. Fix a positive integer n. Let

$$H = \{n, n+1, \ldots, 2n\},$$

and define morphisms $c, d : H^* \to \{a, b\}^*$ with, for any $i \in H$:

$$c(i) \stackrel{\text{def}}{=} a^i\, b^{3n-i}, \qquad\qquad d(i) \stackrel{\text{def}}{=} c(i)\, c(i).$$

Note that $c(i)$ always has length $3n$, begins with at least n a's, and ends with at least n b's. If we now let

$$L_n \stackrel{\text{def}}{=} \{c(i)^n \mid i \in H\},$$

L_n is a finite language of $n+1$ words, each of length $3n^2$ (and any two words in L_n share a common prefix of length not more than $2n$), so $n_{\text{D}}(L_n)$ is in $\Omega(n^3)$. (In fact, it can be shown

that $n_{\mathrm{D}}(L_n) = 3n^3 + 1$.) In the rest of this section we show that both $n_{\mathrm{D}}(\uparrow L_n)$ and $n_{\mathrm{D}}(\downarrow L_n)$ are in $2^{\Omega(n)}$.

Lemma 3.9. For $i, j \in H$, the longest prefix of $c(i)^\omega$ that embeds in $d(j) = c(j)\,c(j)$ is $c(i)$ if $i \neq j$ and $c(i)\,c(i)$ if $i = j$.

Proof sketch. The case $i = j$ is clear. Fig. 3.4 displays the leftmost embedding of $c(i)^\omega$ in $d(j)$ in a case where $i > j$. The remaining case, $i < j$, is similar. □

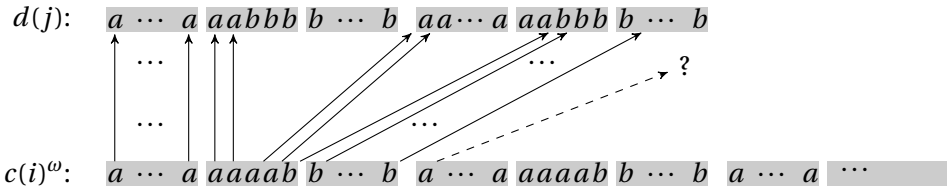


Figure 3.4: Case "$i > j$" in Lemma 3.9: here $i = n+4$ and $j = n+2$ for $n = 5$.

For each $i \in H$, let the morphisms $\eta_i, \theta_i : H^* \to (\mathbb{N}, +)$ be defined by

$$\eta_i(j) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i \neq j\,, \\ 2 & \text{if } i = j\,, \end{cases} \qquad \theta_i(j) \stackrel{\text{def}}{=} \begin{cases} 2 & \text{if } i \neq j\,, \\ 1 & \text{if } i = j\,. \end{cases}$$

Thus for $\sigma = p_1\,p_2 \cdots p_s \in H^*$, $\eta_i(\sigma)$ is s plus the number of occurrences of i in σ, while $\theta_i(\sigma)$ is $2s$ minus the number of these occurrences of i.

Lemma 3.10. Let $\sigma \in H^*$. The smallest ℓ such that $c(\sigma)$ embeds in $c(i)^\ell$ is $\theta_i(\sigma)$.

Proof. We write $\sigma = p_1\,p_2 \cdots p_s$ and prove the result by induction on s. The case where $s = 0$ is trivial. The case where $s = 1$ follows from Lemma 3.9, since for any p_1 and i, $c(p_1) \sqsubseteq c(i)$ iff $p_1 = i$, and $c(p_1) \sqsubseteq d(i) = c(i)^2$ always.

Assume now $s > 1$, write $\sigma = \sigma' p_s$ and let $\ell' = \theta_i(\sigma')$. By the induction hypothesis, $c(\sigma') \not\sqsubseteq c(i)^{\ell'-1}$ and $c(\sigma') \sqsubseteq c(i)^{\ell'} = c(i)^{\ell'-1} a^i b^{3n-i}$. Write now $c(i)^{\ell'} = w\,v$ where w is the shortest prefix of $c(i)^{\ell'}$ with $c(\sigma') \sqsubseteq w$. Since $c(\sigma')$ ends with some b that only embeds in the $a^i b^{3n-i}$ suffix of $c(i)^{\ell'}$, v is necessarily b^r for some r. So, for all $z \in \{a,b\}^*$, $c(p_s) \sqsubseteq v\,z$ if and only if $c(p_s) \sqsubseteq z$. We have $c(p_s) \sqsubseteq c(i)^{\theta_i(p_s)}$ and $c(p_s) \not\sqsubseteq v\,c(i)^{\theta_i(p_s)-1}$. Noting that $\sigma = \sigma' p_s$, we get $c(\sigma) \sqsubseteq c(i)^{\theta_i(\sigma)}$ and $c(\sigma) \not\sqsubseteq c(i)^{\theta_i(\sigma)-1}$. □

We now derive a lower bound on $n_{\mathrm{D}}(\downarrow L_n)$. For every subset X of H of size $n/2$ (assume n is even), let $w_X \in \{a,b\}^*$ be defined as follows: let the elements of X be $p_1 < p_2 < \cdots < p_{n/2}$ and let

$$w_X \stackrel{\text{def}}{=} c(p_1 p_2 \cdots p_{n/2})\,.$$

Note that $\theta_i(p_1 p_2 \cdots p_{n/2}) = n$ if $i \notin X$ and $\theta_i(p_1 p_2 \cdots p_{n/2}) = n-1$ if $i \in X$.

Lemma 3.11. Let X and Y be subsets of H of size $n/2$ with $X \neq Y$. There exists a word $v \in \{a,b\}^*$ such that $w_X v \in {\downarrow} L_n$ and $w_Y v \notin {\downarrow} L_n$.

Proof. Let $i \in X \setminus Y$. Let $v = c(i)$. Then

- By Lemma 3.10, $w_X \sqsubseteq c(i)^{n-1}$, and so $w_X v \sqsubseteq c(i)^n$. So $w_X v \in {\downarrow} L_n$.
- By Lemma 3.10, the smallest ℓ such that $w_Y v \sqsubseteq c(i)^\ell$ is $n+1$. Similarly, for $j \neq i$, the smallest ℓ such that $w_Y v \sqsubseteq c(j)^\ell$ is at least $n-1+2 = n+1$ (the w_Y contributes at least $n-1$ and the v contributes 2). So $w_Y v \notin {\downarrow} L_n$. □

This shows that for any DFA A recognizing ${\downarrow} L_n$, the state of A reached from the start state by every word in $\{w_X \mid X \subseteq H, |X| = n/2\}$ is distinct. Thus A has at least $\binom{n+1}{n/2}$ states, which is $\approx \frac{2^{n+3/2}}{\sqrt{\pi n}}$.

For $n_{\mathrm{D}}({\uparrow} L_n)$, the reasoning is similar:

Lemma 3.12. Let $\sigma \in H^*$. For all $i \in H$, the longest prefix of $c(i)^\omega$ that embeds in $d(\sigma)$ is $c(i)^{\eta_i(\sigma)}$.

Proof. By induction on the length of σ and applying Lemma 3.9. □

For every subset X of H of size $n/2$ (assume n is even), let $w'_X \in \{a,b\}^*$ be defined as follows: let the elements of X be $p_1 < p_2 < \cdots < p_{n/2}$ and let

$$w'_X \stackrel{\text{def}}{=} d(p_1 p_2 \cdots p_{n/2}) = c(p_1 p_1 p_2 p_2 \cdots p_{n/2} p_{n/2}).$$

Lemma 3.13. Let X and Y be subsets of H of size $n/2$ with $X \neq Y$. There exists a word $v \in \{a,b\}^*$ such that $w'_X v \in {\uparrow} L_n$ and $w'_Y v \notin {\uparrow} L_n$.

Proof. Let $i \in X \setminus Y$. Let $v = c(i)^{n-(n/2+1)} = c(i)^{n/2-1}$.

- By Lemma 3.12, $c(i)^{n/2+1} \sqsubseteq w'_X$, thus $c(i)^n \sqsubseteq w'_X v$, hence $w'_X v \in {\uparrow} L_n$.
- By Lemma 3.12, the longest prefix of $c(i)^n$ that embeds in $w'_Y v$ is at most $c(i)^\ell$ where $\ell = n/2 + n/2 - 1 = n-1$. The longest prefix of $c(j)^n$ that embeds in $w'_Y v$ for $j \neq i$ is at most $c(j)^\ell$ where
$$\ell = \frac{n}{2} + 1 + \left\lceil \frac{n/2-1}{2} \right\rceil \leq n-1.$$
Therefore $c(j)^n \not\sqsubseteq w'_Y v$ when $j = i$ and also when $j \neq i$. Thus $w'_Y v \notin {\uparrow} L_n$. □

With Lemma 3.13 we reason exactly as we did for $n_{\mathrm{D}}({\downarrow} L_n)$ after Lemma 3.11 and conclude that $n_{\mathrm{D}}({\uparrow} L_n) \geq \binom{n+1}{n/2}$ here too.

3.4 State complexity of interiors

Recall Eq. (3.1) that expresses interiors with closures and complements. Since complementation of DFAs does not increase the number of states, the state complexity of interiors, seen as DFA to DFA operations, is the same as the state complexity of closures (modulo swapping of up and down).

The remaining question is the *nondeterministic state complexity* of interiors, now seen as NFA to NFA operations. For this, Eq. (3.1) provides an obvious 2^{2^n} upper bound on the nondeterministic state complexity of both upward and downward interiors, simply by combining the powerset construction for complementation and the results of Section 3.2. Note that this procedure yields DFAs for the interiors while we are happy to accept NFAs if it improves the state complexity.

In the rest of this section, we prove that the nondeterministic state complexity of $\overset{\frown}{\downarrow} L$ and $\overset{\frown}{\uparrow} L$ are in $2^{2^{\Theta(n)}}$.

3.4.1 Upper bounds for interiors and the approximation problem

A generic argument lets us improve slightly on the 2^{2^n} upper bound:

Proposition 3.14. The (deterministic) state complexity of both the upward interior and the downward interior is $< \psi(n)$.

Here $\psi(n)$ is the Dedekind number that counts the number of antichains in the lattice of subsets of an n-element set, ordered by inclusion. Kahn [72, Coro. 1.4] shows

$$\binom{n}{\lfloor \frac{n}{2} \rfloor} \leq \log_2 \psi(n) \leq \left(1 + \frac{2\log(n+1)}{n}\right)\binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Note that the $\psi(n)$ upper bound is still in $2^{2^{\Theta(n)}}$, or "doubly exponential".

We prove Proposition 3.14 in a uniform way for both interiors. For this we adapt a technique already present in [33, Theo. 6.1]: the state complexity of a language that is a positive Boolean combination of left-quotients of some regular L is $\leq \psi(n_{\mathrm{N}}(L))$.

Let K_0 and $K_1, \ldots, K_p$ be arbitrary languages in Σ^* (these need not be regular). With the K_i's we associate an alphabet $\Gamma = \{b_1, \ldots, b_p\}$ and a substitution σ given inductively by $\sigma(\epsilon) \stackrel{\text{def}}{=} K_0$ and $\sigma(w\, b_i) \stackrel{\text{def}}{=} \sigma(w) \cdot K_i$. With a language $L \subseteq \Sigma^*$, we associate the language $W \subseteq \Gamma^*$ defined by

$$W \stackrel{\text{def}}{=} \{x \in \Gamma^* \mid \sigma(x) \subseteq L\}.$$

Remark 3.15. In the classical setting there is no K_0 and $\sigma(\epsilon) = \{\epsilon\}$, see [36, Chapter 6] and [90, Section 6]. There W is the best under-approximation of L by sums of products of K_i's and it is known that if L is regular then W is too. We allowed $\sigma(\epsilon) = K_0$ to account directly for upward interiors.

Proposition 3.16 (State complexity of approximations)**.** If L is regular then W is regular. Furthermore $n_{\mathrm{D}}(W) < \psi(n_{\mathrm{N}}(L))$.

Proof. Assume $A_1 = (\Sigma, Q, \delta_1, I_1, F_1)$ is an n-state NFA for L. We first start with a simple construction which gives a DFA for W with 2^{2^n} states, and then improve it to $\psi(n)$.

Using the powerset construction on A_1, one obtains an equivalent DFA $A_2 = (\Sigma, Q_2, \delta_2, i_2, F_2)$ for L. We have as usual $Q_2 = 2^Q$, with typical elements $S, S', \ldots$, $i_2 = I_1$, $F_2 = \{S \mid S \cap F_1 \neq \emptyset\}$, and δ_2 given by $\delta_2(S,a) = \bigcup_{q \in S} \delta_1(q,a)$.

We now use A_2 to get a DFA $A_3 = (\Gamma, Q_3, \delta_3, i_3, F_3)$ for W, where

- $Q_3 = 2^{Q_2}$, with typical elements $U, U', \ldots$;
- $\delta_3(U, b_j) = \{\delta_2(S,z) \mid S \in U, z \in K_j\}$;
- $i_3 = \{\delta_2(i_2, z) \mid z \in K_0\}$;
- $F_3 = 2^{F_2} = \{U \mid U \subseteq F_2\}$.

Claim 3.17. For all words $w \in \Gamma^*$, $\delta_3(i_3, w) = \{\delta_2(i_2, z) \mid z \in \sigma(w)\}$.

Proof. By induction on w. For the base case, one has $\delta_3(i_3, \epsilon) = i_3 = \{\delta_2(i_2, z) \mid z \in K_0\}$ by definition, and $\sigma(\epsilon) = K_0$. For the inductive case, one has

$$
\begin{aligned}
&\delta_3(i_3, w\, b_j) \\
&= \delta_3(\delta_3(i_3, w), b_j) \\
&= \delta_3(\{\delta_2(i_2, z) \mid z \in \sigma(w)\}, b_j) && \text{(ind. hypothesis)} \\
&= \left\{\delta_2(S, z') \mid S \in \{\delta_2(i_2, z) \mid z \in \sigma(w)\}, z' \in K_j\right\} && \text{(defn. of } \delta_3\text{)} \\
&= \{\delta_2(\delta_2(i_2, z), z') \mid z \in \sigma(w), z' \in \sigma(b_j)\} && \text{(rearrange, use } \sigma(b_j) = K_j\text{)} \\
&= \{\delta_2(i_2, z'') \mid z'' \in \sigma(w\, b_j)\}\,. && \square
\end{aligned}
$$

Corollary 3.18. The language accepted by A_3 is W.

Proof. For all $w \in \Gamma^*$, $w \in W$ iff $\sigma(w) \subseteq L$ iff $\forall z \in \sigma(w)[\delta_2(i_2, z) \in F_2]$ iff $\{\delta_2(i_2, z) \mid z \in \sigma(w)\} \subseteq F_2$ iff $\delta_3(i_3, w) \in F_3$ iff w is accepted by A_3. $\square$

So far, we have a DFA A_3 for W, with $|Q_3| = 2^{2^n}$ states. We now examine the construction more closely to detect equivalent states. Observe that the powerset construction for A_2 in terms of

A_1 is "existential", that is, a state of A_2 accepts if and only if at least one of its constituent states from A_1 accepts. In contrast, the powerset construction for A_3 in terms of A_2 is "universal", that is, a state of A_3 accepts if and only if all of its constituent states from A_2 accept. Suppose $B \subseteq C \in Q_2$ are two states of A_2. Then if some word is accepted by A_2 starting from B, it is also accepted starting from C. If a state of A_3 contains both B and C, then B already imposes a stronger constraint than C, and so C can be eliminated. We make this precise below:

Define an equivalence relation $\equiv$ on Q_3 as follows:

$$U \equiv U' \overset{\text{def}}{\Leftrightarrow} (\forall S \in U : \exists S' \in U' : S' \subseteq S) \wedge (\forall S' \in U' : \exists S \in U : S \subseteq S') .$$

A state $U \in Q_3$ is called an *antichain* if it does not contain some $S, S' \in Q_2$ with $S \subsetneq S'$. It is easy to see that every state $U \in Q_3$ is $\equiv$-equivalent to the antichain $U_{\min}$ obtained by retaining only the minimal-by-inclusion elements of U. Further, no two distinct antichain states can be $\equiv$-equivalent.

We now claim that, in A_3, $\equiv$-equivalent states accept the same language. First $U \equiv V$ and $U \in F_3$ imply $V \in F_3$ (proof: for any $S \in V$, there is a $S' \in U$ which is a subset, and since $S' \in F_2$, also $S \in F_2$). Furthermore, for each b_j, $\delta(U, b_j) \equiv \delta(V, b_j)$ (proof: an arbitrary element of $\delta_3(U, b_j)$ is $\delta_2(S, z)$ for some $S \in U$ and $z \in K_j$. There exists $S' \in V$ such that $S' \subseteq S$, and then $\delta_2(S', z)$ belongs to $\delta_3(V, b_j)$ and is a subset of $\delta_2(S, z)$ because δ_2 is monotone in its first argument. The reasoning in the reverse direction is similar).

Thus we can quotient the DFA A_3 by $\equiv$ to get an equivalent DFA for W. The number of states of $A_3 / \equiv$ is exactly the number of subsets of 2^Q which are antichains, and this is the Dedekind number $\psi(n)$. Further, we can remove (the equivalence class of) the sink state $\{\emptyset\}$. □

We instantiate the above for the upward and downward interiors to conclude that the nondeterministic state complexity of both is $< \psi(n)$: Choose alphabets $\Sigma = \Gamma = \{b_1, \ldots, b_k\}$. For the upward interior, define $K_0 = \Sigma^*$ and $K_i = \Sigma^* b_i \Sigma^*$, and apply Proposition 3.16. For the downward interior, define $K_0 = \{\epsilon\}$ and $K_i = \{b_i, \epsilon\}$, and apply Proposition 3.16. This completes the proof of Proposition 3.14.

3.4.2 Lower bound for downward interiors

We first establish a doubly-exponential lower bound for downward interiors:

Proposition 3.19. The nondeterministic state complexity of the downward interior is at least $2^{2^{\lfloor \frac{n-3}{2} \rfloor}}$.

Let ℓ be a positive integer, and let $\Sigma = \{0, 1, 2, \ldots, 2^\ell - 1\}$, so that $|\Sigma| = 2^\ell$. Let

$$L \overset{\text{def}}{=} \Sigma^* \setminus \{a\,a \mid a \in \Sigma\} = \{w \mid |w| \neq 2\} \cup \{a\,b \mid a, b \in \Sigma, a \neq b\} .$$

Then $\upharpoonright L$ consists of all words where every letter is distinct (equivalently, no letter appears more than once), a language called V_Σ in Lemma 3.2, showing $n_{\mathrm{N}}(\upharpoonright L) \geq 2^{|\Sigma|} = 2^{2^\ell}$.

Claim 3.20. $n_{\mathrm{N}}(L) \leq 2\ell + 3$.

Proof. Two letters in Σ, viewed as ℓ-bit sequences, are distinct if and only if they differ in at least one bit. An NFA can check this by guessing the position in which they differ and checking that the letters indeed differ in this position. Figure 3.5 shows an NFA for $\{a\,b \mid a \neq b\}$ with $2\ell + 2$ states.

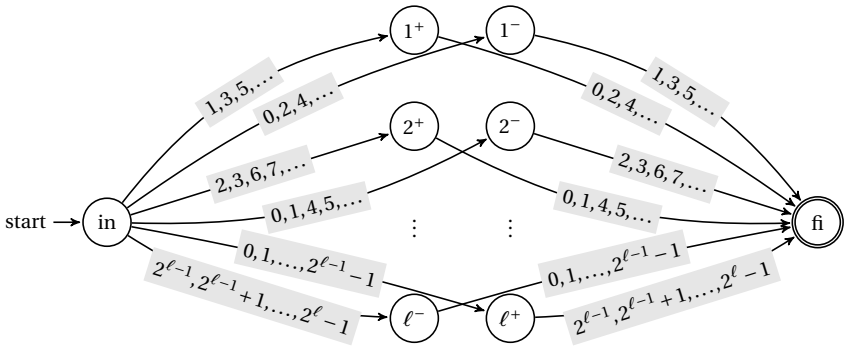


Figure 3.5: NFA for $\{a\,b \mid a, b \in \Sigma_{2^\ell}, a \neq b\}$ with $2\ell + 2$ states.

We need to modify this NFA to accept all words whose length is not 2. For this, we add a new state Z, and add the following transitions, on all letters: from each i^+ and i^- to Z, from Z to fi, and from fi to itself. We declare all states other than Z accepting. L is accepted by this resulting NFA, with $2\ell + 3$ states. □

Finally, combining $n_{\mathrm{N}}(L) \leq 2\ell + 3$ with the previously observed $n_{\mathrm{N}}(\upharpoonright L) \geq 2^{2^\ell}$ concludes the proof of Proposition 3.19.

3.4.3 Lower bound for upward interiors

We now establish the following doubly-exponential lower bound:

Proposition 3.21. The nondeterministic state complexity of the upward interior is $\geq 2^{2^{\lfloor (n-4)/3 \rfloor}} + 1$.

Our parameter is $\ell \in \mathbb{N}$ and we let $\Gamma \stackrel{\text{def}}{=} \{0, 1, \ldots, 2^\ell - 1\}$, $\Upsilon \stackrel{\text{def}}{=} \{1, \ldots, \ell\}$ and $\Sigma \stackrel{\text{def}}{=} \Gamma \cup \Upsilon$. The symbols in Γ, denoted $x, y, \ldots$ are disjoint from the symbols in Υ, denoted $k, k', \ldots$ (e.g., we can imagine that they have different colors) and one has $|\Sigma| = 2^\ell + \ell$.

For $x, y \in \Gamma$ and $k \in \Upsilon$, we write $x =_k y$ when x and y, viewed as ℓ-bit sequences, have the same kth bit. We consider the following languages:

$$L_1 \stackrel{\text{def}}{=} \{x\,w\,y\,k\,w' \in \Gamma \cdot \Sigma^* \cdot \Gamma \cdot \Upsilon \cdot \Sigma^* \mid x =_k y\},$$
$$L_2 \stackrel{\text{def}}{=} \Gamma \cdot (\Gamma \cdot \Upsilon)^*,$$
$$L \stackrel{\text{def}}{=} L_1 \cup (\Sigma^* \setminus L_2).$$

L_1 contains all words such that the initial letter $x \in \Gamma$ has one common bit with a later $y \in \Gamma$ and this bit is indicated by the $k \in \Upsilon$ that immediately follows the occurrence of y. Fig. 3.6 displays an NFA for L_1: it reads the first letter x, nondeterministically guesses k, and switches to a state r_k^+ or r_k^- depending on what is x's kth bit. From there it waits nondeterministically for the appearance of a factor $y\,k$ with $x =_k y$ before accepting. This uses $3\ell + 2$ states. Combining with an NFA for $\Sigma^* \setminus L_2$, we see that $n_{\mathrm{N}}(L) \leq 3\ell + 4$.

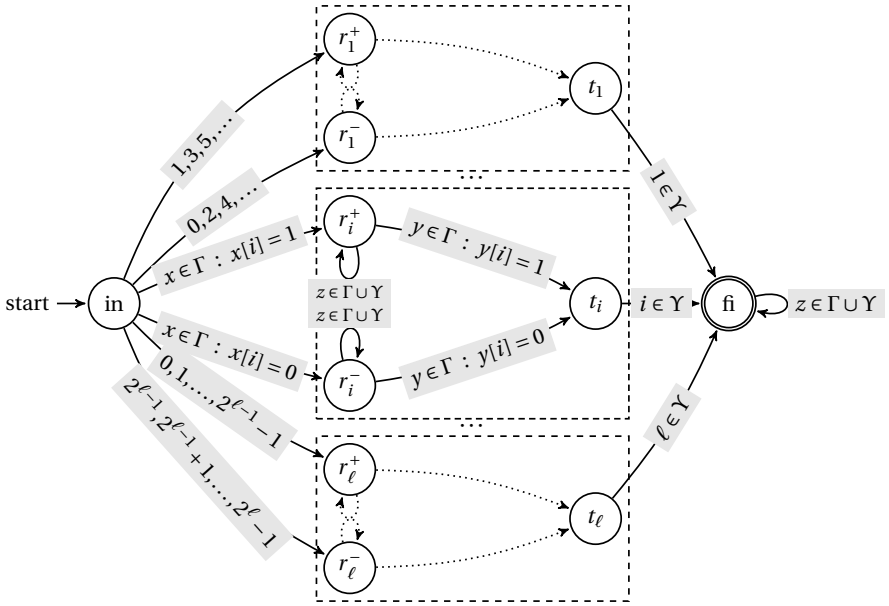


Figure 3.6: NFA for L_1 with $3\ell + 2$ states.

We consider the upward interior of L. As in Lemma 3.2, let $U_\Gamma \subseteq \Gamma^*$ be the language that contains all words over Γ where every letter appears at least once and $U'_\Gamma = \Gamma \cdot U_\Gamma$ be the language that has all words where the first suffix $w[2..]$ is in U_Γ.

Claim 3.22. $(\curvearrowleft L) \cap \Gamma^* = U'_\Gamma$.

Proof. We first show $(\curvearrowleft L) \cap \Gamma^* \subseteq U'_\Gamma$, by showing the contrapositive. Let $w \in \Gamma^* \setminus U'_\Gamma$. If $w = \epsilon$, then clearly $w \notin \curvearrowleft L$. Otherwise, $w = z\,z_1 \cdots z_p$, where $z, z_i \in \Gamma$. Since $z_1 \cdots z_p$ is not in U_Γ,

there is some $x \in \Gamma$ that differs from all the z_i's. Pick $k_1, \ldots, k_p$ witnessing this, i.e., such that $x \neq_{k_i} z_i$ for all i. If $x = z$ we let $w' \stackrel{\text{def}}{=} z\,z_1\,k_1 \cdots z_p\,k_p$ so that $w' \in L_2$ and $w' \notin L_1$, i.e., $w' \notin L$. If $x \neq z$ we let $w' \stackrel{\text{def}}{=} x\,z\,k\,z_1\,k_1 \cdots z_p\,k_p \notin L$ for some k witnessing $x \neq z$, so that $w' \notin L$. In both cases $w \sqsubseteq w' \notin L$ and we deduce $w \notin (\!\curvearrowleft\! L$.

We now show $U'_\Gamma \subseteq (\!\curvearrowleft\! L) \cap \Gamma^*$. Let $w = z\,z_1 \cdots z_p \in U'_\Gamma$. We show that $w \in (\!\curvearrowleft\! L$ by showing that $w' \in L$ for every w' such that $w \sqsubseteq w'$. If $w' \notin L_2$, then $w' \in L$. So assume $w' = x\,y_1\,k_1 \cdots y_n\,k_n \in L_2$. There is some i such that $x = z_i$ (since $w \in U'_\Gamma$) and some j such that $z_i = y_j$ (since $w \sqsubseteq w'$). We then have $x =_{k_j} y_j$ (this does not depend on the actual value of k_j). Hence $w' \in L_1 \subseteq L$. Thus $w \in (\!\curvearrowleft\! L$. □ □

Corollary 3.23. $n_{\mathrm{N}}(\!\curvearrowleft\! L) \geq 2^{2^\ell} + 1$.

Proof. From Lemma 3.2 we know that $n_{\mathrm{N}}(U'_\Gamma) = 2^{2^\ell} + 1$ and it is easily observed that $n_{\mathrm{N}}((\!\curvearrowleft\! L \cap \Gamma^*) \leq n_{\mathrm{N}}(\!\curvearrowleft\! L)$. □

Finally, combining $n_{\mathrm{N}}(\!\curvearrowleft\! L) \geq 2^{2^\ell} + 1$ with the previously observed $n_{\mathrm{N}}(L) \leq 3\ell + 4$ concludes the proof of Proposition 3.21.

3.4.4 On interiors of languages over a fixed alphabet

The doubly-exponential lower bounds exhibited in Props. 3.19 and 3.21 rely on alphabets of exponential size. It is an open question whether, in the case of a fixed alphabet, the nondeterministic state complexity of downward and/or upward interior is still doubly-exponential.

At some point we considered the language

$$P_n = \Sigma^* \setminus \{w\#w \mid |w| = n \text{ and } w \text{ has no } \#\}$$

over an alphabet of the form $\Sigma = \{\#\} \cup \{a_1, \ldots, a_k\}$ for some fixed k. The point is that P_n avoid words that are "squares" $w\#w$ of words w of length n, separated by the special symbol #. As a consequence, an arbitrary $u\#v$ in $\Sigma_k^*\#\Sigma_k^*$ is in $\curvearrowright\! P_n$ iff $\downarrow_{=n} u$ and $\downarrow_{=n} v$ are disjoint, where $\downarrow_{=n} x$ denotes the set $\Sigma^n \cap \downarrow x$ of subwords of x having length exactly n.

Let us write $u \sim'_n v$ when $\downarrow_{=n} u = \downarrow_{=n} v$ and $\mathscr{C}_k(n)$ for the number of $\sim_n$-equivalence classes in Σ_k^*: obviously $\mathscr{C}_k(n) \leq 2^{k^n}$.

Claim 3.24. $n_{\mathrm{N}}(P_n) \leq (k+1)n + k + 3$ and $n_{\mathrm{D}}(\curvearrowright\! P_n) \geq \mathscr{C}_k(n)$.

Proof sketch. One can recognize all words w with $w[i] \neq w[i+n+1]$ for some position i using a NFA with $k(n+1)+2$ states. With $n+1$ extra states, the NFA also recognizes the words that are not of the form $\Sigma_k^n\#\Sigma_k^n$.

For $n_{\mathrm{D}}(\curvearrowright\! P_n) \geq \mathscr{C}_k(n)$, we claim that if $\mathscr{C}(u) \neq \mathscr{C}(v)$ then $\delta(q_{\mathrm{init}}, u) \neq \delta(q_{\mathrm{init}}, v)$ in any DFA for $\curvearrowright\! P_n$. Indeed, pick some x in $\downarrow_{=n} u \setminus \downarrow_{=n} (v)$ (interchanging u and v if necessary) and note that $u\#x \notin \curvearrowright\! P_n \ni v\#x$. □

Thus if $\mathscr{C}_k(n)$ is doubly-exponential in n (for some k), P_n witnesses a doubly-exponential lower bound for downward interiors over a fixed alphabet, at least if we accept an NFA as input and DFA as output, which would still be an improvement over existing results.

Estimating $\mathscr{C}_k(n)$ was an open problem raised by Sakarovitch and Simon more than thirty years ago in [108, p. 110] and no doubly-exponential lower bound was known. However, chapter 4 shows that $\mathscr{C}_k(n)$ is in $2^{O(n^k)}$, hence not doubly exponential as hoped. As we will see, chapter 4 considers a slightly different equivalence relation $\sim_n$ by which two words are equivalent if they have the same subwords of length *at most* n. The only difference between the two is that as per $\sim_n$ all words of length upto $n-1$ are inequivalent, while as per $\sim'_n$ they are all equivalent. The two equivalences agree on words of length at least n.

At the moment we can only demonstrate a $2^{2^{\Omega(\sqrt{n})}}$ lower bound for the nondeterministic state complexity of *restricted interiors* over a 3-letter alphabet: this relies on a notion of "restricted" subwords where the alphabet is partitioned in two sets: letters than can be omitted (as usual) when building subwords, and letters that must be retained, see [79, Theo. 4.3] for details.

3.5 Complexity of decision problems on subwords

In automata-based procedures for logic and verification, the state complexity of automata constructions is not always the best measure of computational complexity. In this section we gather some elementary results on the complexity of subword-related decision problems on automata: deciding whether the languages they describe are downward (or upward) closed, and deciding whether they describe the same language modulo downward (or upward) closure. This is in the spirit of the work done in [73, 106] for closures by prefixes, suffixes, and factors. Some of the results we give are already known but they remain scattered in the literature.

3.5.1 Deciding closedness

Deciding whether $L(A)$ is upward-closed, or downward-closed, is unsurprisingly PSPACE-complete for NFAs, and NL-complete for DFAs. (For upward-closedness, this is already shown in [57], and quadratic-time algorithms that decide upward-closedness of $L(A)$ for a DFA A already appear in [10, 99].)

Proposition 3.25. Deciding whether $L(A)$ is upward-closed or downward-closed is PSPACE-complete when A is an NFA, even in the 2-letter alphabet case.

Proof sketch. A PSPACE algorithm simply tests for inclusion between two automata, A and $A^{\uparrow}$ (or $A^{\downarrow}$). PSPACE-hardness can be shown by adapting the proof for hardness of universality. Let R be a length-preserving semi-Thue system and x, x' two strings of same length. It is

PSPACE-hard to say whether $x \xrightarrow{*}_R x'$, even for a fixed R over a 2-letter alphabet Σ. We reduce (the negation of) this question to our problem.

Fix x and x' of length $n > 1$: a word $x_1 x_2 \cdots x_m$ of length $n \times m$ encodes a derivation if $x_1 = x$, $x_m = x'$, and $x_i \to_R x_{i+1}$ for all $i = 1, \ldots, m-1$. The language L of words that do *not* encode a derivation from x to x' is regular and recognized by an NFA with $O(n)$ states. Now, there is a derivation $x \xrightarrow{*}_R x'$ iff $L \neq \Sigma^*$. Since L contains all words of length not divisible by $n > 1$, it is upward-closed, or downward-closed, iff $L = \Sigma^*$, iff $\neg(x \xrightarrow{*}_R x')$. □

Proposition 3.26. Deciding whether $L(A)$ is upward-closed or downward-closed is NL-complete when A is a DFA, even in the 2-letter alphabet case.

Proof. Since L is downward-closed if, and only if, $\Sigma^* \setminus L$ is upward-closed, and since one easily builds a DFA for the complement of $L(A)$, it is sufficient to prove the result for upward-closedness.

We rely on the following easy lemma: L is upward-closed iff for all $u, v \in \Sigma^*$, $uv \in L$ implies $uav \in L$ for all $a \in \Sigma$. Therefore, $L(A)$ is not upward-closed —for $A = (\Sigma, Q, \delta, \{q_{\text{init}}\}, F)$— iff there are states $p, q \in Q$, a letter a, and words u, v such that $\delta(q_{\text{init}}, u) = p$, $\delta(p, a) = q$, $\delta(p, v) \in F$ and $\delta(q, v) \notin F$. If such words exist, in particular one can take u and v of length $< n = |Q|$ and respectively $< n^2$. Hence testing (the negation of) upward-closedness can be done in nondeterministic logarithmic space by guessing u, a, and v within the above length bounds, finding p and q by running u and then a from q_{init}, then running v from both p and q.

For hardness, one may reduce from vacuity of DFAs, a well-known NL-hard problem that is essentially equivalent to GAP, the Graph Accessibility Problem. Note that for any DFA A (in fact any NFA) the following holds:

$$L(A) = \emptyset \text{ iff } L(A) \cap \Sigma^{<n} = \emptyset \text{ iff } L(A) \cap \Sigma^{<n} \text{ is upward-closed,}$$

where n is the number of states of A. This provides the required reduction since, given a DFA A, one easily builds a DFA for $L(A) \cap \Sigma^{<n}$. □

3.5.2 Deciding equivalence modulo closure

The question whether $\downarrow L(A) = \downarrow L(B)$ or, similarly, whether $\uparrow L(A) = \uparrow L(B)$, is relevant in some settings where closures are used to build regular overapproximations of more complex languages.

Bachmeier *et al.* recently showed that the above two questions are coNP-complete when A and B are NFAs [14, Section 5], hence "easier" than deciding whether $L(A) = L(B)$. Here we give an improved version of their result.

Proposition 3.27 (after [14])**.** 1. Deciding whether $\downarrow L(A) \subseteq \downarrow L(B)$ or whether $\uparrow L(A) \subseteq \uparrow L(B)$ is coNP-complete when A and B are NFAs.

2. Deciding $\downarrow L(A) = \downarrow L(B)$ or $\uparrow L(A) = \uparrow L(B)$ is coNP-hard even when A and B are DFAs over a two-letter alphabet.

3. These problems are NL-complete when restricting to NFAs over a 1-letter alphabet.

Proof. 1. Let $B = (\Sigma, Q, \delta, I, F)$ and $n_B = |Q|$. Assume that $\downarrow L(A) \not\subseteq \downarrow L(B)$ and pick a shortest witness $x = x_1 \cdots x_\ell \in \Sigma^*$ with $x \in \downarrow L(A)$ and $x \notin \downarrow L(B)$. We claim that $|x| < n_B$: indeed in the powerset automaton obtained by determinizing $B^\downarrow$, the (unique) run $Q = S_0 \xrightarrow{x_1} S_1 \xrightarrow{x_2} \dots \xrightarrow{x_\ell} S_\ell$ of x is such that $S_0 \supseteq S_1 \supseteq S_2 \cdots \supseteq S_\ell$ (recall the proof of Lemma 3.7). If $S_{i-1} = S_i$ for some i, a shorter witness is obtained by omitting the ith letter in x (this does not affect membership in $\downarrow L(A)$ since this language is downward-closed). One concludes that the S_i have strictly diminishing size, hence $\ell < n_B$. This provides an NP algorithm deciding $\downarrow L(A) \not\subseteq \downarrow L(B)$: guess x in $\Sigma^{<n_B}$ and check in polynomial time that it is accepted by $A^\downarrow$ and not by $B^\downarrow$.

For upward closure the reasoning is even simpler and now a shortest witness has length $|x| < n_A$: if x is longer, we can find a subword x' that is still in $A^\uparrow$ (e.g., with pumping lemma), and this x' is not in $\uparrow L(B)$ since x is not.

2. coNP-hardness is shown by reduction from validity of DNF-formulae. Consider an arbitrary DNF formula $\phi = C_1 \vee C_2 \vee \dots \vee C_m$ made of m disjunctive clauses and using k Boolean variables $v_1, \dots, v_k$, e.g., $\phi = (v_1 \wedge \neg v_2 \wedge v_4) \vee (v_2 \wedge \cdots) \cdots$: it is easy to list all the valuations (seen as words in $\{0,1\}^k$) that make ϕ hold true. In order to recognize them with a DFA A_ϕ,

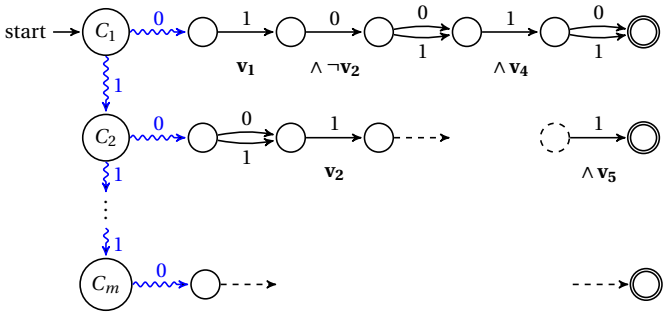


Figure 3.7: DFA A_ϕ for $\phi = (v_1 \wedge \neg v_2 \wedge v_4) \vee (v_2 \wedge \cdots \wedge v_5) \vee \cdots \vee C_m$ with $k = 5$ variables. The wavy and straight transitions and functionally identical.

we prefix each valuation by a string $1^\ell 0$ where ℓ witnesses the index of a clause $C_{\ell+1}$ made true by the valuation: see Figure 3.7 where the prefix $1^\ell 0$ uses wavy blue transitions while the valuation uses straight black letters (this distinction is for the reader only, the automaton only sees the letters 0 and 1).

Now A_ϕ has $m(k+2)$ states and accepts all words $1^\ell 0\, x_1 \cdots x_k$ in $\{0,1\}^*$ such that $x_1 \cdots x_k$ is

(the code for) a valuation that makes $C_{\ell+1}$ true. Let now B_ϕ be a DFA for $L(A_\phi) \cup 1^m 0(0+1)^k$, i.e., where all valuations are allowed after the $1^m 0$ prefix. It is clear that $\uparrow L(A_\phi) = \uparrow L(B_\phi)$ iff all words $1^m 0\, x_1 \cdots x_k$ appear in $\uparrow L(A)$ iff all valuations make ϕ true iff ϕ is valid, which completes the reduction for equality of upward closures.

For downward closures, we modify A_ϕ by adding a transition $C_m \xrightarrow{1} C_1$ so that now A_ϕ accepts all words $1^\ell 0\, x_1 \cdots x_k$ such that $x_1 \cdots x_k$ makes $C_{(\ell+1)\%m}$ true. For B we now take a DFA for $1^*0(0+1)^k$ and see that $\downarrow L(A_\phi) = \downarrow L(B)$ iff all valuations make ϕ true.

3. In the 1-letter case, comparing upward or downward closures amounts to comparing the length of the shortest (resp., longest) word accepted by the automata, which is easily done in nondeterministic logspace. And since $\uparrow L(A) = \downarrow L(A) = \emptyset$ iff $L(A) = \emptyset$, NL-hardness is shown by reduction from emptiness of NFAs, i.e., a question "is there a path from I to F" that is just another version of GAP, the Graph Accessibility Problem. □

A special case of language comparison is universality. The question whether $\uparrow L(A) = \Sigma^*$ is trivial since it amounts to asking whether ϵ is accepted by A. For downward closures one has the following:

Proposition 3.28 (after [106])**.** Deciding whether $\downarrow L(A) = \Sigma^*$ when A is a NFA over Σ is NL-complete.

Proof. Rampersad *et al.* show that the problem can be solved in linear time [106, Section 4.4]. Actually the characterization they use, namely $\downarrow L(A) = \Sigma^*$ iff $A = (\Sigma, Q, \delta, I, F)$ has a state $q \in Q$ with $I \xrightarrow{*} q \xrightarrow{*} F$ and such that for any $a \in \Sigma$ there is a path of the form $q \xrightarrow{*} \xrightarrow{a} \xrightarrow{*} q$ from q to itself, is a FO + TC sentence on A seen as a labeled graph, hence can be checked in NL [68]. NL-hardness can be shown by reduction from emptiness of NFAs, e.g., by adding loops $p \xrightarrow{a} p$ on any accepting state $p \in F$ and for every $a \in \Sigma$. □

3.6 Concluding remarks

For words ordered by the (scattered) subword relation, we considered the state complexity of computing closures and interiors, both upward and downward, of regular languages given by finite-state automata. These operations are essential when reasoning with subwords, e.g., in symbolic model checking for lossy channel systems, see [18, Section 6]. We completed the known results on closures by providing exact state complexities in the case of unbounded alphabets, and by demonstrating an exponential lower bound on downward closures even in the case of a two-letter alphabet.

The nondeterministic state complexity of interiors is a new problem that we introduced and for which we could show doubly-exponential upper and lower bounds.

These results contribute to a more general research agenda: what are the right data structures and algorithms for reasoning with subwords and superwords? The algorithmics of subwords and superwords has mainly been developed in string matching and combinatorics [15, 40] but other applications exist that require handling sets of strings rather than individual strings, e.g., model-checking and constraint solving [66]. When reasoning about sets of strings, there are many different ways of representing closed sets and automata-based representation are not always the preferred option, see, e.g., the SREs used for downward-closed languages in [4]. The existing trade-offs between all the available options are not yet well understood and certainly deserve scrutiny. In this direction, let us mention [19, Theo. 2.1(3)] showing that if $n_{\mathrm{D}}(L) = n$ then $\min(L) \stackrel{\text{def}}{=} \{x \in L \mid \forall y \in L : y \sqsubseteq x \implies y = x\} = L \setminus \mathtt{shuffle}(L, \Sigma)$ may have $n_{\mathrm{N}}(\min(L)) = (n-2)2^{n-3} + 2$, which suggests that it is more efficient to represent $\uparrow L$ directly than by its minimal elements.

The new interior operations open up several new avenues of exploration, such as:

- Deciding equivalence modulo interiors, analogous to deciding equivalence modulo closures from subsection 3.5.2.
- Interiors for other relations, such as prefix, suffix, factor, and the priority order from [53].
- The use of other automata models such as alternating automata and co-nondeterministic automata (motivated by the duality between interiors and closures).

Finally, a more ambitious extension would be to tree automata and tree embeddings, which by Kruskal's tree theorem is a well-quasi-order [93].

Chapter 4

Simon's congruence

Piecewise testable languages, introduced by Imre Simon in the 1970s, are a family of simple regular languages that are definable by the presence and absence of given subwords [116, 108, 98]. Formally, a language $L \subseteq A^*$ is n-piecewise testable if $x \in L$ and $x \sim_n y$ imply $y \in L$, where $x \sim_n y \overset{\text{def}}{\Leftrightarrow} x$ and y have the same subwords of length at most n. Piecewise testable languages are important (e.g. in learning theory or computational biology) because they are the languages defined by $\mathscr{B}\Sigma_1$ formulae, a simple fragment of first-order logic that is prominent in database queries.

It is easy to see that $\sim_n$ is a congruence with finite index and Sakarovitch and Simon raised the question of how to better characterize or evaluate this number [108, p. 110]. Let us write $C_k(n)$ for the number of $\sim_n$ classes over k letters, i.e., when $|A| = k$. It is clear that $C_k(n) \geq k^n$ since two words $x, y \in A^{\leq n}$ (i.e., of length at most n) are related by $\sim_n$ only if they are equal. In fact, this reasoning gives

$$C_k(n) \geq k^n + k^{n-1} + \cdots + k + 1 = \frac{k^{n+1} - 1}{k - 1} \tag{4.1}$$

(assuming $k \neq 1$). On the other hand, any congruence class in $\sim_n$ is completely characterized by a set of subwords in $A^{\leq n}$, hence

$$C_k(n) \leq 2^{\frac{k^{n+1}-1}{k-1}}. \tag{4.2}$$

Estimating the size of $C_k(n)$ has applications in descriptive complexity, for example for estimating the number of n-piecewise testable languages (over a given alphabet), or for bounding the size of canonical automata for n-piecewise testable languages [39, 84, 100].

Unfortunately the above bounds, summarized as $k^n \leq C_k(n) \leq 2^{k^{n+1}}$, leave a large ("exponential") gap and it is not clear towards which side is the actual value leaning.[1] Eq. (4.1) gives

[1] Comparing the bounds from Eqs. (4.1) and (4.2) with actual values does not bring much light here since the magnitude of $C_k(n)$ makes it hard to compute beyond some very small values of k and n, see Table 4.1.

a lower bound that is obviously very naive since it only counts the simplest classes. On the other hand, Eq. (4.2) too makes wide simplifications since not every subset of $A^{\leq n}$ corresponds to a congruence class. For example, if aa and bb are subwords of some x then necessarily x also has ab or ba among its length 2 subwords. In this chapter, log is with respect to base 2.

Since the question of estimating $C_k(n)$ was raised in [108] (and to the best of our knowledge) no progress had been made on the question, until Kátai-Urbán et al. proved the following bounds:

Theorem 4.1 ([81])**.** For all $k > 1$

$$\frac{k^n}{3^{n^2}} \log k \leq \log C_k(n) < 3^n k^n \log k \quad \text{if } n \text{ is even,}$$
$$\frac{k^n}{3^{n^2}} < \log C_k(n) < 3^n k^n \quad \text{if } n \text{ is odd.}$$

The proof is based on two reductions, one showing $C_{k+\ell}(n+2) \geq C_k^{\ell+2}(n)$ for proving lower bounds, and one showing $C_k(n+2) \leq (k+1)^{2k} C_k^{2k-1}(n)$ for proving upper bounds. For fixed n, Theorem 4.1 allows to estimate the asymptotic value of $\log C_k(n)$ as a function of k: it is in $\Theta(k^n)$ or $\Theta(k^n \log k)$ depending on the parity of n. However, these bounds do not say how, for fixed k, $C_k(n)$ grows as a function of n, which is a more natural question in settings where the alphabet is fixed, and where n comes from, e.g., the number of variables in a $\mathscr{B}\Sigma_1$ formula. In particular, the lower bound is useless for $n \geq k$ since in this case $k^n/3^{n^2} < 1$.

Our contribution In this chapter, we provide the following bounds:

Theorem 4.2. For all $k, n > 1$,

$$\left(\frac{n}{k}\right)^{k-1} \log\left(\frac{n}{k}\right) < \log C_k(n) < k\left(\frac{n+2k-3}{k-1}\right)^{k-1} \log n \log k\,.$$

Thus, for fixed k, $\log C_k(n)$ is in $\Theta(n^{k-1} \log n)$. Compared to Theorem 4.1 our bounds are much tighter for fixed k (and much wider for fixed n).

The proof of Theorem 4.2 uses two different reductions, in particular our upper bound is based on an original study of *minimal representatives* of congruence classes. The chapter is organized as follows. Section 4.1 recalls the necessary notations and definitions; the lower bound is proved in Section 4.2 while the upper bound is proved in Section 4.3. An appendix lists the exact values of $C_k(n)$ for small n and k that we managed to compute.

4.1 Preliminaries

We consider words $x, y, w, \ldots$ over a finite k-letter alphabet Σ_k (sometimes written simply Σ). For a word w and a letter a, we use $|w|_a$ to denote the number of occurrences of a in w.

For any $n \in \mathbb{N}$, we write $x \sim_n y$ when x and y have the same subwords of length $\leq n$. For example $x \stackrel{\text{def}}{=} abacb \sim_2 y \stackrel{\text{def}}{=} baaacbb$ since both words have $\{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, cb\}$ as subwords of length ≤ 2. However $x \not\sim_3 y$ since $x \sqsupseteq aba \not\sqsubseteq y$. Note that $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$, and that $x \sim_0 y$ holds trivially. It is well-known (and easy to see) that each $\sim_n$ is a congruence since the subwords of some xy are the concatenations of a subword of x and a subword of y. Simon defined a *piecewise testable* language as any $L \subseteq \Sigma^*$ that is closed by $\sim_n$ for some n [116]. These are exactly the languages definable by $\mathscr{B}\Sigma_1(<, a, b, \ldots)$ formulae, i.e., by Boolean combinations of Σ_1 first-order formulae with monadic predicates of the form $a(i)$, stating that the i-th letter is a. For example, $L = \Sigma^* a \Sigma^* b \Sigma^* = \{x \in \Sigma^* : ab \sqsubseteq x\}$ is definable with the following Σ_1 formula:

$$\exists i : \exists j : i < j \wedge a(i) \wedge b(j)\,.$$

The index of $\sim_n$ Since there are only finitely many words of length $\leq n$, the congruence $\sim_n$ partitions Σ_k^* in finitely many classes, and we write $C_k(n)$ for the number of such classes, i.e., the cardinal of $\Sigma_k^* / \sim_n$.

The following is easy to see

$$C_1(n) = n + 1\,, \qquad C_k(0) = 1\,, \qquad C_k(1) = 2^k\,. \tag{4.3}$$

Indeed, for words over a single letter a, $x \sim_n y$ iff $|x| = |y| < n$ or $|x| \geq n \leq |y|$, hence the first equality. The second equality restates that $\sim_0$ is trivial, as noted above. For the third equality, one notes that $x \sim_1 y$ if, and only if, the same set of letters is occurring in x and y, and that there are 2^k such sets of occurring letters.

4.2 Lower bound

The first half of Theorem 4.2 is proved by first establishing a combinatorial inequality on the $C_k(n)$'s (Proposition 4.5) and then using it to derive Proposition 4.6.

Consider two words $x, y \in \Sigma^*$ and a letter $a \in \Sigma$.

Lemma 4.3. If $x \sim_n y$, then $\min(|x|_a, n) = \min(|y|_a, n)$.

Proof sketch. If $|x|_a = p < n$ then $a^p \sqsubseteq x \not\sqsubseteq a^{p+1}$. From $x \sim_n y$ we deduce $a^p \sqsubseteq y \not\sqsubseteq a^{p+1}$, hence $|y|_a = p$. □

Fix now $k \geq 2$, let $A = A_k = \{a_1, \ldots, a_k\}$ and assume $x \sim_n y$. If $|x|_{a_k} = p < n$, then x is some $x_0 a_k x_1 \cdots a_k x_p$ with $x_i \in A_{k-1}^*$ for $i = 0, \ldots, p$. By Lemma 4.3, y too is some $y_0 a_k y_1 \cdots a_k y_p$ with $y_i \in A_{k-1}^*$.

Lemma 4.4. $x_i \sim_{n-p} y_i$ for all $i = 0, \ldots, p$.

Proof. Suppose $w \sqsubseteq x_i$ and $|w| \leq n-p$. Let $w' \stackrel{\text{def}}{=} a_k^i w a_k^{p-i}$. Clearly $w' \sqsubseteq x$ and thus $w' \sqsubseteq y$ since $x \sim_n y$ and $|w'| \leq n$. Now $w' = a_k^i w a_k^{p-i} \sqsubseteq y$ entails $w \sqsubseteq y_i$.
With a symmetric reasoning we show that every subword of y_i having length $\leq n-p$ is a subword of x_i and we conclude $x_i \sim_{n-p} y_i$. □

Proposition 4.5. For $k \geq 2$, $C_k(n) \geq \sum_{p=0}^{n} C_{k-1}^{p+1}(n-p)$.

Proof. For words $x = x_0 a_k x_1 \ldots x_{p-1} a_k x_p$ which have exactly $p < n$ occurrences of a_k, we have $C_{k-1}(n-p)$ possible choices of $\sim_{n-p}$ equivalence classes for each x_i $(i = 0, \ldots, p)$. By Lemma 4.4 all such choices will result in $\not\sim_n$ words, hence there are exactly $C_{k-1}^{p+1}(n-p)$ classes of words with $p < n$ occurrences of a_k. By Lemma 4.3, these classes are disjoint for different values of p, hence we can add the $C_{k-1}^{p+1}(n-p)$'s. There remain words with $p \geq n$ occurrences of a_k, accounting for at least 1, i.e., $C_{k-1}^{n+1}(0)$, additional class. □

Proposition 4.6. For all $k, n > 0$:

$$\log C_k(n) > \left(\frac{n}{k}\right)^{k-1} \log\left(\frac{n}{k}\right). \tag{4.4}$$

Proof. Eq. (4.4) holds trivially when $\log(\frac{n}{k}) \leq 0$. Hence there only remains to consider the cases where $n > k$. We reason by induction on k. For $k = 1$, Eq. (4.3) gives $\log C_1(n) = \log(n+1) > \log n = \left(\frac{n}{1}\right)^0 \log\left(\frac{n}{1}\right)$. For the inductive case, Proposition 4.5 yields $C_{k+1}(n) \geq C_k^{p+1}(n-p)$ for all $p \in \{0, \ldots, n\}$. For $p = \left\lfloor \frac{n}{k+1} \right\rfloor$ this yields

$$\begin{aligned} \log C_{k+1}(n) &\geq (p+1) \log C_k(n-p) \\ &> (p+1)\left(\frac{n-p}{k}\right)^{k-1} \log\left(\frac{n-p}{k}\right) \end{aligned}$$

by ind. hyp., noting that $n - p > 0$,

$$\geq \frac{n}{k+1}\left(\frac{n}{k+1}\right)^{k-1} \log\left(\frac{n}{k+1}\right)$$

since $\frac{n-p}{k} \geq \frac{n}{k+1} \geq 1$,

$$= \left(\frac{n}{k+1}\right)^{k} \log\left(\frac{n}{k+1}\right)$$

as desired. □

4.3 Upper bound

The second half of Theorem 4.2 is again by establishing a combinatorial inequality on the $C_k(n)$'s (Proposition 4.9) and then using it to derive Proposition 4.10.

Fix $k > 0$ and consider words in Σ_k^*. We say that a word x is *rich* if all the k letters of Σ_k occur in it, and that it is *poor* otherwise. For $\ell > 0$, we further say that x is ℓ-rich if it can be written as a concatenation of ℓ rich factors. We define x to be 0-rich if x is poor. The *richness* of x is the largest $\ell \in \mathbb{N}$ such that x is ℓ-rich. Note that $\forall a \in \Sigma_k : |x|_a \geq \ell$ does not imply that x is ℓ-rich. We shall use the following easy result:

Lemma 4.7. If x_1 and x_2 are respectively ℓ_1-rich and ℓ_2-rich, then $y \sim_n y'$ implies $x_1 y x_2 \sim_{\ell_1+n+\ell_2} x_1 y' x_2$.

Proof. A subword u of $x_1 y x_2$ can be decomposed as $u = u_1 v u_2$ where u_1 is the largest prefix of u that is a subword of x and u_2 is the largest suffix of the remaining $u_1^{-1} u$ that is a subword of x_2. Thus $v \sqsubseteq y$ since $u \sqsubseteq x_1 y x_2$. Now, since x_1 is ℓ_1-rich, $|u_1| \geq \ell_1$ (unless u is too short), and similarly $|u_2| \geq \ell_2$ (unless $u_1^{-1} u$ is too short). Finally $|v| \leq n$ when $|u| \leq \ell_1 + n + \ell_2$, and then $v \sqsubseteq y'$ since $y \sim_n y'$, entailing $u \sqsubseteq x_1 y' x_2$. A symmetrical reasoning shows that subwords of $x_1 y' x_2$ of length $\leq \ell_1 + n + \ell_2$ are subwords of $x_1 y x_2$ and we are done. □

The *rich factorization* of $x \in \Sigma_k^*$ is the decomposition $x = x_1 a_1 \cdots x_m a_m y$ obtained in the following way: if x is poor, we let $m = 0$ and $y = x$; otherwise x is rich, we let $x_1 a_1$ (with $a_1 \in \Sigma_k$) be the shortest prefix of x that is rich, write $x = x_1 a_1 x'$ and let $x_2 a_2 \dots x_m a_m y$ be the rich factorization of the remaining suffix x'. By construction m is the richness of x. E.g., assuming $k = 3$, the following is a rich factorization with $m = 2$:

$$\overbrace{bbaaabbccccaabbbaa}^{x} = \overbrace{bbaaabb}^{x_1} \cdot c \cdot \overbrace{cccaa}^{x_2} \cdot b \cdot \overbrace{bbaa}^{y}$$

Note that, by definition, $x_1, \dots, x_m$ and y are poor.

Lemma 4.8. Consider two words x, x' of richness m with rich factorizations

$$x = x_1 a_1 \dots x_m a_m y \qquad x' = x'_1 a_1 \dots x'_m a_m y'.$$

Suppose that $y \sim_n y'$ and that $x_i \sim_{n+1} x'_i$ for all $i = 1, \dots, m$. Then $x \sim_{n+m} x'$.

Proof. By repeatedly using Lemma 4.7, one shows

$$\begin{aligned} x_1 a_1 x_2 a_2 \dots x_m a_m y &\sim_{n+m} x'_1 a_1 x_2 a_2 \dots x_m a_m y \\ &\sim_{n+m} x'_1 a_1 x'_2 a_2 \dots x_m a_m y \\ &\vdots \\ &\sim_{n+m} x'_1 a_1 x'_2 a_2 \dots x'_m a_m y \\ &\sim_{n+m} x'_1 a_1 x'_2 a_2 \dots x'_m a_m y', \end{aligned}$$

using the fact that each factor $x_i a_i$ is rich. □

Proposition 4.9. For all $n \geq 0$ and $k \geq 2$,

$$C_k(n) \leq 1 + \sum_{m=0}^{n-1} k^{m+1} C_{k-1}^m (n-m+1)\, C_{k-1}(n-m)\,.$$

Furthermore, for $k = 2$,

$$C_2(n) \leq 2 \sum_{m=0}^{2n-1} n^m = 2\frac{n^{2n}-1}{n-1}\,. \tag{4.5}$$

Proof. Consider two words x, x' and their rich factorizations

$$x = x_1 a_1 \dots x_m a_m y \qquad x' = x'_1 a'_1 \dots x'_\ell a'_\ell y'.$$

By Lemma 4.8 they belong to the same $\sim_n$ class if $\ell = m$, $y \sim_{n-m} y'$, and $a_i = a'_i$ and $x_i \sim_{n-m+1} x'_i$ for all $i = 1, \dots, m$. Now for every fixed m, there are at most k^m choices for the a_i's, $C_{k-1}^m(n-m+1)$ non-equivalent choices for the x_i's, $kC_{k-1}(n-m)$ choices for y and a letter that is missing in it. We only need to consider m varying up to $n-1$ since all words of richness $\geq n$ are $\sim_n$-equivalent, accounting for one additional possible $\sim_n$ class.

For the second inequality, assume that $k = 2$ and $\Sigma_2 = \{a, b\}$. A word $x \in \Sigma_2^*$ can be decomposed as a sequence of m non-empty blocks of the same letter, of the form, e.g., $x = a^{\ell_1} b^{\ell_2} a^{\ell_3} b^{\ell_4} \cdots a^{\ell_m}$ (this example assumes that x starts and ends with a, hence m is odd). If two words like $x = a^{\ell_1} b^{\ell_2} a^{\ell_3} b^{\ell_4} \cdots a^{\ell_m}$ and $x' = a^{\ell'_1} b^{\ell'_2} a^{\ell'_3} b^{\ell'_4} \cdots a^{\ell'_m}$ have the same first letter a, the same alternation depth m, and have $\min(\ell_i, n) = \min(\ell'_i, n)$ for all $i = 1, \dots, m$, then they are $\sim_n$-equivalent. For a given $m > 0$, there are 2 possibilities for choosing the first letter and n^m non-equivalent choices for the ℓ_i's. Finally, all words with alternation depths $m \geq 2n$ are $\sim_n$-equivalent, hence we can restrict our attention to $1 \leq m \leq 2n-1$. The extra summand $2\,n^0$ in Eq. (4.5) accounts for the single class with $m \geq 2n$ and the single class with $m = 0$. □

In the following we sometimes use $\exp(x)$ for 2^x.

Proposition 4.10. For all $k, n > 1$:

$$C_k(n) < 2^{k\left(\frac{n+2k-3}{k-1}\right)^{k-1} \log n \log k}.$$

Proof. By induction on k. For $k = 2$, Eq. (4.5) yields:

$$C_2(n) \leq 2\frac{n^{2n}-1}{n-1} < n\frac{n^{2n+1}}{1}$$

since $n \geq 2$,

$$\begin{aligned} &= n^{2n+2} = 2^{2(n+1)\log n} \\ &= \exp\left(k\left(\frac{n+2k-3}{k-1}\right)^{k-1} \log n \log k\right). \end{aligned}$$

For the inductive case, Proposition 4.9 yields:

$$\begin{aligned}
C_{k+1}(n) &\le 1 + \sum_{m=0}^{n-1} (k+1)^{m+1} C_k^m (n-m+1) C_k(n-m) \\
&= 1 + (k+1) C_k(n) \\
&\quad + \sum_{m=1}^{n-1} (k+1)^{m+1} C_k^m (n-m+1) C_k(n-m) \\
&< (k+1)^n C_k(n) + \sum_{m=1}^{n-1} (k+1)^n C_k^{m+1} (n-m+1)
\end{aligned}$$

since $C_k(q) \le C_k(q+1)$,

$$\begin{aligned}
&< (k+1)^n 2^{k\left(\frac{n+2k-3}{k-1}\right)^{k-1} \log n \log k} \\
&\quad + \sum_{m=1}^{n-1} (k+1)^n 2^{k(m+1)\left(\frac{n-m+2k-2}{k-1}\right)^{k-1} \log n \log k}
\end{aligned}$$

by ind. hyp.,

$$< (k+1)^n \sum_{m=0}^{n-1} 2^{k(m+1)\left(\frac{n-m+2k-2}{k-1}\right)^{k-1} \log n \log k}.$$

Since $(m+1)\left(\frac{n-m+2k-2}{k-1}\right)^{k-1} \le \left(\frac{n+2k-1}{k}\right)^k$ for all $m \in \{0,\dots,n-1\}$ (see Lemma 4.11), we may proceed with:

$$\begin{aligned}
C_{k+1}(n) &< (k+1)^n \sum_{m=0}^{n-1} 2^{k\left(\frac{n+2k-1}{k}\right)^k \log n \log k} \\
&= n(k+1)^n 2^{k\left(\frac{n+2k-1}{k}\right)^k \log n \log k} \\
&= \exp\left(\log n + n\log(k+1) + k\left(\frac{n+2k-1}{k}\right)^k \log n \log k\right) \\
&< \exp\left(\left(\log n + n + k\left(\frac{n+2k-1}{k}\right)^k \log n\right)\log(k+1)\right) \\
&< \exp\left((k+1)\left(\frac{n+2k-1}{k}\right)^k \log n \log(k+1)\right)
\end{aligned}$$

since $\log n + n < \left(\frac{n+2k-1}{k}\right)^k \log n$ (see below). This is the desired bound.

To see that $\log n + n < \left(\frac{n+2k-1}{k}\right)^k \log n$, we use

$$\begin{aligned}
\left(\frac{n+2k-1}{k}\right)^k > \left(\frac{n}{k}+1\right)^k &= \sum_{j=0}^{k} \binom{k}{j} \cdot \left(\frac{n}{k}\right)^j \\
&= 1 + k \cdot \left(\frac{n}{k}\right) + \cdots \ge n+1\,.
\end{aligned}$$

This completes the proof. □

The following technical lemma used above remains to be proven:

Lemma 4.11. $(m+1)\left(\frac{n-m+2k-2}{k-1}\right)^{k-1} \leq \left(\frac{n+2k-1}{k}\right)^{k}$ for all $m = 0, \ldots, n-1$.

Proof. For $k > 0$ and $x, y \in \mathbb{R}$, let

$$F_k(x) \stackrel{\text{def}}{=} \left(\frac{x+2k-1}{k}\right)^k,$$
$$G_{k,x}(y) \stackrel{\text{def}}{=} (y+1)F_k(x-y+1) = \frac{(y+1)(x-y+2k)^k}{k^k}.$$

Let us check that $G_{k,x}\left(\frac{k+x}{k+1}\right) = F_{k+1}(x)$ for any $k > 0$ and $x \geq 0$:

$$\begin{aligned} G_{k,x}\left(\frac{k+x}{k+1}\right) &= \left(\frac{k+x}{k+1}+1\right)\frac{1}{k^k}\left(x - \frac{k+x}{k+1} + 2k\right)^k \\ &= \frac{x+2k+1}{k+1}\,\frac{1}{k^k}\left(\frac{kx+2k^2+k}{k+1}\right)^k \\ &= \frac{x+2k+1}{k+1}\,\frac{1}{k^k}\left(\frac{k}{k+1}\right)^k (x+2k+1)^k \\ &= \left(\frac{x+2k+1}{k+1}\right)^{k+1} = F_{k+1}(x). \end{aligned} \tag{†}$$

We now claim that $G_{k,x}(y) \leq F_{k+1}(x)$ for all $y \in [0, x]$. For $n, k \geq 2$, the claim entails $G_{k-1,n}(m) \leq F_k(n)$, i.e. $(m+1)\left(\frac{n-m+2k-2}{k-1}\right)^{k-1} \leq \left(\frac{n+2k-1}{k}\right)^k$, for $m = 0, \ldots, n-1$ as announced.

We now prove this claim. Let $y_{\max} \stackrel{\text{def}}{=} \frac{k+x}{k+1}$. We prove that $G_{k,x}(y) \leq G_{k,x}(y_{\max})$ and conclude using Eq. (†): $G_{k,x}$ is well-defined and differentiable over $\mathbb{R}$, its derivative is

$$\begin{aligned} G'_{k,x}(y) &= \frac{(x-y+2k)^k - (y+1)k(x-y+2k)^{k-1}}{k^k} \\ &= \frac{(x-y+2k)^{k-1}}{k^k}\big((x-y+2k) - (y+1)k\big) \\ &= \frac{(x-y+2k)^{k-1}}{k^k}\big(x+k-y(k+1)\big). \end{aligned}$$

Thus $G'_{k,x}(y)$ is 0 for $y = y_{\max}$, is strictly positive for $0 \leq y < y_{\max}$, and strictly negative for $y_{\max} < y \leq x$. Hence, over $[0, x]$, $G_{k,x}$ reaches its maximum at $y_{\max}$. □

By combining the two bounds in Propositions 4.6 and 4.10 we obtain Theorem 4.2, implying that $\log C_k(n)$ is in $\Theta(n^{k-1}\log n)$ for fixed alphabet size k.

4.4 Concluding remarks

We proved that, over a fixed k-letter alphabet, $C_k(n)$ is in $2^{\Theta(n^{k-1}\log n)}$. This shows that $C_k(n)$ is not doubly exponential in n as Eq. (4.2) and Theorem 4.1 would allow. It also is not simply exponential, bounded by a term of the form $2^{f(k)\cdot n^c}$ where the exponent c does not depend on k.

We are still far from having a precise understanding of how $C_k(n)$ behaves and there are obvious directions for improving Theorem 4.2. For example, its bounds are not monotonic in k (while the bounds in Theorem 4.1 are not monotonic in n) and it only partially uses the combinatorial inequalities given by Propositions 4.5 and 4.9.

4.5 Appendix: first values for $C_k(n)$

We computed the first values of $C_k(n)$ by a brute-force method. The cells left blank in the tables below were not computed for lack of memory.

Consider $\Sigma^*_{\geq n}$, the set of all words over Σ of length at least n. On $\Sigma^*_{\geq n}$, define the equivalence relation $x \sim'_n y$ iff x and y have the same subwords of length *exactly* n. Observe that for x and y in $\Sigma^*_{\geq n}$, $x \sim_n y$ if and only if $x \sim'_n y$. We have, with $|\Sigma| = k$,

$$\left|\Sigma^* / \sim_n\right| = \left|\Sigma^*_{\geq n} / \sim'_n\right| + k^{n-1} + k^{n-2} + \ldots + k + 1$$

and thus computing the index of $\sim_n$ reduces to computing the index of $\sim'_n$. Let $\mathscr{S}_n(w)$ denote the set of all n-length subwords of w. For a word w of length at least n and a letter a, observe that

$$\mathscr{S}_n(wa) = \bigcup_{v \in \mathscr{S}_n(w)} \mathscr{S}_n(va)$$

Thus to compute $\mathscr{S}_n$ of arbitrary words of length at least n, it suffices to compute it for words of length n and $n+1$, and take unions. For a word w of length n and a letter a, define $\texttt{next}(w, a)$ to be $\mathscr{S}_n(wa)$.

To compute the index of $\sim'_n$, the algorithm works as follows:

1. Fix an enumeration of $\Sigma^*_{=n}$, the set of all words of length n. The order doesn't matter, but it is important to fix it. A subset of $\Sigma^*_{=n}$ is now represented as a bit vector of length k^n.

2. Precompute `next`.

3. Define a *tagged word* to be a pair $(w, \mathscr{S}_n(w))$, for some $w \in \Sigma^*_{\geq n}$. Initalize a queue with all tagged word corresponding to words of length n.

4. As long as the queue is nonempty, repeatedly do the following: extract a tagged word (z, A) from the queue. For every letter $c \in \Sigma$,

Compute $B_c \stackrel{\text{def}}{=} \bigcup_{t \in A} \texttt{next}(t, c)$. If B_c has not been seen before as the second component of a tagged word, mark it as seen and push (zc, B_c) onto the queue.

Note that this needs maintinaing a global set of seen sets.

Finally, $C_k(n)$ is simply the number of elements pushed onto the set. This algorithm is essentially a breadth-first exploration of a suitable graph.

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	k
$n=0$	1	1	1	1	1	1	1	1
$n=1$	2	4	8	16	32	64	128	2^k
$n=2$	3	16	152	2326	52132	1602420	64529264	

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	k
$n=3$	4	68	5312	1395588	1031153002	
$n=4$	5	312	334202			
$n=5$	6	1560	38450477			
$n=6$	7	8528				
$n=7$	8	50864				
$n=8$	9	329248				
$n=9$	10	2298592				
$n=10$	11	17203264				
$n=11$	12	137289920				
n	$n+1$					

Table 4.1: Computed values of $C_k(n)$

Chapter 5

Post Embedding Problem

The *Regular Post Embedding Problem* (PEP for short, named by analogy with Post's Correspondence Problem, aka PCP) is the problem of deciding, given two morphisms on words $u, v : \Sigma^* \to \Gamma^*$ and a regular language $R \in \mathsf{Reg}(\Sigma)$, whether there is $\sigma \in R$ such that $u(\sigma)$ is a (scattered) subword of $v(\sigma)$. One then calls σ a *solution* of the PEP instance. We will drop the "Regular" and use "Post Embedding Problem" to refer to this problem.

Equivalently, PEP is the question whether a rational relation, or a transduction, $T \subseteq \Gamma^* \times \Gamma^*$ intersects non-vacuously the subword relation [16], hence it is a special case of the intersection problem for two rational relations.

This problem, introduced in [27], is quite remarkable: it is decidable but surprisingly hard since it is not primitive-recursive.[1] The problem is in fact $\mathsf{F}_{\omega^\omega}$-complete [76], that is, it sits at the first level above multiply-recursive in the Ordinal-Recursive Complexity Hierarchy [111].

A variant problem was introduced in [27]: $\mathsf{PEP}_{\mathrm{dir}}$ asks for the existence of a *direct* solution, i.e., some $\sigma \in R$ such that $u(\tau) \sqsubseteq v(\tau)$ for every prefix τ of σ. It turns out that PEP and $\mathsf{PEP}_{\mathrm{dir}}$ are inter-reducible (though not trivially) [29] and have the same complexity.

In this chapter we introduce $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{dir}}$, or "PEP with *partial* directness": Instead of requiring $u(\tau) \sqsubseteq v(\tau)$ for all prefixes of a solution (as in $\mathsf{PEP}_{\mathrm{dir}}$), or for none (as in PEP), $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{dir}}$ lets us select, by means of a regular language, which prefixes should verify the requirement. Thus $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{dir}}$ generalizes both PEP and $\mathsf{PEP}_{\mathrm{dir}}$.

Our main result is that $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{dir}}$ and the mirror problem $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ are decidable. The proof combines two ideas. Firstly, by Higman's Lemma (Theorem 2.20), a long solution must eventually contain "*comparable*" so-called cutting points, from which one deduces that the solution is not minimal (or unique, or ...). Secondly, the above notion of "*eventually*", that comes from Higman's Lemma, can be turned into an effective upper bound thanks to a Length Func-

[1] But the problem becomes easy, decidable in linear-time and logarithmic space [27], when restricted to $R = \Sigma^+$ as in PCP.

tion Theorem [112].

The decidability of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ not only generalizes the decidability of PEP and $\mathsf{PEP}_{\text{dir}}$: it is also simpler than the earlier proofs for PEP or $\mathsf{PEP}_{\text{dir}}$, and it easily leads to an $\mathsf{F}_{\omega^\omega}$ complexity upper bound.

In a second part of the chapter, we extend our main result and show the decidability of universal and/or counting versions of the extended $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ problem. We also explain how our attempts at further generalisation, most notably by considering the combination of directness and codirectness in a same instance, lead to undecidable problems.

Applications to channel machines. Our interest in PEP and its variants comes from their close connection with channel systems. Here, PEP and its variants provide abstract versions of problems on channel machines, bringing greater clarity and versatility in both decidability and undecidability (more generally, hardness) proofs.

In this context, a further motivation for considering $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is that it allows solving the decidability of UCSTs, i.e., unidirectional channel systems (with one reliable and one lossy channel) *extended with the possibility of testing the contents of channels* (see chapter 6). We recall that PEP was introduced for UCSs, unidirectional channel systems where tests on channels are not supported [29, 28], and that $\mathsf{PEP}_{\text{dir}}$ corresponds to LCSs, i.e., lossy channel systems, for which decidability uses techniques from WSTS theory [7, 48]. Fig. 5.1 depicts the resulting situation.

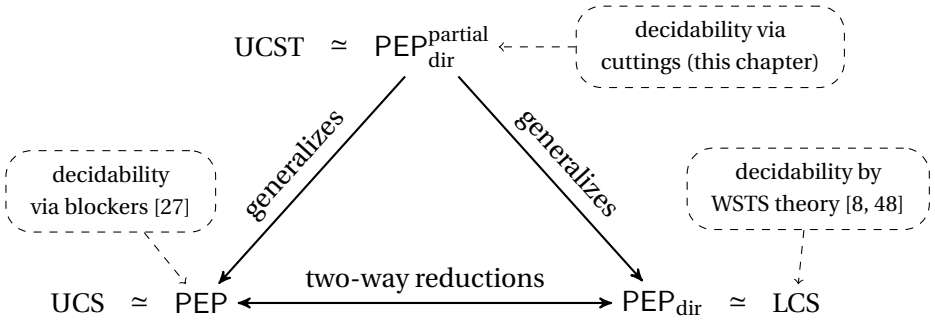


Figure 5.1: Three decidability proofs for PEP and variants

Outline of the chapter. Section 5.1 recalls basic notations and definitions. Section 5.1.1 explains the Length Function Theorem for Higman's Lemma. Section 5.2 contains the main result of the chapter, a direct decidability proof for $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, a problem subsuming both PEP and $\mathsf{PEP}_{\text{dir}}$. Section 5.3 builds on this result and shows the decidability of counting problems on $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$. Section 5.4 further shows the decidability of universal variants of

these questions. Section 5.5 contains undecidability results for some extensions of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$.

5.1 Preliminaries

With a language $R \subseteq \Gamma^*$ one associates a congruence (with respect to concatenation) given by $s \sim_R t \overset{\text{def}}{\Leftrightarrow} \forall x, y(xsy \in R \Leftrightarrow xty \in R)$ and called the Myhill congruence (also, the syntactic congruence). This equivalence has finite index if (and only if) R is regular. For regular R, let $\mu(R)$ denote this index: it satisfies $\mu(\widetilde{R}) = \mu(\Gamma^* \setminus R) = \mu(R)$ and $\mu(R \cap R') \leq \mu(R)\,\mu(R')$. Also, $\mu(R)$ is computable from R, and in particular, $\mu(R) \leq m^m$ when R is recognized by an m-state complete DFA [62].

5.1.1 Higman's Lemma and the length of bad sequences

Recall that the subword relation $\sqsubseteq$ on words over a finite alphabet is a wqo (Higman's Lemma, Corollary 2.21). For $n \in \mathbb{N}$, we say that a sequence (finite or infinite) of words is *n-good* if it contains an increasing subsequence of length n. It is *n-bad* otherwise. Higman's Lemma states that every infinite sequence is n-good for every n. Hence every n-bad sequence is finite.

It is often said that Higman's Lemma is "non-effective" or "non-constructive" since it does not come with any explicit information on the maximal length of bad sequences. Consequently, when one uses Higman's Lemma to prove that an algorithm terminates, no meaningful upper bound on the algorithm's running time is derived from the proof. However, the length of bad sequences can be bounded if one takes into account the complexity of the sequences, or more precisely, of the process that generates bad sequences. The interested reader can consult [112, 113] for more details. In this chapter we only use the simplest version of these results, i.e., the statement that when sequences only grow in a restricted way then the maximal length of bad sequences is computable, as we now explain.

For $k \in \mathbb{N}$, we say that a sequence of words $x_1, x_2, \ldots$ is *k-controlled* if $|x_i| \leq i\,k$ for all $i = 1, 2, \ldots$ Let $H(n, k, \Gamma)$ be the maximum length (if it exists) of an n-bad k-controlled sequence of words over a finite alphabet Γ.

Theorem 5.1 (Length Function Theorem)**.** *H is a computable (total) function. Furthermore, H is monotonic in its three arguments.*

Proof. Any prefix of a finite k-controlled n-bad sequence is k-controlled and n-bad. In particular, the empty sequence is. We arrange the set of all finite k-controlled n-bad sequences into a tree denoted $T_{n,k,\Gamma}$, or simply T, where the empty sequence is the root of T, and where a non-empty sequence of the form $x_1, \ldots, x_{l+1}$ is a child of its immediate prefix $x_1, \ldots, x_l$.

If T has an infinite path, this path is a chain of finite bad sequences linearly ordered by the prefix ordering and with which we can build an infinite k-controlled n-bad sequence by taking a limit. Thus T has no infinite paths since, by Higman's Lemma, Γ^* has no infinite bad sequences. Furthermore T is finitely branching, since the sequences it contains are k-controlled and Γ is finite. Thus, by Kőnig's Lemma, T is finite and $H(n,k,\Gamma)$ exists: it is the length of the longest sequence appearing in T, and also the length of T's longest path from the root.

H is computable since $T_{n,k,\Gamma}$ can be constructed effectively, starting from the root and listing the finitely many ways a current n-bad sequence can be extended in a k-controlled way. Finally, H is monotonic since, when $n' \leq n$ and $k' \leq k$, the n-bad k-controlled sequences over Γ include in particular all the n'-bad k'-controlled sequences over a subalphabet. □

Remark 5.2. Note that there is in general no maximum length of n-bad sequences over Γ if one does not restrict to k-controlled sequences. However, the proof of the Length Function Theorem can accommodate more liberal notions of controlled sequences, e.g., having $|x_i| \leq f(i)$ for all i, where f is a given computable function.

Note also that if $|\Gamma| = |\Gamma'|$ then $H(n,k,\Gamma) = H(n,k,\Gamma')$: only the number of different letters in Γ matters, and we sometimes write $H(n,k,p)$ for $H(n,k,\Gamma)$ where $p = |\Gamma|$. Upper bounds on $H(n,k,p)$ can be derived from the results given in [112] but these bounds are enormous, hard to express and hard to understand. In this chapter we content ourselves with the fact that H is computable.

Below, we use the Length Function Theorem contrapositively: a k-controlled sequence of length greater than $H(n,k,\Gamma)$ is necessarily n-good, i.e., contains an increasing subsequence $x_{i_1} \sqsubseteq x_{i_2} \sqsubseteq \cdots \sqsubseteq x_{i_n}$ of length n.

5.2 Deciding $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, or PEP with partial directness

We introduce $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, a problem generalizing both PEP and $\mathsf{PEP}_{\text{dir}}$, and show its decidability. This is proved by showing that if a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance has a solution, then it has a solution whose length is bounded by a computable function of the input. This is simpler and more direct than the earlier decidability proof (for PEP only) based on blockers [27].

Definition 5.3. $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the problem of deciding, given morphisms $u, v : \Sigma^* \to \Gamma^*$ and regular languages $R, R' \in \mathsf{Reg}(\Sigma)$, whether there is $\sigma \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ and $u(\tau) \sqsubseteq v(\tau)$ for all prefixes τ of σ belonging to R' (in which case σ is called a *solution*).
$\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the variant problem of deciding whether there is $\sigma \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ and $u(\tau) \sqsubseteq v(\tau)$ for all suffixes τ of σ that belong to R'.

Both PEP and $\mathsf{PEP}_{\text{dir}}$ are special cases of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$, obtained by taking $R' = \emptyset$ and $R' = \Sigma^*$ respectively. Obviously $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are two equivalent presentations, modulo mirroring, of a same problem. Given a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ or $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance, we let $K_u \stackrel{\text{def}}{=} \max_{a\in\Sigma}|u(a)|$ denote the *expansion factor* of u and define

$$L \stackrel{\text{def}}{=} H(\mu(R)\,\mu(R')+1, K_u, \Gamma)$$

(recall that $\mu(R)$ and $\mu(R')$ are the indexes of the Myhill congruences associated with R and R', while $H(n,k,\Gamma)$ is defined with the Length Function Theorem).

In this section we prove:

Theorem 5.4. A $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance has a solution if, and only if, it has a solution of length at most $2L$.
This entails that $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is decidable.

Decidability is an obvious consequence since the length bound is computable, and since it is easy to check whether a candidate σ is a solution.

For the proof of Theorem 5.4, we consider an arbitrary $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance (Σ,Γ,u,v,R,R') and a solution σ. Write $N = |\sigma|$ for its length, $\sigma[0,i)$ and $\sigma[i,N)$ for, respectively, its prefix of length i and its suffix of length $N-i$. Two indices $i,j \in [0,N]$ are *congruent* if $\sigma[i,N) \sim_R \sigma[j,N)$ and $\sigma[i,N) \sim_{R'} \sigma[j,N)$. When σ is fixed, as in the rest of this section, we use shorthand notations like $u_{0,i}$ and $v_{i,j}$ to denote the images, here $u(\sigma[0,i))$ and $v(\sigma[i,j))$, of factors of σ.

We prove two "cutting lemmas" giving sufficient conditions for "cutting" a solution $\sigma = \sigma[0,N)$ along certain indices $a < b$, yielding a shorter solution $\sigma' = \sigma[0,a)\sigma[b,N)$, i.e., σ with the factor $\sigma[a,b)$ cut out. Here the following notation is useful. We associate, with every suffix τ of σ', a corresponding suffix, denoted $S(\tau)$, of σ: if τ is a suffix of $\sigma[b,N)$, then $S(\tau) \stackrel{\text{def}}{=} \tau$, otherwise, $\tau = \sigma[i,a)\sigma[b,N)$ for some $i < a$ and we let $S(\tau) \stackrel{\text{def}}{=} \sigma[i,N)$. In particular $S(\sigma') = \sigma$.

An index $i \in [0,N]$ is said to be *blue* if $u_{i,N} \sqsubseteq v_{i,N}$, it is *red* otherwise. In particular, N is blue trivially, 0 is blue since σ is a solution, and i is blue whenever $\sigma[i,N) \in R'$. If i is a blue index, let $l_i \in \Gamma^*$ be the longest suffix of $u_{0,i}$ such that $l_i\, u_{i,N} \sqsubseteq v_{i,N}$ and call it the *left margin* at i.

Lemma 5.5 (Cutting lemma for blue indices)**.** Let $a < b$ be two congruent and blue indices. If $l_a \sqsubseteq l_b$, then $\sigma' = \sigma[0,a)\sigma[b,N)$ is a solution (shorter than σ).

Proof. Clearly $\sigma' \in R$ since $\sigma \in R$ and a and b are congruent. Also, for all suffixes τ of σ', $S(\tau) \in R'$ iff $\tau \in R'$.

We claim that, for any suffix τ of σ', if $u(S(\tau)) \sqsubseteq v(S(\tau))$ then $u(\tau) \sqsubseteq v(\tau)$. This is obvious when $\tau = S(\tau)$, so we assume $\tau \neq S(\tau)$, i.e., $\tau = \sigma[i,a)\sigma[b,N)$ and $S(\tau) = \sigma[i,N)$ for some $i < a$. Assume $u(S(\tau)) \sqsubseteq v(S(\tau))$, i.e., $u_{i,N} \sqsubseteq v_{i,N}$. Now both $u_{i,a}$ and l_a are suffixes of $u_{0,a}$, so that one is a suffix of the other, which gives two cases.

1. If $u_{i,a}$ is a suffix of l_a, then

$$\begin{aligned} u(\tau) = u_{i,a}\,u_{b,N} &\sqsubseteq l_a\,u_{b,N} && \text{since } u_{i,a} \text{ is a suffix of } l_a,\\ &\sqsubseteq l_b\,u_{b,N} && \text{since } l_a \sqsubseteq l_b \text{ by assumption},\\ &\sqsubseteq v_{b,N} && \text{by definition of } l_b,\\ &\sqsubseteq v_{i,a}\,v_{b,N} = v(\tau)\,. \end{aligned}$$

2. Otherwise, $u_{i,a} = x\,l_a$ for some x, as illustrated in Fig. 5.2 where slanted arrows follow the rightmost embedding of $u(\sigma)$ into $v(\sigma)$. Here $u_{i,N} \sqsubseteq v_{i,N}$ rewrites as $x\,l_a\,u_{a,N} \sqsubseteq v_{i,a}\,v_{a,N}$.

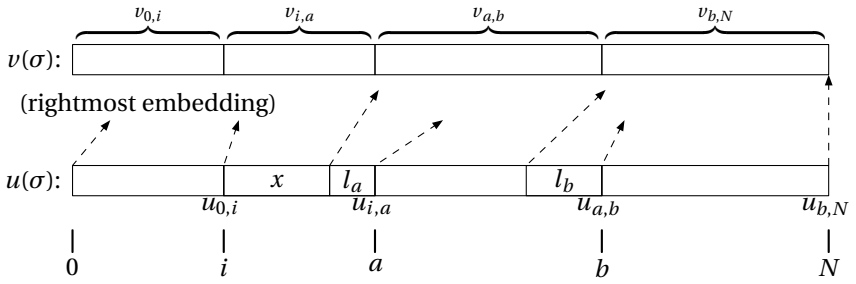


Figure 5.2: Schematics for Lemma 5.5, with $l_a \sqsubseteq l_b$

Now, and since l_a is (by definition) the longest suffix for which $l_a\,u_{a,N} \sqsubseteq v_{a,N}$, Lemma 2.7 entails $x \sqsubseteq v_{i,a}$. Then

$$\begin{aligned} u(\tau) = u_{i,a}\,u_{b,N} &= x\,l_a\,u_{b,N}\\ &\sqsubseteq v_{i,a}\,l_b\,u_{b,N} && \text{since } x \sqsubseteq v_{i,a} \text{ and } l_a \sqsubseteq l_b,\\ &\sqsubseteq v_{i,a}\,v_{b,N} = v(\tau) && \text{by definition of } l_b. \end{aligned}$$

We can now infer $u(\tau) \sqsubseteq v(\tau)$ for any suffix $\tau \in R'$ (or for $\tau = \sigma'$) from the corresponding $u(S(\tau)) \sqsubseteq v(S(\tau))$. This shows that σ' is a solution. □

If i is a red index, i.e., if $u_{i,N} \not\sqsubseteq v_{i,N}$, let $r_i \in \Gamma^*$ be the shortest prefix of $u_{i,N}$ such that $r_i^{-1}u_{i,N} \sqsubseteq v_{i,N}$ (equivalently $u_{i,N} \sqsubseteq r_i\,v_{i,N}$) and call it the *right margin* at i.

Lemma 5.6 (Cutting lemma for red indices)**.** Let $a < b$ be two congruent and red indices. If $r_b \sqsubseteq r_a$, then $\sigma' = \sigma[0,a)\sigma[b,N)$ is a solution (shorter than σ).

Proof. Write x for $r_b^{-1}u_{b,N}$. Then $u_{b,N} = r_b\,x$ and $x \sqsubseteq v_{b,N}$. We proceed as for Lemma 5.5 and show that $u(S(\tau)) \sqsubseteq v(S(\tau))$ implies $u(\tau) \sqsubseteq v(\tau)$ for all suffixes τ of σ'. Assume $u(S(\tau)) \sqsubseteq v(S(\tau))$ for some τ. The only interesting case is when $\tau \neq S(\tau)$, i.e., when $\tau = \sigma[i,a)\sigma[b,N)$ for some $i < a$ (see Figure 5.3).

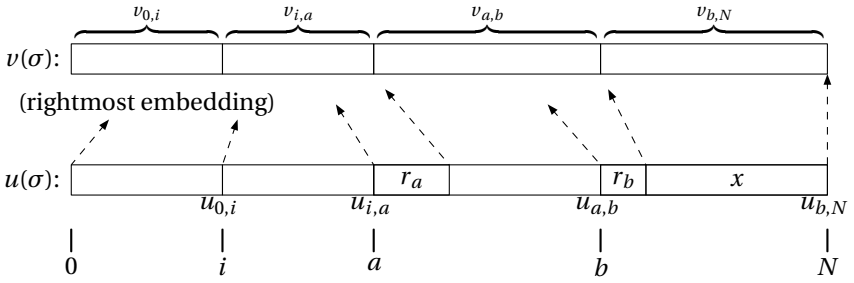


Figure 5.3: Schematics for Lemma 5.6, with $r_b \sqsubseteq r_a$

From $u_{i,N} = u_{i,a}\,u_{a,N} \sqsubseteq v_{i,a}\,v_{a,N} = v_{i,N}$, i.e., $u(S(\tau)) \sqsubseteq v(S(\tau))$, and $u_{a,N} \not\sqsubseteq v_{a,N}$ (since a is a red index), Lemma 2.8 entails $u_{i,a}\,r_a \sqsubseteq v_{i,a}$ by definition of r_a. Then

$$\begin{aligned} u(\tau) = u_{i,a}\,u_{b,N} = u_{i,a}\,r_b\,x &\sqsubseteq u_{i,a}\,r_a\,v_{b,N} && \text{since } r_b \sqsubseteq r_a \text{ and } x \sqsubseteq v_{b,N},\\ &\sqsubseteq v_{i,a}\,v_{b,N} = v(\tau) && \text{since } u_{i,a}\,r_a \sqsubseteq v_{i,a}. \end{aligned}$$ □

For the next step let $g_1 < g_2 < \cdots < g_{N_1}$ be all the blue indices in σ, and let $b_1 < b_2 < \cdots < b_{N_2}$ be the red indices. Observe that $N_1 + N_2 = N + 1$ since each index in $0, \ldots, N$ is either blue or red. We consider the corresponding sequences $(l_{g_i})_{i=1,\ldots,N_1}$ of left margins and $(r_{b_i})_{i=1,\ldots,N_2}$ of right margins.

Lemma 5.7. $|l_{g_i}| \le (i-1)\times K_u$ for all $i = 1, \ldots, N_1$, and $|r_{b_i}| \le (N_2 - i + 1)\times K_u$ for all $i = 1, \ldots, N_2$. In other words, the sequence of left margins and the *reversed* sequence of right margins are K_u-controlled.

Proof. We prove that $|l_{g_i}| \le (i-1)\times K_u$ by induction on i, showing $|l_{g_1}| = 0$ and $|l_{g_i}| - |l_{g_{i-1}}| \le K_u$ for $i > 1$.

The base case $i = 1$ is easy: obviously $g_1 = 0$ since 0 is a blue index, and $l_0 = \epsilon$ since it is the only suffix of $u_{0,0} = \epsilon$, so that $|l_{g_1}| = 0$.

For the inductive step $i > 1$, write p for g_{i-1} and q for g_i. By definition, l_p is the longest suffix of $u_{0,p}$ with $l_p\,u_{p,N} = l_p\,u_{p,q}\,u_{q,N} \sqsubseteq v_{p,N}$. Since $l_q\,u_{q,N} \sqsubseteq v_{q,N} \sqsubseteq v_{p,N}$, l_q must be a suffix of $l_p\,u_{p,q}$, hence $|l_q| \le |l_p| + |u_{p,q}| \le |l_p| + K_u(q-p)$. This proves the claim in the case where $q = p+1$, i.e., when p and $p+1$ are blue.

There remains the case where $q > p+1$ and where all the indices from $p+1$ to $q-1$ are red. Thus in particular $u_{q-1,N} = u_{q-1,q}\,u_{q,N} \not\sqsubseteq v_{q-1,N}$. On the other hand q is blue and $l_q\,u_{q,N} \sqsubseteq v_{q,N} \sqsubseteq v_{q-1,N}$. We conclude that l_q must be a suffix of $u_{q-1,q}$, so that $|l_q| \le K_u$ which proves the claim.

The reasoning for $|r_{b_i}|$ is similar:

If $b_{i+1} = b_i + 1$, then both b_i and the next index are red. Then r_{b_i} is a prefix of $u_{b_i,b_i+1}\,r_{b_{i+1}}$ so that $|r_{b_i}| \le K_u + |r_{b_{i+1}}|$.

If $b_{i+1} > b_i + 1$, then $b_i + 1$ is blue and r_{b_i} is a prefix of u_{b_i, b_i+1} so that $|r_{b_i}| \leq K_u$.

For the base case, we have $b_{N_2} < N$ since N is blue. Hence $b_{N_2} + 1$ is blue and $|r_{b_{N_2}}| \leq K_u$ as above.

Finally, $|r_{b_i}| \leq (N_2 + 1 - i) \times K_u$ for all $i = 1, \ldots, N_2$. □

We are now ready to conclude the proof of Theorem 5.4. Let $N_c \stackrel{\text{def}}{=} \mu(R)\,\mu(R') + 1$ and $L \stackrel{\text{def}}{=} H(N_c, K_u, \Gamma)$ and assume that $N > 2L$. Since $N_1 + N_2 = N + 1$, either σ has at least $L + 1$ blue indices and, by definition of L and H, there exist N_c blue indices $a_1 < a_2 < \cdots < a_{N_c}$ with $l_{a_1} \sqsubseteq l_{a_2} \sqsubseteq \cdots \sqsubseteq l_{a_{N_c}}$, or σ has at least $L + 1$ red indices and there exist N_c red indices $a'_1 < a'_2 < \cdots < a'_{N_c}$ with $r_{a'_{N_c}} \sqsubseteq \cdots \sqsubseteq r_{a'_2} \sqsubseteq r_{a'_1}$ (since it is the reversed sequence of right margins that is controlled). Out of $N_c = \mu(R)\,\mu(R') + 1$ indices, two must be congruent, fulfilling the assumptions of either Lemma 5.5 or Lemma 5.6. Therefore σ can be cut to obtain a shorter solution.

Since $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are equivalent problems modulo mirroring of R, u and v, we deduce that $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ too is decidable, and more precisely:

Corollary 5.8. A $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance has a solution if, and only if, it has a solution of length at most $2L$.

5.3 Counting the number of solutions

We consider two counting questions: $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the question whether a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ instance has infinitely many solutions (a decision problem), while $\#\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is the problem of computing the number of solutions of the instance (a number in $\mathbb{N} \cup \{\infty\}$). For technical convenience, we often deal with the (equivalent) codirected versions, $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\#\mathsf{PEP}^{\text{partial}}_{\text{codir}}$.

For an instance $(\Sigma, \Gamma, u, v, R, R')$, we let $K_v \stackrel{\text{def}}{=} \max_{a \in \Sigma} |v(a)|$ and define

$$M \stackrel{\text{def}}{=} H(\mu(R)\,\mu(R') + 1, K_v, \Gamma)\,, \qquad M' \stackrel{\text{def}}{=} H\big((2M+2)\mu(R)\,\mu(R') + 1, K_u, \Gamma\big)\,.$$

In this section we prove:

Theorem 5.9. For a $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ or $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance, the following are equivalent:

(a) it has infinitely many solutions;

(b) it has a solution of length N with $2M < N$;

(c) it has a solution of length N with $2M < N \leq 2M'$.

This entails the decidability of $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, and also the computability of $\#\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\#\mathsf{PEP}^{\text{partial}}_{\text{codir}}$.

As with Theorem 5.4, the length bounds $2M$ and $2M'$ are computable, so that $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and $\exists^{\infty}\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ can be decided by finite enumeration. When the number of solutions is finite, counting them can also be done by finite enumeration since we know all solutions have then length at most $2M$.

For the proof of Theorem 5.9, we first observe that if the instance has a solution of length $N > 2M$, it has a solution with R replaced by $R^{>} \stackrel{\text{def}}{=} R \cap \Sigma^{2M+1}\Sigma^*$. The syntactic congruence associated with $R^{>}$ has index at most $(2M+2)\mu(R)$. From Theorem 5.4, we deduce that the modified instance has a solution of length at most $2M'$. Hence (b) and (c) are equivalent.

It remains to show that (b) implies (a) since obviously (a) implies (b). For this we fix an arbitrary $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance $(\Sigma, \Gamma, u, v, R, R')$ and consider a solution σ, of length N. We develop two so-called "iteration lemmas" that are similar to the cutting lemmas from section 5.2, with the difference that they expand σ instead of reducing it.

As before, an index $i \in [0, N]$ is said to be *blue* if $u_{i,N} \sqsubseteq v_{i,N}$, and *red* otherwise. With a blue (resp., a red) index $i \in [0, N]$ we associate a word s_i (resp., t_i) in Γ^*. The s_i's and t_i's are analogous to the l_i's and r_i's from section 5.2, however they are factors of $v(\sigma)$, not of $u(\sigma)$ like l_i or r_i, and this explains the difference between M and L. The terms "left margin" and "right margin" will be reused here for these factors.

We start with blue indices. For a blue index $i \in [0, N]$, let s_i be the longest prefix of $v_{i,N}$ such that $u_{i,N} \sqsubseteq s_i^{-1} v_{i,N}$ (equivalently, such that $s_i\, u_{i,N} \sqsubseteq v_{i,N}$) and call it the *right margin* at i.

Lemma 5.10. Suppose $a < b$ are two blue indices with $s_b \sqsubseteq s_a$. Then for all $k \geq 1$, $s_a (u_{a,b})^k \sqsubseteq (v_{a,b})^k s_b$.

Proof. $s_a\, u_{a,N} \sqsubseteq v_{a,N}$ expands as $(s_a\, u_{a,b}) u_{b,N} \sqsubseteq v_{a,b}\, v_{b,N}$. Since b is blue, $u_{b,N} \sqsubseteq v_{b,N}$ and, by definition of s_b, Lemma 2.9 further yields $s_a\, u_{a,b} \sqsubseteq v_{a,b}\, s_b$. One concludes with Lemma 2.11, using $s_b \sqsubseteq s_a$. □

Lemma 5.11 (Iteration lemma for blue indices)**.** Let $a < b$ be two congruent blue indices. If $s_b \sqsubseteq s_a$, then for every $k \geq 1$, $\sigma' = \sigma[0, a).\sigma[a, b)^k.\sigma[b, N)$ is a solution.

Proof. Let τ be any suffix of σ'. We show that $u(\tau) \sqsubseteq v(\tau)$ when $\tau \in R'$ or $\tau = \sigma'$, which will complete the proof. There are three cases, depending on how long τ is.

- τ is a suffix of $\sigma[a, N)$. Then τ is a suffix of σ itself, and this case is trivial since σ is a solution.

- τ is $\sigma[i,b)\sigma[a,b)^p\sigma[b,N)$ for some $p \geq 1$ and $a < i \leq b$. Since a and b are congruent, $\tau \in R'$ implies $\sigma[i,N) \in R'$. Thus $u_{i,N} \sqsubseteq v_{i,N}$, hence $u_{i,b} \sqsubseteq v_{i,b}\, s_b$ (since $u_{b,N} \sqsubseteq v_{b,N}$).

$$\begin{aligned}
u(\tau) &= u_{i,b}(u_{a,b})^p\, u_{b,N} \\
&\sqsubseteq v_{i,b}\, s_b(u_{a,b})^p\, u_{b,N} \\
&\sqsubseteq v_{i,b}\, s_a(u_{a,b})^p\, u_{b,N} && \text{since } s_b \sqsubseteq s_a \\
&\sqsubseteq v_{i,b}(v_{a,b})^p\, s_b\, u_{b,N} && \text{by Lemma 5.10} \\
&\sqsubseteq v_{i,b}(v_{a,b})^p\, v_{b,N} && \text{by definition of } s_b \\
&= v(\tau)\,.
\end{aligned}$$

- τ is $\sigma[i,a)\sigma[a,b)^k\sigma[b,N)$ for some $0 \leq i < a$. Since a and b are congruent, $\tau \in R'$ (or $\tau = \sigma$) implies $u_{i,N} \in R'$ (or $u_{i,N} = \sigma$) so that $u_{i,N} \sqsubseteq v_{i,N}$, from which we deduce $u_{i,a} \sqsubseteq v_{i,a}\, s_a$ as in the previous case. Then, using Lemma 5.10 and $s_b\, u_{b,N} \sqsubseteq v_{b,N}$, we get

$$\begin{aligned}
u(\tau) &= u_{i,a}(u_{a,b})^k\, u_{b,N} \\
&\sqsubseteq v_{i,a}\, s_a(u_{a,b})^k\, u_{b,N} \\
&\sqsubseteq v_{i,a}(v_{a,b})^k\, s_b\, u_{b,N} && \text{by Lemma 5.10} \\
&\sqsubseteq v_{i,a}(v_{a,b})^k\, v_{b,N} && \text{by definition of } s_b \\
&= v(\tau)\,.
\end{aligned}$$

□

Now to red indices. For a red index $i \in [0,N]$, let t_i be the shortest suffix of $v_{0,i}$ such that $u_{i,N} \sqsubseteq t_i\, v_{i,N}$. This is called the *left margin* at i. Thus, for a blue j such that $j < i$, $u_{j,N} \sqsubseteq v_{j,N}$ implies $u_{j,i}\, t_i \sqsubseteq v_{j,i}$ by Lemma 2.10.

Lemma 5.12 (Iteration lemma for red indices)**.** Let $a < b$ be two congruent red indices. If $t_a \sqsubseteq t_b$, then for every $k \geq 1$, $\sigma' = \sigma[0,a).\sigma[a,b)^k.\sigma[b,N)$ is a solution.

Proof. Let τ be any suffix of σ'. We show that $u(\tau) \sqsubseteq v(\tau)$ when $\tau \in R'$ or $\tau = \sigma'$, which will complete the proof. There are three cases, depending on how long τ is.

- τ is a suffix of $\sigma[a,N)$. Then τ is a suffix of σ itself, and this case is trivial since σ is a solution.

- τ is $\sigma[i,b)\sigma[a,b)^p\sigma[b,N)$ for some $p \geq 1$ and $a < i \leq b$. Since a and b are congruent, $\tau \in R'$ implies $\sigma[i,N) \in R'$ and so $u_{i,N} \sqsubseteq v_{i,N}$. By definition of t_a, we have $u_{a,b}u_{b,N} \sqsubseteq (t_a v_{a,b})v_{b,N}$. Using Lemma 2.10 and the definition of t_b we get $u_{a,b}\, t_b \sqsubseteq t_a\, v_{a,b}$, and

then $(u_{a,b})^p\, t_b \sqsubseteq t_a (v_{a,b})^p$ with Lemma 2.12. Then

$$\begin{aligned}
u(\tau) &= u_{i,b}(u_{a,b})^p\, u_{b,N} \\
&\sqsubseteq u_{i,b}(u_{a,b})^p\, t_b\, v_{b,N} && \text{by definition of } t_b \\
&\sqsubseteq u_{i,b}\, t_a (v_{a,b})^p\, v_{b,N} && \text{as above} \\
&\sqsubseteq u_{i,b}\, t_b (v_{a,b})^p\, v_{b,N} && \text{since } t_a \sqsubseteq t_b \\
&\sqsubseteq v_{i,b}(v_{a,b})^p\, v_{b,N} && \text{since } u_{i,N} \sqsubseteq v_{i,N},\ b \text{ is red, Lemma 2.10} \\
&= v(\tau)\,.
\end{aligned}$$

- τ is $\sigma[i,a)\sigma[a,b)^k\sigma[b,N)$ for some $0 \le i < a$ and $k \ge 1$. Since a and b are congruent, $\tau \in R'$ (or $\tau = \sigma$) implies $u_{i,N} \in R'$ (or $u_{i,N} = \sigma$) so that $u_{i,N} \sqsubseteq v_{i,N}$, from which we deduce $u_{i,a}\, t_a \sqsubseteq v_{i,a}$ as in the previous case. Then

$$\begin{aligned}
u(\tau) &= u_{i,a}(u_{a,b})^k\, u_{b,N} \\
&\sqsubseteq u_{i,a}(u_{a,b})^k\, t_b\, v_{b,N} && \text{by definition of } t_b \\
&\sqsubseteq u_{i,a}\, t_a\, (v_{a,b})^k\, v_{b,N} && \text{as before} \\
&\sqsubseteq v_{i,a}(v_{a,b})^k\, v_{b,N} && \text{as above} \\
&= v(\tau)\,. && \square
\end{aligned}$$

We may now prove that the $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ instance has infinitely many solutions if it has solution of length $N > 2M$, i.e., that (b) implies (a) in Theorem 5.9.

Lemma 5.13. Suppose $a < b$ are two blue indices. Then s_a is a prefix of $v_{a,b}\, s_b$.

Proof. Both s_a and $v_{a,b}s_b$ are prefixes of $v_{a,N}$, hence one of them is a prefix of the other. Assume, by way of contradiction, that $v_{a,b}\, s_b$ is a proper prefix of s_a, say $s_a = v_{a,b}\, s_b\, x$ for some $x \neq \epsilon$. Then $s_a\, u_{a,N} \sqsubseteq v_{a,N}$ rewrites as $v_{a,b}\, s_b\, x\, u_{a,N} \sqsubseteq v_{a,b}\, v_{b,N}$. Cancelling $v_{a,b}$ on both sides gives $s_b\, x\, u_{a,N} \sqsubseteq v_{b,N}$, i.e., $(s_b\, x\, u_{a,b})u_{b,N} \sqsubseteq v_{b,N}$, which contradicts the definition of s_b. $\square$

Suppose there are N_1 blue indices in σ, say $g_1 < g_2 < \cdots < g_{N_1}$; and N_2 red indices, say $b_1 < b_2 < \cdots < b_{N_2}$.

Lemma 5.14. $|s_{g_i}| \le (N_1 - i + 1) \times K_v$ for all $i = 1, \ldots, N_1$, and $|t_{b_i}| \le (i-1) \times K_v$ for all $i = 1, \ldots, N_2$. That is, the *reversed* sequence of right margins and the sequence of left margins are K_v-controlled.

Proof. We start with blue indices and right margins.

We now show that $s_{g_{N_1}}, \ldots, s_{g_1}$ is K_v-controlled. N is a blue index, and $|s_N| = 0$. For $i \in [0, N)$, if both i and $i+1$ are blue indices, then by Lemma 5.13, $|s_i| \le |s_{i+1}| + K_v$. If i is blue and $i+1$

is red, then it is easy to see that s_i is a prefix of $v(\sigma_i)$, and hence $|s_i| \leq K_v$. So we get that $s_{g_{N_1}}, \ldots, s_{g_1}$ is K_v-controlled.
Now to red indices and left margins. 0 is not a red index. For $i \in [0, N)$, if both i and $i+1$ are red, then it is easy to see that t_{i+1} is a suffix of $t_i\, v(\sigma_i)$, and so $|t_{i+1}| \leq |t_i| + K_v$. If i is blue and $i+1$ is red, then t_{i+1} is a suffix of $v(\sigma_i)$, and so $|t_{i+1}| \leq K_v$. So we get that $t_{b_1}, \ldots, t_{b_{N_2}}$ is K_v-controlled. □

Assume that σ is a long solution of length $N > 2M$. At least $M+1$ indices among $[0, N]$ are blue, or at least $M+1$ are red. We apply one of the two above claims, and from either $s_{g_{N_1}}, \ldots, s_{g_1}$ (if $N_1 > M$) or $t_{b_1}, \ldots, t_{b_{N_2}}$ (if $N_2 > M$) we get an increasing subsequence of length $\mu(R)\,\mu(R') + 1$. Among these there must be two congruent indices. Then we get infinitely many solutions by Lemma 5.11 or Lemma 5.12.

5.4 Universal variants of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$

We consider universal variants of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ (or rather $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ for the sake of uniformity). Formally, given instances $(\Sigma, \Gamma, u, v, R, R')$ as usual, $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the question whether *every* $\sigma \in R$ is a solution, i.e., satisfies both $u(\sigma) \sqsubseteq v(\sigma)$ and $u(\tau) \sqsubseteq v(\tau)$ for all suffixes τ that belong to R'. Similarly, $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the question whether "almost all", i.e., *all but finitely many*, σ in R are solutions, and $\#\neg\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is the associated counting problem that asks how many $\sigma \in R$ are not solutions.

These universal questions can also be seen as Post *non*-embedding problems, asking whether there exists some $\sigma \in R$ such that $u(\sigma) \not\sqsubseteq v(\sigma)$? Introduced in [32] with $\forall\mathsf{PEP}$, they are significantly less challenging than the standard PEP problems, and decidability is easier to establish. For this reason, we just show in this chapter how $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ reduce to $\forall^\infty\mathsf{PEP}$ whose decidability was shown in [32]. The point is that partial codirectness constraints can be eliminated since universal quantifications commute with conjunctions (and since the codirectness constraint is universal itself).

Lemma 5.15. $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ many-one reduce to $\forall^\infty\mathsf{PEP}$.

Corollary 5.16. $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are decidable, $\#\neg\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is computable.

We now prove Lemma 5.15. First, $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ easily reduces to $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$: add an extra letter z to Σ with $u(z) = v(z) = \epsilon$ and replace R and R' with $R.z^*$ and $R'.z^*$. Hence the second half of the lemma entails its first half by transitivity of reductions.
For reducing $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, it is easier to start with the negation of our question:

$$\exists^\infty \sigma \in R : \big(u(\sigma) \not\sqsubseteq v(\sigma) \text{ or } \sigma \text{ has a suffix } \tau \text{ in } R' \text{ with } u(\tau) \not\sqsubseteq v(\tau)\big). \qquad (*)$$

Call $\sigma \in R$ a *type 1 witness* if $u(\sigma) \not\sqsubseteq v(\sigma)$, and a *type 2 witness* if it has a suffix $\tau \in R'$ with $u(\tau) \not\sqsubseteq v(\tau)$. Statement $(*)$ holds if, and only if, there are infinitely many type 1 witnesses or infinitely many type 2 witnesses. The existence of infinitely many type 1 witnesses (call that "case 1") is the negation of a $\forall^\infty\mathsf{PEP}$ question. Now suppose that there are infinitely many type 2 witnesses, say $\sigma_1, \sigma_2, \ldots$ For each i, pick a suffix τ_i of σ_i such that $\tau_i \in R'$ and $u(\tau_i) \not\sqsubseteq v(\tau_i)$. The set $\{\tau_i \mid i = 1,2,\ldots\}$ of these suffixes can be finite or infinite. If it is infinite ("case 2a"), then

$$u(\tau) \not\sqsubseteq v(\tau) \text{ for infinitely many } \tau \in (\overrightarrow{R} \cap R')\,, \qquad (**)$$

where $\overrightarrow{R}$ is short for $\overrightarrow{^{\geq 0}R}$ and for $k \in \mathbb{N}$, $\overrightarrow{^{\geq k}R} \stackrel{\text{def}}{=} \{y \mid \exists x : (|x| \geq k \text{ and } xy \in R)\}$ is the set of the suffixes of words from R one obtains by removing at least k letters. Observe that, conversely, $(**)$ implies the existence of infinitely many type 2 witnesses (for a proof, pick $\tau_1 \in \overrightarrow{R} \cap R'$ satisfying the above, choose $\sigma_1 \in R$ of which τ_1 is a suffix. Then choose τ_2 such that $|\tau_2| > |\sigma_1|$, and proceed similarly).

On the other hand, if $\{\tau_i \mid i = 1,2,\ldots\}$ is finite ("case 2b"), then there is a $\tau \in R'$ such that $u(\tau) \not\sqsubseteq v(\tau)$ and $\sigma'\tau \in R$ for infinitely many σ'. By a standard pumping argument, the second point is equivalent to the existence of some such σ' with also $|\sigma'| > k_R$, where k_R is the size of a NFA for R (taking $k_R = \mu(R)$ also works). Write now $\hat{R}$ for $\overrightarrow{^{>k_R}R}$: if $\{\tau_i \mid i = 1,2,\ldots\}$ is finite, then $u(\tau) \not\sqsubseteq v(\tau)$ for some τ in $(R' \cap \hat{R})$, and conversely this implies the existence of infinitely many type 2 witnesses.

To summarize, and since $\overrightarrow{R}$ and $\hat{R}$ are regular and effectively computable from R, we have just reduced $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ to the following conjunction

$$\begin{aligned}
&\forall^\infty \sigma \in R : u(\sigma) \sqsubseteq v(\sigma) && \text{(not case 1)}\\
\bigwedge\ &\forall^\infty \tau \in (\overrightarrow{R} \cap R') : u(\tau) \sqsubseteq v(\tau) && \text{(not case 2a)}\\
\bigwedge\ &\forall \tau \in (\hat{R} \cap R') : u(\tau) \sqsubseteq v(\tau)\,. && \text{(not case 2b)}
\end{aligned}$$

This is now reduced to a single $\forall^\infty\mathsf{PEP}$ instance by rewriting the $\forall\mathsf{PEP}$ into a $\forall^\infty\mathsf{PEP}$ (as explained in the beginning of this proof) and relying on a distributivity property of the form

$$\bigwedge_{i=1}^{n} \left[\forall^\infty \sigma \in R_i : u(\sigma) \sqsubseteq v(\sigma)\right] \equiv \forall^\infty \sigma \in \left[\bigcup_{i=1}^{n} R_i\right] : u(\sigma) \sqsubseteq v(\sigma)$$

to handle the resulting conjunction of 3 $\forall^\infty\mathsf{PEP}$ instances.

To see that $\#\neg\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is computable, consider a given instance $(\Sigma, \Gamma, u, v, R, R')$. We need to compute the number of $\sigma \in R$ which are not solutions. First we use $\forall^\infty\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ to find whether this number is finite or infinite. If it is infinite, we are done. Otherwise, we find a bound on the length of the longest non-solution: it is the smallest n such that replacing R by $R \cap \{w : |w| > n\}$ results in a "yes" instance of $\forall\mathsf{PEP}^{\text{partial}}_{\text{codir}}$. This n is guaranteed to exist. Then the set of non-solutions in R can be counted by explicit enumeration.

5.5 Undecidability for $\mathsf{PEP}_{\text{co\&dir}}$ and other extensions

The decidability of $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is a non-trivial generalization of previous results for PEP. It is a natural question whether one can further generalize the idea of partial directness and maintain decidability. In this section we describe two attempts that lead to undecidability, even though they remain inside the regular PEP framework.[2]

Allowing non-regular R'. One direction for extending $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ is to allow *more expressive R' sets* for partial (co)directness. Let $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ and $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ be like $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ except that R' can be any deterministic context-free $R' \in \mathsf{DCFL}(\Sigma)$ (resp., any Presburger-definable $R' \in \mathsf{Pres}(\Sigma)$, i.e., a language consisting of all words whose Parikh image lies in a given Presburger, or semilinear, subset of $\mathbb{N}^{|\Sigma|}$). Note that $R \in \mathsf{Reg}(\Sigma)$ is still required.

Theorem 5.17 (Undecidability)**.** $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ and $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ are Σ^0_1-complete.

Since both problems clearly are in Σ^0_1, one only has to prove hardness by reduction, e.g., from PCP, Post's Correspondence Problem. Let (Σ, Γ, u, v) be a PCP instance (where the question is whether there exists $x \in \Sigma^+$ such that $u(x) = v(x)$). Extend Σ and Γ with new symbols: $\Sigma' \stackrel{\text{def}}{=} \Sigma \cup \{1,2\}$ and $\Gamma' \stackrel{\text{def}}{=} \Gamma \cup \{\#\}$. Now define $u', v' : \Sigma'^* \to \Gamma'^*$ by extending u, v on the new symbols with $u'(1) = v'(2) = \epsilon$ and $u'(2) = v'(1) = \#$. Define now $R = 12\Sigma^+$ and $R' = \{\tau 2 \tau' \mid \tau, \tau' \in \Sigma^* \text{ and } |u(\tau\tau')| \neq |v(\tau\tau')|\}$. Note that R' is deterministic context-free and Presburger-definable.

Lemma 5.18. The PCP instance (Σ, Γ, u, v) has a solution if and only if the $\mathsf{PEP}^{\text{partial[Pres]}}_{\text{codir}}$ and $\mathsf{PEP}^{\text{partial[DCFL]}}_{\text{codir}}$ instance $(\Sigma', \Gamma', u', v', R, R')$ has a solution.

Proof. Suppose σ is a solution to the PCP problem. Then $\sigma \neq \epsilon$ and $u(\sigma) = v(\sigma)$. Now $\sigma' \stackrel{\text{def}}{=} 12\sigma$ is a solution to the partially codirected problem since $12\sigma \in R$, $u'(12\sigma) = \#u(\sigma) \sqsubseteq v'(12\sigma) = \#v(\sigma)$, and σ' has no suffix in R' (indeed $2\sigma \notin R'$ since $|u(\sigma)| = |v(\sigma)|$).

Conversely, suppose σ' is a solution to the partially codirected problem. Then $\sigma' = 12\sigma$ for some $\sigma \neq \epsilon$. Since $u'(\sigma') = \#u(\sigma) \sqsubseteq v'(\sigma') = \#v(\sigma)$, we have $u(\sigma) \sqsubseteq v(\sigma)$. If $|u(\sigma)| \neq |v(\sigma)|$, then $2\sigma \in R'$, and so we must have $u'(2\sigma) = \#u(\sigma) \sqsubseteq v'(2\sigma) = v(\sigma)$. This is not possible as $\#$ does not occur in $v(\sigma)$. So $|u(\sigma)| = |v(\sigma)|$, and $u(\sigma) = v(\sigma)$. Thus σ is a solution to the PCP problem. □

Combining directness and codirectness. Another direction is to allow *combining* directness and codirectness constraints. Formally, $\mathsf{PEP}_{\text{co\&dir}}$ is the problem of deciding, given Σ, Γ, u, v, and $R \in \mathsf{Reg}(\Sigma)$ as usual, whether there exists $\sigma \in R$ such that $u(\tau) \sqsubseteq v(\tau)$ and

[2] PEP is undecidable if we allow constraint sets R outside $\mathsf{Reg}(\Sigma)$ [27]. Other extensions, like $\exists x \in R_1 : \forall y \in R_2 : u(xy) \sqsubseteq v(xy)$, for $R_1, R_2 \in \mathsf{Reg}(\Sigma)$, have been shown undecidable [31].

$u(\tau') \sqsubseteq v(\tau')$ for all decompositions $\sigma = \tau.\tau'$. In other words, σ is both a direct and a codirect solution.

Note that $\mathsf{PEP}_{\text{co\&dir}}$ has no R' parameter (or, equivalently, has $R' = \Sigma^*$) and requires directness and codirectness at all positions. However, this restricted combination is already undecidable:

Theorem 5.19 (Undecidability)**.** $\mathsf{PEP}_{\text{co\&dir}}$ is Σ^0_1-complete.

Membership in Σ^0_1 is clear and we prove hardness by reducing from the Reachability Problem for length-preserving semi-Thue systems.

A semi-Thue system $S = (\Upsilon, \Delta)$ has a finite set $\Delta \subseteq \Upsilon^* \times \Upsilon^*$ of string rewrite rules over some finite alphabet Υ, written $\Delta = \{l_1 \to r_1, \ldots, l_k \to r_k\}$. The one-step rewrite relation $\to_\Delta \subseteq \Upsilon^* \times \Upsilon^*$ is defined as usual with $x \to_\Delta y \overset{\text{def}}{\Leftrightarrow} x = zlz'$ and $y = zrz'$ for some rule $l \to r$ in Δ and strings z, z' in Υ^*. We write $x \overset{m}{\to}_\Delta y$ and $x \overset{*}{\to}_\Delta y$ when x can be rewritten into y by a sequence of m (respectively, any number, possibly zero) rewrite steps.

The *Reachability Problem* for semi-Thue systems is "Given $S = (\Upsilon, \Delta)$ and two regular languages $P_1, P_2 \in \mathsf{Reg}(\Upsilon)$, is there $x \in P_1$ and $y \in P_2$ s.t. $x \overset{*}{\to}_\Delta y$?". It is well-known (or easy to see by encoding Turing machines in semi-Thue systems) that this problem is undecidable (in fact, Σ^0_1-complete) even when restricted to *length-preserving systems*, i.e., systems where $|l| = |r|$ for all rules $l \to r \in \Delta$.

We now construct a many-one reduction to $\mathsf{PEP}_{\text{co\&dir}}$. Let $S = (\Upsilon, \Delta)$, P_1, P_2 be a length-preserving instance of the Reachability Problem. W.l.o.g., we assume $\epsilon \notin P_1$ and we restrict to reachability via an even and non-zero number of rewrite steps. With any such instance we associate a $\mathsf{PEP}_{\text{co\&dir}}$ instance $u, v : \Sigma^* \to \Gamma^*$ with $R \in \mathsf{Reg}(\Sigma)$ such that the following Correctness Property holds:

$$\begin{aligned} & \exists x \in P_1, \exists y \in P_2, \exists m \text{ s.t. } x \overset{m}{\to}_\Delta y \text{ (and } m > 0 \text{ is even)} \\ \text{iff} \quad & \exists \sigma \in R \text{ s.t. } \sigma = \tau\tau' \text{ implies } u(\tau) \sqsubseteq v(\tau) \text{ and } u(\tau') \sqsubseteq v(\tau')\,. \end{aligned} \tag{CP}$$

The reduction uses letters like a, b and c taken from Υ, and adds † as an extra letter. We use six copies of each such "plain" letter. These copies are obtained by priming and double-priming letters, and by overlining. Hence the six copies of a are $a, a', a'', \overline{a}, \overline{a'}, \overline{a''}$. As expected, for a "plain" word (or alphabet) x, we write x' and $\overline{x}$ to denote a version of x obtained by priming (respectively, overlining) all its letters. Formally, letting $\Upsilon_\dagger$ being short for $\Upsilon \cup \{\dagger\}$, one has $\Sigma = \Gamma \overset{\text{def}}{=} \Upsilon_\dagger \cup \Upsilon'_\dagger \cup \Upsilon''_\dagger \cup \overline{\Upsilon_\dagger} \cup \overline{\Upsilon'_\dagger} \cup \overline{\Upsilon''_\dagger}$.

We define and explain the reduction by running it on the following example:

$$\Upsilon = \{a, b, c\} \text{ and } \Delta = \{ab \to bc,\ cc \to aa\}. \tag{S_{exmp}}$$

Assume that $abc \in P_1$ and $baa \in P_2$. Then $P_1 \xrightarrow{*}_\Delta P_2$ since $abc \xrightarrow{*}_\Delta baa$ as witnessed by the following (even-length) derivation π = "$abc \to_\Delta bcc \to_\Delta baa$". In our reduction, a rewrite step like "$abc \to_\Delta bcc$" appears in the PEP solution σ as the letter-by-letter interleaving $a\overline{b}b\overline{c}c\overline{c}$, denoted $abc|||bcc$, of a plain string and an overlined copy of a same-length string.

Write $T_\blacktriangleright(\Delta)$, or just $T_\blacktriangleright$ for short, for the set of all $x|||y$ such that $x \to_\Delta y$. Obviously, and since we are dealing with length-preserving systems, $T_\blacktriangleright$ is a regular language, as seen by writing it as $T_\blacktriangleright = \left(\sum_{a\in\Upsilon} a\overline{a}\right)^*.\{l|||r \mid l \to r \in \Delta\}.\left(\sum_{a\in\Upsilon} a\overline{a}\right)^*$, where $\{l|||r \mid l \to r \in \Delta\}$ is a finite, hence regular, language.

$T_\blacktriangleright$ accounts for odd-numbered steps. Symmetrically, for even-numbered steps like $bcc \to_\Delta baa$ in π above, we use $b\overline{b}a\overline{c}a\overline{c}$, i.e., $baa|||bcc$. Here too $T_\blacktriangleleft \stackrel{\text{def}}{=} \{y|||x \mid x \to_\Delta y\}$ is regular. Finally, a derivation π of the general form

$$x_0 \to_\Delta x_1 \to_\Delta x_2 \ldots \to_\Delta x_{2k},$$

where $K \stackrel{\text{def}}{=} |x_0| = \ldots = |x_{2k}|$, is encoded as a solution σ_π of the form

$$\sigma_\pi = \rho_0\sigma_1\rho_1\sigma_2\ldots\rho_{2k-1}\sigma_{2k}\rho_{2k}$$

that alternates between the encodings of steps (the σ_i's) in $T_\blacktriangleright \cup T_\blacktriangleleft$, and *fillers*, (the ρ_i's) defined as follows:

$$\sigma_i \stackrel{\text{def}}{=} \begin{cases} x_{i-1}|||x_i & \text{for odd } i\,, \\ x_i|||x_{i-1} & \text{for even } i\,, \end{cases}$$

$$\rho_0 \stackrel{\text{def}}{=} x''_0|||\dagger''^K\,, \qquad \rho_{2k} \stackrel{\text{def}}{=} x''_{2k}|||\dagger''^K\,,$$

$$\rho_i \stackrel{\text{def}}{=} \begin{cases} \dagger'^K|||x'_i & \text{for odd } i\,, \\ x'_i|||\dagger'^K & \text{for even } i \neq 0, 2k\,. \end{cases}$$

Note that the extremal fillers ρ_0 and ρ_{2k} use double-primed letters, when the internal fillers use primed letters. Continuing our example, the σ_π associated with the derivation $abc \to_\Delta bcc \to_\Delta baa$ is

$$\sigma_\pi = \underbrace{a''\overline{\dagger''}b''\overline{\dagger''}c''\overline{\dagger''}}_{a''b''c''|||\dagger''\dagger''\dagger''}\ \underbrace{a\overline{b}b\overline{c}c\overline{c}}_{abc|||bcc}\ \underbrace{\dagger'\overline{b'}\dagger'\overline{c'}\dagger'\overline{c'}}_{\dagger'\dagger'\dagger'|||b'c'c'}\ \underbrace{b\overline{b}a\overline{c}a\overline{c}}_{baa|||bcc}\ \underbrace{b''\overline{\dagger''}a''\overline{\dagger''}a''\overline{\dagger''}}_{b''a''a''|||\dagger''\dagger''\dagger''}.$$

The point with primed and double-primed copies is that u and v associate them with different images. Precisely, we define

$$\begin{array}{lllll} u(a) = a, & u(a') = \dagger, & u(\dagger') = \dagger, & u(a'') = \epsilon, & u(\dagger'') = \epsilon, \\ v(a) = \dagger, & v(a') = a, & v(\dagger') = w_\Upsilon, & v(a'') = a, & v(\dagger'') = w_\Upsilon, \end{array}$$

where a is any letter in Υ, and where w_Υ is a word listing all letters in Υ. E.g., $w_{\{a,b,c\}} = abc$ in our running example. The extremal fillers use special double-primed letters because we

want $u(\rho_0) = u(\rho_{2k}) = \epsilon$ (while v behaves the same on primed and double-primed letters). Finally, overlining is preserved by u and v: $u(\overline{x}) \stackrel{\text{def}}{=} \overline{u(x)}$ and $v(\overline{x}) \stackrel{\text{def}}{=} \overline{v(x)}$.

This ensures that, for $i > 0$, $u(\sigma_i) \sqsubseteq v(\rho_{i-1})$ and $u(\rho_i) \sqsubseteq v(\sigma_i)$, so that a σ_π constructed as above is a direct solution. It also ensures $u(\sigma_i) \sqsubseteq v(\rho_i)$ and $u(\rho_{i-1}) \sqsubseteq v(\sigma_i)$ for all $i > 0$, so that σ_π is also a codirect solution. One can check it on our running example by writing $u(\sigma_\pi)$ and $v(\sigma_\pi)$ alongside:

$$\begin{array}{rl}
\sigma_\pi = & \overbrace{a''\overline{\dagger''}b''\overline{\dagger''}c''\overline{\dagger''}}^{\rho_0}\ \overbrace{a\overline{b}b\overline{c}c\overline{c}}^{\sigma_1}\ \overbrace{\dagger'\overline{b'}\dagger'\overline{c'}\dagger'\overline{c'}}^{\rho_1}\ \overbrace{b\overline{b}a\overline{c}a\overline{c}}^{\sigma_2}\ \overbrace{b''\overline{\dagger''}a''\overline{\dagger''}a''\overline{\dagger''}}^{\rho_2} \\
\hline
u(\sigma_\pi) = & \qquad\qquad a\overline{b}b\overline{c}c\overline{c}\ \ \dagger\overline{\dagger}\dagger\overline{\dagger}\dagger\overline{\dagger}\ \ b\overline{b}a\overline{c}a\overline{c} \\
v(\sigma_\pi) = & a\,\overline{abc}\,b\,\overline{abc}\,c\,\overline{abc}\ \ \dagger\overline{\dagger}\dagger\overline{\dagger}\dagger\overline{\dagger}\ \ abc\,\overline{b}\,abc\,\overline{c}\,abc\,\overline{c}\ \ \dagger\overline{\dagger}\dagger\overline{\dagger}\dagger\overline{\dagger}\ \ b\,\overline{abc}\,a\,\overline{abc}\,a\,\overline{abc}
\end{array}$$

There remains to define R. Since $\rho_0 \in \left(\Upsilon''\overline{\dagger''}\right)^+$, since $\sigma_i \in T_\blacktriangleright$ for odd i, etc., we let

$$R \stackrel{\text{def}}{=} \left(\Upsilon''\overline{\dagger''}\right)^+ . T_\blacktriangleright^{\cap P_1} . \left(\dagger'\overline{\Upsilon'}\right)^+ . \left(T_\blacktriangleleft . \left(\Upsilon'\overline{\dagger'}\right)^+ . T_\blacktriangleright . \left(\dagger'\overline{\Upsilon'}\right)^+\right)^* . T_\blacktriangleleft^{\cap P_2} . \left(\Upsilon''\overline{\dagger''}\right)^+,$$

where $T_\blacktriangleright^{\cap P_1} \stackrel{\text{def}}{=} \{x \,|||\, y \mid x \rightarrow_\Delta y \wedge x \in P_1\} = T_\blacktriangleright \cap \{x \,|||\, y \mid x \in P_1 \wedge |x| = |y|\}$ is clearly regular when P_1 is, and similarly for $T_\blacktriangleleft^{\cap P_2} \stackrel{\text{def}}{=} \{y \,|||\, x \mid x \rightarrow_\Delta y \wedge y \in P_2\}$. Since $\sigma_\pi \in R$ when π is an even-length derivation from P_1 to P_2, we deduce that the left-to-right implication in (CP) holds.

We now prove the right-to-left implication, which concludes the proof of Theorem 5.19.

Assume that there is a $\sigma \in R$ such that $u(\tau) \sqsubseteq v(\tau)$ and $u(\tau') \sqsubseteq v(\tau')$ for all decompositions $\sigma = \tau\tau'$. By definition of R, σ must be of the form

$$\sigma = \rho_0\sigma_1\rho_1(\sigma_2\rho_2\sigma_3\rho_3)\ldots(\ldots\sigma_{2k-1}\rho_{2k-1})\sigma_{2k}\rho_{2k}$$

for some $k > 0$, with $\rho_0 \in \left(\Upsilon''\overline{\dagger''}\right)^+$, with $\sigma_i \in T_\blacktriangleright$ for odd i and $\sigma_i \in T_\blacktriangleleft$ for even i, etc. These $4k+1$ non-empty factors, $(\sigma_i)_{1\leq i\leq 2k}$ and $(\rho_i)_{0\leq i\leq 2k}$, are called the "*segments*" of σ, and numbered $s_0, \ldots, s_{4k}$ in order.

Lemma 5.20. *$u(s_p) \sqsubseteq v(s_{p-1})$ and $u(s_{p-1}) \sqsubseteq v(s_p)$ for all $p = 1, \ldots, 4k$.*

Proof. First note that the definition of u and v ensures that $u(s_p)$ and $v(s_p)$ use disjoint alphabets. More precisely, all $u(\sigma_i)$'s and $v(\rho_i)$'s are in $(\Upsilon\overline{\Upsilon})^*$, while the $v(\sigma_i)$'s and the $u(\rho_i)$'s are in $(\dagger\overline{\dagger})^*$, with the special case that $u(\rho_0) = u(\rho_{2k}) = \epsilon$ since ρ_0 and ρ_{2k} are made of double-primed letters.

Since σ is a direct solution, $u(s_0 \ldots s_p) \sqsubseteq v(s_0 \ldots s_p)$ for any p, and even

$$u(s_0 \ldots s_p) \sqsubseteq v(s_0 \ldots s_{p-1}), \tag{A_p}$$

since $v(s_p)$ has no letter in common with $u(s_p)$. We now claim that, for all $p = 1, \ldots, 4k$

$$u(s_0 s_1 \ldots s_p) \not\sqsubseteq v(s_0 s_1 \ldots s_{p-2}), \tag{B_p}$$

as we prove by induction on p. For the base case, $p = 1$, the claim is just the obvious $u(s_0 s_1) \not\sqsubseteq \epsilon$. For the inductive case $p > 1$, one combines $u(s_0 \ldots s_{p-1}) \not\sqsubseteq v(s_0 \ldots s_{p-3})$ (ind. hyp.) with $u(s_p) \not\sqsubseteq v(s_{p-2})$ (different alphabets) and gets $u(s_0 \ldots s_p) \not\sqsubseteq v(s_0 \ldots s_{p-2})$.

We now combine (A_p), i.e., $u(s_0 \ldots s_p) \sqsubseteq v(s_0 \ldots s_{p-1})$, and (B_{p-1}), i.e., $u(s_0 s_1 \ldots s_{p-1}) \not\sqsubseteq v(s_0 s_1 \ldots s_{p-3})$, yielding $u(s_p) \sqsubseteq v(s_{p-2} s_{p-1})$, hence $u(s_p) \sqsubseteq v(s_{p-1})$ since $u(s_p)$ and $v(s_{p-2})$ share no letter: we have proved one half of the Lemma. The other half is proved symmetrically, using the fact that σ is also a codirect solution. □

Lemma 5.21. $|s_1| = |s_2| = \ldots = |s_{4k-1}|$.

Proof. For any p with $0 < p < 4k$, $u(s_p) \sqsubseteq v(s_{p-1})$ (Lemma 5.20) implies $|s_p| \leq |s_{p-1}|$, as can easily be seen either when s_p is some $x \,|||\, y$ or when s_p is some filler like $\dagger'^{L} \,|||\, x'$. Thus $|s_0| \geq |s_1| \geq \cdots \geq |s_{4k-1}|$. Similarly, the other half of Lemma 5.20, i.e., $u(s_{p-1}) \sqsubseteq v(s_p)$, entails $|s_1| \leq |s_2| \leq \cdots \leq |s_{4k}|$. □

Now pick any $i \in \{1, \ldots, 2k\}$. If i is odd, then by definition of R, $\sigma_i \in T_{\blacktriangleright}$ is some $x_{i-1} \,|||\, y_i$ with $x_{i-1} \rightarrow_\Delta y_i$ and $\sigma_{i+1} \in T_{\blacktriangleleft}$ is some $y_{i+1} \,|||\, x_i$ with $x_i \rightarrow_\Delta y_{i+1}$. Furthermore, ρ_i is some $\dagger'^{|z_i|} \,|||\, z'_i$. With Lemma 5.20, we deduce $y_i \sqsubseteq z_i$ and $x_i \sqsubseteq z_i$. With Lemma 5.21, we further deduce $|y_i| = |z_i| = |x_i|$, hence $y_i = x_i$. A similar reasoning shows that $y_i = x_i$ also holds when i is even, so that the steps $x_{i-1} \rightarrow_\Delta y_i$ can be chained. Finally, we deduce from σ the existence of a derivation $x_0 \rightarrow_\Delta x_1 \rightarrow_\Delta \cdots \rightarrow_\Delta x_{2k}$. Since $\sigma_0 \in T_{\blacktriangleright}^{\cap P_1}$ and $\sigma_{2k} \in T_{\blacktriangleleft}^{\cap P_2}$, we further deduce $x_0 \in P_1$ and $x_{2k} \in P_2$. Hence the existence of σ entails $P_1 \xrightarrow{2k}_\Delta P_2$, which concludes the proof.

5.6 Complexity

Complexity-wise, PEP and $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ lie at level $\mathsf{F}_{\omega^\omega}$ of the fast-growing hierarchy [76, 30]. Further, PEP with an alphabet size of $k+2$ is F_{ω^k}-hard and in $\mathsf{F}_{\omega^{k+1}+1}$. For the lower bound, one encodes computations of a F_{ω^ω}-space-bounded Turing machine (or more simply, semi-Thue system) as a rational relation in a way that is robust to losses, which then is easily translated to PEP. This requires an elaborate encoding of ordinals upto ω^{ω^k} which is robust to losses. See [76] for details. For the upper bound, one uses Theorem 5.4, and the bounds on the Length Function H provided in [112].

5.7 Concluding remarks

We introduced partial directness in Post Embedding Problems and proved the decidability of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ by showing that an instance has a solution if, and only if, it has a solution of length bounded by a computable function of the input.

This generalizes and simplifies earlier proofs for PEP and $\mathsf{PEP}_{\text{dir}}$. The added generality is non-trivial and leads to decidability for UCST, or UCS (that is, unidirectional channel systems) extended with tests (chapter 6). The simplification lets us deal smoothly with counting or universal versions of the problem. Finally, we showed that *combining* directness and codirectness constraints leads to undecidability.

Chapter 6

Unidirectional channel systems with tests

Unidirectional channel systems, "UCSes" for short, are channel systems where a Sender process communicates to a Receiver process via *one reliable* and *one lossy* channel, see Fig. 6.1.

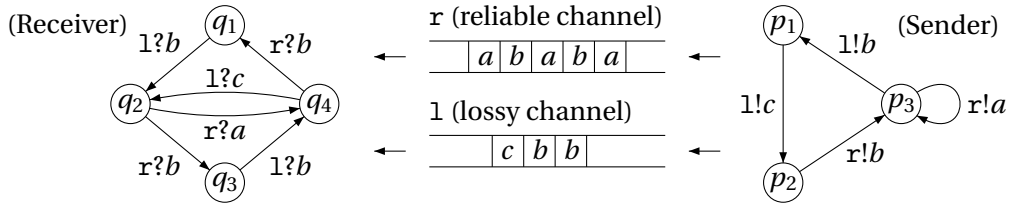


Figure 6.1: UCS = buffered one-way communication via one reliable and one lossy channels

UCSes are limited to one-way communication: there are no channels going from Receiver to Sender. One-way communication appears, e.g., in half-duplex protocols [67] or in the acyclic networks of [87, 13].

UCSes were introduced by Chambart and Schnoebelen [28] who identified them as a minimal setting to which one can reduce reachability problems for more complex combinations of lossy and reliable channels. The reachability problem for UCSes is quite challenging: it was proved decidable by reformulating it more abstractly as PEP, the (Regular) Post Embedding Problem, which is easier to analyze (chapter 5). While PEP is a natural variant of Post's Correspondence Problem, it was only identified through questions on UCSes.

Testing channel contents

Basic channel machines are not allowed to inspect the contents of the channels. However, it is sometimes useful to enrich the basic setup with tests. For example, a multiplexer pro-

cess will check each of its input channels in turn and will rely on emptiness and/or non-emptiness tests to ensure that this round robin policy does not block when one input channel is empty [107]. In other settings, channel systems with insertion errors becomes more expressive when emptiness tests are allowed [23].

In this chapter we consider such emptiness and non-emptiness tests, as well as more general tests given by arbitrary regular predicates on channel contents. A simple example is given below in Figure 6.2 (see page 78) where some of Sender's actions depend on the parity of the number of messages currently in the reliable channel `r`. When verifying plain UCSes, one can reorder steps and assume a two-phase behaviour where all Sender steps occur before all Receiver steps. When one has tests, one can no longer assume this.

Our contribution

We extend UCSes with the possibility of testing channel contents with regular predicates (Section 6.1). This makes reachability undecidable even with restricted sets of simple tests (section 6.2). Our main result (Theorem 6.4) is that reachability is decidable for UCSes extended with emptiness and non-emptiness tests. The proof goes through a series of reductions, some of them nontrivial, that leave us with UCSes extended by only emptiness tests on a single side of a single channel, called "Z_1^1 tests" (sections 6.4 and 6.5). This minimal extension is then reduced (Section 6.6) to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, or "PEP with partial codirectness", a nontrivial extension of PEP that is proved decidable in chapter 5. This last reduction extends the reduction from UCS to PEP in [29]. Finally, Section 6.7 proves that emptiness and/or non-emptiness tests strictly enrich the basic UCS model.

Related work.

Emptiness and non-emptiness tests have been considered already in [107], while Promela (SPIN's input language) offers head tests (that test the first available message without consuming it) [63]. Beyond such *specific* tests, we are not aware of results that consider models with a general notion of tests on channel contents (except in the case of LCSes where very general tests can be allowed without compromising the main decidability results, see [18, sect. 6]).

Regarding unidirectional channels, the decidability results in [13, 87, 59, 58, 35] apply to systems where communication between two agents is limited to a *single* one-way channel (sometimes complemented with a finite shared memory, real-time clock, integer-valued counter, or local pushdown stack).

6.1 Preliminaries

6.1.1 Syntax

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

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

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

In graphical representations like Fig. 6.1, Sender and Receiver are depicted as two disjoint directed graphs, where states appear as nodes and where rules $q \xrightarrow{\mathtt{c},\alpha} q'$ appear as edges from q to q' with the corresponding labellings.

6.1.2 Operational Semantics

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

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

states $q = q_1$ and $q' = q_1'$ and $q_2 = q_2'$ (for Sender rules), or $q = q_2$ and $q' = q_2'$ and $q_1 = q_1'$ (for Receiver rules);

tests if δ is a test rule $q \xrightarrow{\mathtt{c}:R} q'$, then $\mathtt{c} = \mathtt{r}$ and $u \in R$, or $\mathtt{c} = \mathtt{l}$ and $v \in R$, and furthermore $u' = u$ and $v' = v$;

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

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

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

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

Remark 6.1 (On reliable steps)**.** As is usual with unreliable channel systems, the reliable semantics plays a key role even though the object of our study is reachability via not necessarily reliable steps. First it is a normative yardstick from which one defines the unreliable semantics by extension. Then many hardness results on lossy systems are proved via reductions where a lossy system simulates in some way the reliable (and Turing-powerful) behaviour: proving the correctness of such reductions requires having the concept of reliable steps.

Remark 6.2 (UCSTs and well-structured systems)**.** It is well-known that $(\mathtt{M}^*, \sqsubseteq)$ is a well-quasi-order (a wqo): any infinite sequence $v_0, v_1, v_2, \ldots$ of words over $\mathtt{M}$ contains an infinite increasing subsequence $v_{i_0} \sqsubseteq v_{i_1} \sqsubseteq v_{i_2} \sqsubseteq \cdots$ This classic result, called Higman's Lemma, plays a fundamental role in the algorithmic verification of lossy channel systems and other well-structured systems [48, 113]. However, we note that $(Conf, \sqsubseteq)$ is *not* a wqo since $C \sqsubseteq D$ requires equality on channel $\mathtt{r}$, so that UCSTs are not well-structured systems despite the presence of a lossy channel.

6.1.3 Reachability

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

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

Since U, V, U', V' can be taken as singleton sets, the G-G-Reach problem is more general than asking whether S has a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ for some given initial and final configurations. We shall need the added generality in section 6.5 in particular. However, sometimes we will also need to put restrictions on U, V, U', V'. We use E-G-Reach to denote the reachability problem where $U = V = \{\epsilon\}$, i.e., where C_{in} has empty channels (E is for "Empty"), while $U', V' \in \mathsf{Reg}(\mathtt{M})$ are not constrained. We will also consider the E-E-Reach restriction where $U = V = U' = V' =$

$\{\epsilon\}$. It is known —see [28, Theo 3.1]— that E-E-Reach is decidable for UCSes, i.e., UCSTs that do not use tests.

6.2 Testing channels and the undecidability of reachability

Despite their similarities, UCSes and LCSes (lossy channel systems) behave differently. The algorithms deciding reachability for LCSes can easily accommodate regular (or even more expressive) tests [18, Sect. 6]. By contrast, UCSes become Turing-powerful when equipped with regular tests. The main result of this section is the undecidability of reachability for UCSTs. To state the respective theorem in a stronger version, we first introduce a notation for restricting the (regular) tests.

6.2.1 Restricted sets of tests

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

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

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

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

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

Theorem 6.3. Reachability (E-E-Reach) is undecidable for both UCST$[\mathscr{P}_1^{\mathtt{r}}]$ and UCST$[\mathscr{H}_1^{\mathtt{r}}]$.

We now proceed to prove Theorem 6.3 by simulating queue automata with UCSTs.

6.2.2 Simulating queue automata

Like queue automata, UCSes have a reliable channel but, unlike them, Sender (or Receiver) cannot both read *and* write from/to it. If Sender could somehow read from the head of r, it would be as powerful as a queue automaton, i.e., Turing-powerful. Now we show that parity tests used by Sender on r allow us to construct a simple protocol making Receiver act as a proxy for Sender and implement read actions on its behalf. See Figure 6.2 for an illustrating example of how Sender simulates a rule $p_1 \xrightarrow{\mathtt{r}?a} p_2$.

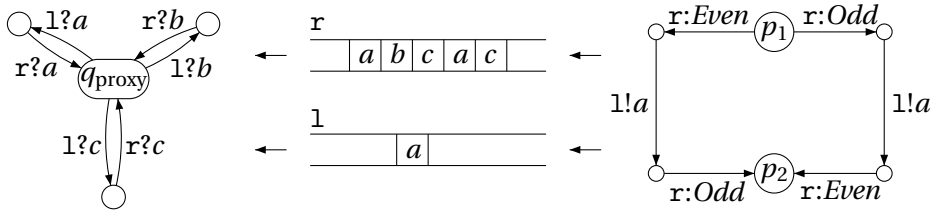


Figure 6.2: Sender simulates "$p_1 \xrightarrow{\mathtt{r}?a} p_2$" with parity tests and proxy Receiver

Described informally, the protocol is the following:

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

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

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

To prove undecidability for UCST[$\mathscr{H}_1^{\mathtt{r}}$] we just modify the previous protocol. We use two copies of the message alphabet, e.g., using two "colours". When writing on r, Sender strictly

alternates between the two colours. If now Sender wants to read a given letter, say a, from $\mathtt{r}$, it checks that an a (of the right colour) is present at the head of $\mathtt{r}$ by using $\mathscr{H}_1^{\mathtt{r}}$ tests. It then asks Receiver to read a by sending a message via $\mathtt{l}$. Since colours alternate in $\mathtt{r}$, Sender can check (i.e., wait until), again via head tests, that the reading of a occurred .

6.3 Main theorem and a roadmap for its proof

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

Theorem 6.4. Reachability (G-G-Reach) is decidable for UCST[Z,N].

Hence adding emptiness and nonemptiness tests to UCSes still maintains the decidability of reachability (unlike what happens with parity or head tests).

Our proof of Theorem 6.4 is quite long, being composed of several consecutive reductions, some of which are nontrivial. A scheme of the proof is depicted in Figure 6.3, and we give a brief outline in the rest of this section.

We first recall that the reachability problem for UCSes (i.e., for UCST[$\emptyset$]) was shown decidable via a reduction to PEP (Post's Embedding Problem) in [29]. Relying on this earlier result (by reducing UCST[Z,N] to UCST[$\emptyset$]) or extending its proof (by reducing UCST[Z,N] to PEP directly) does not seem at all trivial. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ from chapter 5 is used as an intermediate step instead.

Having the decidability of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$, our proof for Theorem 6.4 is composed of two main parts:

1. One part, given in Section 6.6, is a reduction of E-E-Reach for UCST[Z_1^1] to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$. It is relatively compact, since we have found a suitable intermediate notion between runs of UCST[Z_1^1] and solutions of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$.

2. The other part, given in sections 6.4 and 6.5, reduces G-G-Reach for UCST[Z,N] to E-E-Reach for UCST[Z_1^1]. It has turned out necessary to decompose this reduction in a series of smaller steps (as depicted in Figure 6.3) where features such as certain kinds of tests, or general initial and final conditions, are eliminated step by step. The particular way in which these features are eliminated is important. For example, we eliminate Z_2 and N_2 tests by one simulation reducing G-G-Reach[Z, N] to G-G-Reach[Z_1, N_1] (Sec. 6.4.2); the simulation would not work if we wanted to eliminate Z_2 and N_2 separately, one after the other.

One of the crucial steps in our series is the reduction from E-E-Reach[Z_1] to G-G-Reach[Z_1^1]. This is a Turing reduction, while we otherwise use logspace many-one reductions. Even

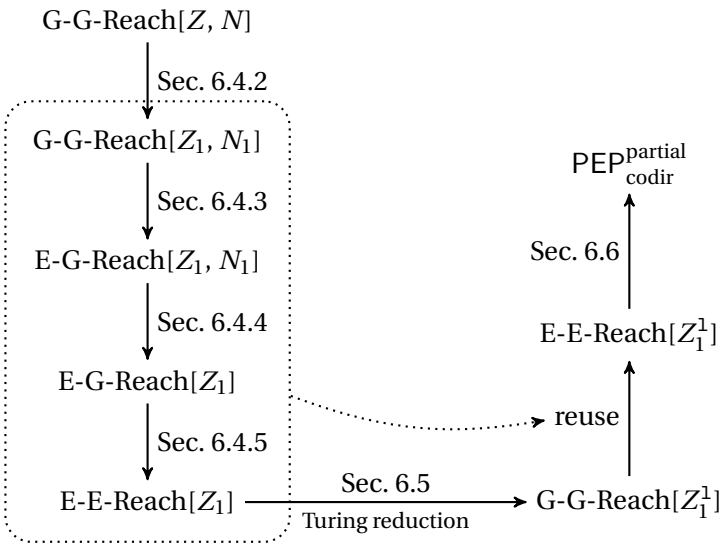


Figure 6.3: Roadmap of the reductions from G-G-Reach[Z, N] to PEP$^{\text{partial}}_{\text{codir}}$

though we start with a problem instance where the initial and final configurations have empty channel contents, we need oracle calls to a problem where the initial and final conditions are more general. This alone naturally leads to considering the G-G-Reach instances.

It seems also worth noting that all reductions in Section 6.4 treat the two channels in the same way; no special arrangements are needed to handle the lossiness of 1. The proofs of correctness, of course, do need to take the lossiness into account.

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

This section describes four simulations that, put together, entail Point 1 in Theorem 6.5 below. Moreover, the last three simulations also yield Point 2. We note that the simulations are tailored to the reachability problem: they may not preserve other behavioural aspects like, e.g., termination or deadlock-freedom.

Theorem 6.5.
(1) G-G-Reach[Z, N] many-one reduces to E-E-Reach[Z_1].
(2) G-G-Reach[Z_1^1] many-one reduces to E-E-Reach[Z_1^1].

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

6.4.1 Commuting steps in UCST[Z, N] systems

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

Lemma 6.6 (Commutation)**.** Two consecutive steps $C \xrightarrow{\delta_1} C' \xrightarrow{\delta_2} C''$ commute in any of the following cases:

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

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

We will use Lemma 6.6 several times and in different ways. For the time being, we consider in particular the convenient restriction to "head-lossy" runs. Formally, a message loss $C \xrightarrow{\text{los}} C'$ is *head-lossy* if it is of the form $(p, q, u, av) \xrightarrow{\text{los}} (p, q, u, v)$ where $a \in \mathtt{M}$ (i.e., the lost message was the head of $\mathtt{l}$). A *run* $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ is *head-lossy* if all its message loss steps are head-lossy, or occur only after all the reliable steps in the run (it is convenient to allow unconstrained losses at the end of the run). Repeated use of Point (2) in Lemma 6.6 easily yields the next corollary:

Corollary 6.7. If there is a run from C_{in} to C_{fi} then there is a head-lossy run from C_{in} to C_{fi}.

6.4.2 Reducing G-G-Reach[Z, N] to G-G-Reach[Z_1, N_1]

Our first reduction eliminates Z and N tests by Receiver. These tests are replaced by reading two special new messages, "z" and "n", that Sender previously put in the channels.

Formally, we consider an instance of G-G-Reach[Z, N], made of a given UCST $S = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, given states $p_{\text{in}}, p_{\text{fi}} \in Q_1$, $q_{\text{in}}, q_{\text{fi}} \in Q_2$, and given languages $U, V, U', V' \in \mathsf{Reg}(\mathtt{M})$. We construct a new UCST S' from S as follows (see Fig. 6.4):

1. We add two special new messages z, n to $\mathtt{M}$, thus creating the alphabet $\mathtt{M}' \stackrel{\text{def}}{=} \mathtt{M} \uplus \{n, z\}$.

2. For each channel $\mathtt{c} \in \{\mathtt{r},\mathtt{l}\}$ and each Sender's state $p \in Q_1$ we add new states $p^1_\mathtt{c}$, $p^2_\mathtt{c}$ and an "*(emptiness) testing loop*" $p \xrightarrow{\mathtt{c}:Z} p^1_\mathtt{c} \xrightarrow{\mathtt{c}!z} p^2_\mathtt{c} \xrightarrow{\mathtt{c}:Z} p$ (i.e., three new rules).

3. For every Sender's writing rule θ of the form $p \xrightarrow{\mathtt{c}!x} p'$ we add a new state p_θ and the following three rules: $p \xrightarrow{\top} p_\theta$, $p_\theta \xrightarrow{\mathtt{c}!n} p_\theta$ (a "*padding loop*"), and $p_\theta \xrightarrow{\mathtt{c}!x} p'$.

4. For every Receiver's rule $q \xrightarrow{\mathtt{c}:Z} q'$ (testing emptiness of $\mathtt{c}$) we add the rule $q \xrightarrow{\mathtt{c}?z} q'$.

5. For every Receiver's rule $q \xrightarrow{\mathtt{c}:N} q''$ (testing non-emptiness of $\mathtt{c}$) we add the rule $q \xrightarrow{\mathtt{c}?n} q''$.

6. At this stage, the resulting system is called S_{aux}.

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

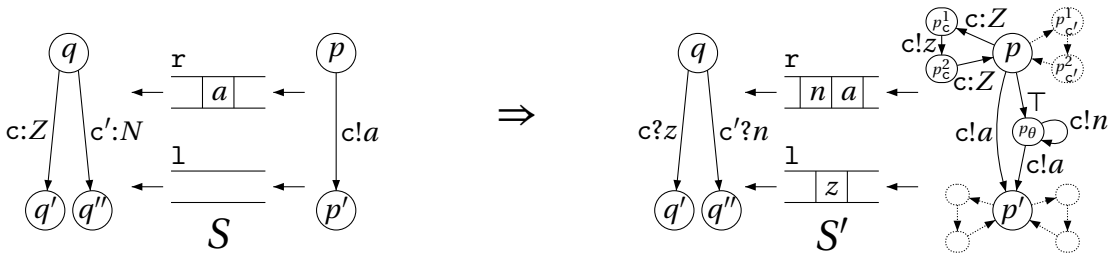


Figure 6.4: Reducing G-G-Reach$[Z,N]$ to G-G-Reach$[Z_1,N_1]$: eliminating Receiver's tests

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

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

By replacing S, U, V with S', $pad(U)$, $pad(V)$ (and keeping p_{in}, p_{fi}, q_{in}, q_{fi}, U', V' unchanged), the initial G-G-Reach$[Z,N]$ instance is transformed into a G-G-Reach$[Z_1,N_1]$ instance. The correctness of this reduction is captured by the next lemma, that we immediately proceed to prove in the rest of section 6.4.2:

Lemma 6.8. For any $u, v, u', v' \in \mathtt{M}^*$, S has a run $(p_{\text{in}}, q_{\text{in}}, u, v) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ if, and only if, S' has a run $(p_{\text{in}}, q_{\text{in}}, \hat{u}, \hat{v}) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ for some padded words $\hat{u} \in pad(u)$ and $\hat{v} \in pad(v)$.

Though we are ultimately interested in S and S', it is convenient to consider special runs of S_{aux} since S_{aux} "contains" both S and S'. We rely on Corollary 6.7 and tacitly assume that all runs are head-lossy. We say that a (head-lossy) *run* $C_0 \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} \cdots \xrightarrow{\delta_n} C_n$ of S_{aux} is *faithful* if $C_0 = (p_0, q_0, u_0, v_0)$ with $u_0, v_0 \in pad(\mathtt{M}^*)$, $C_n = (p_n, q_n, u_n, v_n)$ with $u_n, v_n \in \mathtt{M}^*$, $p_0, p_n \in Q_1$, $q_0, q_n \in Q_2$, and the following two properties are satisfied (for all $i = 1, 2, \ldots, n$):

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

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

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

We now exhibit two reversible transformations of runs of S_{aux}, one for Z tests in §6.4.2, the other for N tests in §6.4.2, that preserve faithfulness. This will allow us to translate runs of S, witnessing the original instance, to faithful runs of S', witnessing the created instance, and vice versa. Finally we show in §6.4.2 that if there is a run of S' witnessing the created instance, then there is a faithful one as well.

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

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

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

Trading Z_2 tests for P1-segments.

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

$$(p, q, w, \varepsilon) \xrightarrow{\mathtt{l}:Z} (p^1_1, q, w, \varepsilon) \xrightarrow{\mathtt{l}!z} (p^2_1, q, w, z) \xrightarrow{\mathtt{l}?z} (p^2_1, q', w, \varepsilon) \xrightarrow{\mathtt{l}:Z} (p, q', w, \varepsilon) \qquad (6.1)$$

using the rules introduced in S_{aux}. This transforms (the faithful run) π into another faithful run π', decreasing the number of Receiver's tests (by one occurrence of a Z_2-test). In the other direction, if π contains a P1-segment $C_{m-1} \xrightarrow{*} C_{m+3}$, it must be of the form (6.1), when the involved channel is $\mathtt{c} = \mathtt{1}$, and we can replace it with one step $C_{m-1} \xrightarrow{\mathtt{c}:Z} C_{m+3}$, preserving faithfulness.

Trading N_2 tests for occurrences of n.

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

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

If $k = 0$ (i.e., if x is present at the beginning of π), we have exhibited a faithful run $C_0' \xrightarrow{*} C_m' \xrightarrow{\mathtt{1}?n} C_{m+1} \xrightarrow{*} C_n$, starting from $C_0' = (p_{\text{in}},q_{\text{in}},u_0,w_0\,n\,x\,w_0')$, where $w_0\,n\,x\,w_0' \in pad(v_0)$ since $v_0 = w_0\,x\,w_0'$. If $k > 0$, the highlighted occurrence of x necessarily appears in C_k via $\delta_k = p_{k-1} \xrightarrow{\mathtt{1}!x} p_k$ and we have $v_k = v_{k-1}x$. If δ_k is a rule of S, we may exhibit a sequence $C_{k-1} \xrightarrow{*} C_k'$ using the new rules

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

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

For the backward transformation we assume that n occurs in a configuration of π. We select one such occurrence and let $C_k, C_{k+1}, \ldots, C_m$ ($0 \le k \le m < n$) be the part of π where this occurrence of n appears. For $i = k,k+1,\ldots,m$, we highlight this occurrence of n by writing v_i in the form $w_i\,n\,w_i'$ (assuming without loss of generality that the n occurs in $\mathtt{1}$), i.e., we write $C_i = (p_i,q_i,u_i,w_i\,n\,w_i')$. Removing the n yields new configurations $C_k', C_{k+1}', \ldots, C_m'$ given by $C_i' = (p_i,q_i,u_i,w_i\,w_i')$.

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

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

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

Handling S' runs and faithfulness.

Since a witness run of S is (trivially) faithful, the above transformations allow us to remove one by one all occurrences of Receiver's Z and N tests, creating a (faithful) witness run for S' (with a possibly padded C_0). We have thus proved the "only-if" part of Lemma 6.8. The "if" part is shown analogously, now using the two transformations in the other direction and removing occurrences of the new z and n messages, *with one proviso*: we only transform faithful runs. We thus need to show that if S' has a (head-lossy) run $(p_{\text{in}}, q_{\text{in}}, \hat{u}, \hat{v}) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ then it also has a faithful one.

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

Assume that some rule $\delta_i = p \xrightarrow{\top} p_\theta$ is used in π, and that P2 fails on this occurrence of δ_i. Since π does not end in state p_θ, Sender necessarily continues with some (possibly zero) $p_\theta \xrightarrow{\mathtt{c}!n} p_\theta$ steps, followed by some $\delta_j = p_\theta \xrightarrow{\mathtt{c}!x} p'$. Now all Receiver or message loss steps between δ_i and δ_j can be swapped and postponed after δ_j since Receiver has no tests and Sender does not test between δ_i and δ_j (recall Lemma 6.6(3)). After the transformation, δ_i and the rules after it form a P2-segment. Also, since message losses have been postponed, the run remains head-lossy.

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

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

6.4.3 Reducing G-G-Reach$[Z_1, N_1]$ to E-G-Reach$[Z_1, N_1]$

A G-G-Reach$[Z_1, N_1]$ instance where the initial contents of $\mathtt{r}$ and $\mathtt{l}$ are restricted to (regular languages) U and V respectively can be transformed into an equivalent instance where U and V are both replaced with $\{\epsilon\}$. For this, one adds a new (fresh) initial state to Sender, from which Sender first generates a word in U nondeterministically, writing it on $\mathtt{r}$, then generates a word in V, writing it on $\mathtt{l}$, and then enters the original initial state.

Stating the correctness of this reduction has the form

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

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

6.4.4 Reducing E-G-Reach$[Z_1, N_1]$ to E-G-Reach$[Z_1]$

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

Formally, we start with some instance $(S, p_{\mathrm{in}}, p_{\mathrm{fi}}, q_{\mathrm{in}}, q_{\mathrm{fi}}, \{\epsilon\}, \{\epsilon\}, U', V')$ of E-G-Reach$[Z_1, N_1]$, where $S = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, and we create $S' = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1', \Delta_1', Q_2, \Delta_2)$ arising from S as follows (see Fig. 6.5). We put $Q_1' = Q_1 \times (\mathtt{M} \cup \{\epsilon\}) \times (\mathtt{M} \cup \{\epsilon\})$; the components x, y in a state $\langle q, x, y \rangle$ denote the contents of the buffers for $\mathtt{r}$ and $\mathtt{l}$, respectively. We now replace each rule $q \xrightarrow{\mathtt{r}!x} q'$ with $\langle q, \epsilon, y \rangle \xrightarrow{\top} \langle q', x, y \rangle$ for all $y \in \mathtt{M} \cup \{\epsilon\}$ (Fig. 6.5 uses "$\top_!$" to highlight these transformed rules). Each $q \xrightarrow{\mathtt{r}:N} q'$ is replaced with $\langle q, x, y \rangle \xrightarrow{\top} \langle q', x, y \rangle$ for all x, y where $x \neq \epsilon$ (Fig. 6.5 uses "$\top_N$"). Each $q \xrightarrow{\mathtt{r}:Z} q'$ is replaced with $\langle q, \epsilon, y \rangle \xrightarrow{\mathtt{r}:Z} \langle q', \epsilon, y \rangle$ (for all y). Analogously we replace

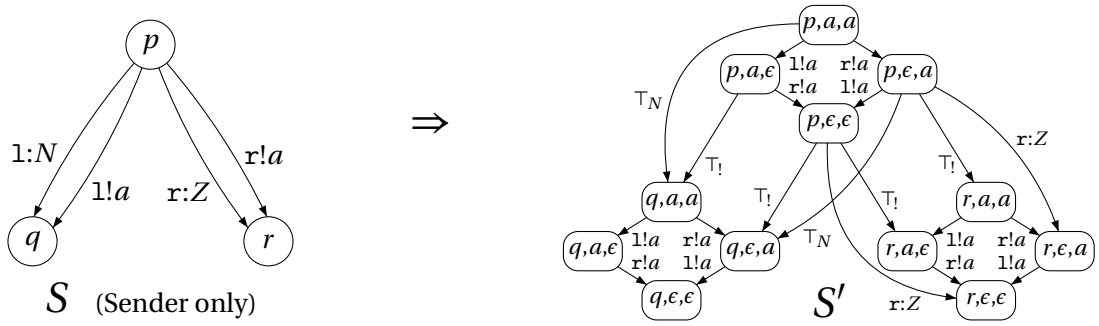


Figure 6.5: Reducing E-G-Reach$[Z_1, N_1]$ to E-G-Reach$[Z_1]$

all $q \xrightarrow{\mathtt{l}!x} q'$, $q \xrightarrow{\mathtt{l}:N} q'$, and $q \xrightarrow{\mathtt{l}:Z} q'$. Moreover, we add the rules $\langle q,x,y\rangle \xrightarrow{\mathtt{r}!x} \langle q,\epsilon,y\rangle$ (for $x \neq \epsilon$) and $\langle q,x,y\rangle \xrightarrow{\mathtt{l}!y} \langle q,x,\epsilon\rangle$ (for $y \neq \epsilon$). Our desired reduction is completed, by the next lemma:

Lemma 6.9. *S has a run $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, \epsilon, \epsilon) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v') = C_{\text{fi}}$ if, and only if, S' has a run $C'_{\text{in}} = (\langle p_{\text{in}}, \epsilon, \epsilon\rangle, \langle q_{\text{in}}, \epsilon, \epsilon\rangle, \epsilon, \epsilon) \xrightarrow{*} (\langle p_{\text{fi}}, \epsilon, \epsilon\rangle, \langle q_{\text{fi}}, \epsilon, \epsilon\rangle, u', v') = C'_{\text{fi}}$.*

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

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

6.4.5 Reducing E-G-Reach$[Z_1]$ to E-E-Reach$[Z_1]$

The idea of the reduction is similar to what was done in section 6.4.3. The regular final conditions "$u' \in U'$" and "$v' \in V'$" are checked by Receiver consuming the final channel contents. When Sender (guesses that it) is about to write the first message that will be part of the final

u' in r (respectively, the final v' in l), it signals this by inserting a special symbol # just before. After it has written # to a channel, Sender is not allowed to test that channel anymore.

Formally we start with an instance $(S, p_{\text{in}}, p_{\text{fi}}, q_{\text{in}}, q_{\text{fi}}, \{\epsilon\}, \{\epsilon\}, U', V')$ of E-G-Reach$[Z_1]$, where $S = (\{\mathtt{r},\mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$. With S we associate S' where $\mathtt{M}' = \mathtt{M} \uplus \{\#\}$, as sketched in Fig. 6.6. This yields the instance $(S', p'_{\text{in}}, p'_{\text{fi}}, q_{\text{in}}, q_f, \{\epsilon\}, \{\epsilon\}, \{\epsilon\}, \{\epsilon\})$ of E-E-Reach$[Z_1]$, for the new final Receiver state q_f.

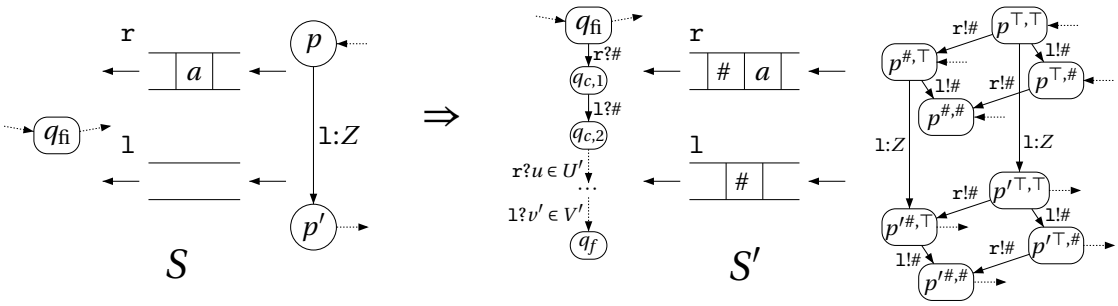


Figure 6.6: Reducing E-G-Reach$[Z_1]$ to E-E-Reach$[Z_1]$

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

The Sender part Q'_1, Δ'_1 of S' is obtained from Q_1, Δ_1 as follows. We put $Q'_1 \stackrel{\text{def}}{=} Q_1 \times \{\top, \#\} \times \{\top, \#\}$, and $p'_{\text{in}} = \langle p_{\text{in}}, \top, \top \rangle$, $p'_{\text{fi}} = \langle p_{\text{fi}}, \#, \# \rangle$. A state $\langle p, x, y \rangle$ "remembers" if # has been already written to r ($x = \#$) or not ($x = \top$); similarly for l (by $y = \#$ or $y = \top$). For changing the status (just once for each channel), Δ'_1 contains the rules $\langle p, \top, y \rangle \xrightarrow{\mathtt{r}!\#} \langle p, \#, y \rangle$ and $\langle p, x, \top \rangle \xrightarrow{\mathtt{l}!\#} \langle p, x, \# \rangle$ for each $p \in Q_1$ and $x, y \in \{\top, \#\}$. Moreover, any rule $p \xrightarrow{\mathtt{c},\alpha} p'$ in Δ_1 induces the rules $\langle p, x, y \rangle \xrightarrow{\mathtt{c},\alpha} \langle p', x, y \rangle$, *except for the rules* $\langle p, \#, y \rangle \xrightarrow{\mathtt{r}:Z} \dots$ and $\langle p, x, \# \rangle \xrightarrow{\mathtt{l}:Z} \dots$ (i.e., $Z_1^{\mathtt{c}}$ tests are forbidden after # has been written to c). The next lemma shows that the above reduction is correct.

Lemma 6.10. *S has a run $(p_{\text{in}}, q_{\text{in}}, \epsilon, \epsilon) \xrightarrow{*} (p_{\text{fi}}, q_{\text{fi}}, u', v')$ for some $u' \in U'$ and $v' \in V'$ if, and only if, S' has a run $(\langle p_{\text{in}}, \top, \top \rangle, q_{\text{in}}, \epsilon, \epsilon) \xrightarrow{*} (\langle p_{\text{fi}}, \#, \# \rangle, q_f, \epsilon, \epsilon)$.*

Proof. "⇒": Suppose $C_0 = (p_{\text{in}}, q_{\text{in}}, \epsilon, \epsilon) \xrightarrow{\delta_1} C_1 \cdots \xrightarrow{\delta_n} C_n = (p_{\text{fi}}, q_{\text{fi}}, u', v')$, where $u' \in U'$, $v' \in V'$, is a run of S. We first transform it into a mimicking run $C'_0 = (\langle p_{\text{in}}, \top, \top \rangle, q_{\text{in}}, \epsilon, \epsilon) \xrightarrow{*} C'_n = (\langle p_{\text{fi}}, \#, \# \rangle, q_{\text{fi}}, \#u', \#v')$. This amounts to find some right points for inserting two steps of the forms $(\langle p, \top, y \rangle, q, u, v) \xrightarrow{\mathtt{r}!\#} (\langle p, \#, y \rangle, q, u\#, v)$ and $(\langle p, x, \top \rangle, q, u, v) \xrightarrow{\mathtt{l}!\#} (\langle p, x, \# \rangle, q, u, v\#)$ (in some order). For the first one, if $u' \neq \epsilon$ then we find the least index i_1 such that $C_{i_1} \xrightarrow{\mathtt{r}!a} C_{i_1+1}$ and the written occurrence of a is *permanent*, i.e., it is the first symbol-occurrence in u' in

$C_n = (p_{\mathrm{fi}}, q_{\mathrm{fi}}, u', v')$; if $u' = \epsilon$ then we find the least i_1 such that no $\mathtt{r}!a$ and no $\mathtt{r}{:}Z$ are performed in $C_j \xrightarrow{\delta_{j+1}} C_{j+1}$ with $j \geq i_1$. For $\mathtt{l}$ (and v') we find i_2 analogously. In either case, after i_1 (respectively, i_2) the channel $\mathtt{r}$ (respectively, $\mathtt{l}$) is not tested for $\mathtt{r}{:}Z$.

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

"$\Leftarrow$": Consider a run $C_0 = (\langle p_{\mathrm{in}}, \top, \top \rangle, q_{\mathrm{in}}, \epsilon, \epsilon) \xrightarrow{*} (\langle p_{\mathrm{fi}}, \#, \# \rangle, q_f, \epsilon, \epsilon) = C_n$ of S'. Since Receiver is in state q_{in} at the beginning and in q_f at the end, the Receiver step sequence must be composed of two parts: the first from q_{in} to q_{fi}, and the second from q_{fi} to q_f; the latter corresponds to a sequence of cleaning (reading) rules. The cleaning steps can be commuted after message losses (recall Lemma 6.6(4)), and after Sender's rules (Lemma 6.6(3)) since the first cleaning steps are $\mathtt{r}?\#$ and $\mathtt{l}?\#$ and Sender does not test the channels after having written # on them.

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

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

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

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

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

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

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

$W \subseteq Conf_{\mathtt{r}=\epsilon}$ is *upward-closed* (in $Conf_{\mathtt{r}=\epsilon}$) if $C \in W$, $C \sqsubseteq C'$ and $C' \in Conf_{\mathtt{r}=\epsilon}$ imply $C' \in W$. It is *downward-closed* if $Conf_{\mathtt{r}=\epsilon} \setminus W$ is upward-closed. The upward-closure $\uparrow W$ of $W \subseteq Conf_{\mathtt{r}=\epsilon}$ is the smallest upward-closed set that contains W. A well-known consequence of Higman's Lemma (see Remark 6.2) is that upward-closed and downward-closed subsets of $Conf_{\mathtt{r}=\epsilon}$ are

regular, and that upward-closed subsets can be canonically represented by their finitely many minimal elements.

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

Lemma 6.11. If S is a UCST$[Z_1^1]$ system and W is a regular subset of $Conf_{\mathtt{r}=\epsilon}$, then $\mathrm{Pre}^*(W)$ is upward-closed; moreover, given an oracle for G-G-Reach$[Z_1^1]$, $\mathrm{Pre}^*(W)$ is computable from S and W.

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

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

We put $U \stackrel{\text{def}}{=} Conf_{\mathtt{r}=\epsilon} \setminus \uparrow F_i$; note that U is regular. We check whether there exist some $C \in U$ and $D \in W$ such that $C \stackrel{*}{\to} D$; this can be decided using the oracle (it is a finite disjunction of G-G-Reach$[Z_1^1]$ instances, obtained by considering all possibilities for Sender and Receiver states). If the answer is "no", then $\uparrow F_i = \mathrm{Pre}^*(W)$; we then put $F = F_i$ and we are done.

Otherwise, the answer is "yes" and we look for some concrete $C \in U$ s.t. $C \stackrel{*}{\to} D$ for some $D \in W$. This can be done by enumerating all $C \in U$ and by using the decidability of G-G-Reach$[Z_1^1]$ again. We are bound to eventually find such C since $U \cap \mathrm{Pre}^*(W)$ is not empty.

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

Lemma 6.12. E-E-Reach$[Z_1]$ is Turing reducible to G-G-Reach$[Z_1^1]$.

Proof. Assume $S = (\{\mathtt{r}, \mathtt{l}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$ is a UCST$[Z_1]$, and we ask if there is a run $C_{\text{in}} = (p_{\text{in}}, q_{\text{in}}, \epsilon, \epsilon) \stackrel{*}{\to} (p_{\text{fi}}, q_{\text{fi}}, \epsilon, \epsilon) = C_{\text{fi}}$. By S' we denote the UCST$[Z_1^1]$ system arising from S by removing all $Z_1^{\mathtt{r}}$ rules. Hence Lemma 6.11 applies to S'. The set of configurations of S and S' is the same, so there is no ambiguity in using the notation $Conf$ and $Conf_{\mathtt{r}=\epsilon}$.

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

By Lemma 6.11, and using an oracle for G-G-Reach$[Z_1^1]$, we can compute $\mathrm{Pre}^*_{S'}(\{C_{\text{fi}}\})$, where the "$S'$" subscript indicates that we consider runs in S', not using $Z_1^{\mathtt{r}}$ tests. Hence $T_0 =$

$\mathrm{Pre}^*_{S'}(\{C_{\mathrm{fi}}\})$ is computable. Given T_k, we compute T_{k+1} as follows. We put

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

Thus $T'_k \subseteq \mathit{Conf}_{\mathtt{r}=\epsilon}$ is the set of configurations from which one can reach T_k with one (reliable) $Z_1^{\mathtt{r}}$ step. Clearly T'_k is upward-closed (since T_k is) and can be computed from a finite representation of T_k, e.g., its minimal elements. Then $T_{k+1} = T_k \cup \mathrm{Pre}^*_{S'}(T'_k)$, and we use Lemma 6.11 again to compute it.

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

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

As stated in Theorem 6.5 (see also Figure 6.3), our series of reductions from G-G-Reach[Z_1, N_1] to E-E-Reach[Z_1] also reduces G-G-Reach[Z_1^1] to E-E-Reach[Z_1^1]; this can be easily checked by recalling that the respective reductions do not introduce new tests. In subsection 6.6.1 we show a (polynomial) many-one reduction from E-E-Reach[Z_1^1] to $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$, a generalization of Post's Embedding Problem. Since $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ was shown decidable in chapter 5, our proof of Theorem 6.4 will be thus completed. We also add subsection 6.6.2 that shows a simple reduction in the opposite direction, from $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ to E-E-Reach[Z_1^1].

6.6.1 E-E-Reach[Z_1^1] reduces to $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$

We recall the definition of $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$:

Definition 6.13 (Post embedding with partial codirectness (chapter 5))**.** $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ is the question, given two finite alphabets Σ, Γ, two morphisms $u, v : \Sigma^* \to \Gamma^*$, and two regular languages $R, R' \in \mathrm{Reg}(\Sigma)$, whether there is $\sigma \in R$ (called a *solution*) such that $u(\sigma) \sqsubseteq v(\sigma)$, and such that furthermore $u(\sigma') \sqsubseteq v(\sigma')$ for all suffixes σ' of σ that belong to R'.

Lemma 6.14. E-E-Reach[Z_1^1] reduces to $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ (via a polynomial reduction).

We now prove the lemma. The reduction from E-E-Reach[Z_1^1] to $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ extends an earlier reduction from UCS to PEP [29]. In our case the presence of Z_1^1 tests creates new difficulties. We fix an instance $S = (\{\mathtt{r},\mathtt{1}\}, \mathtt{M}, Q_1, \Delta_1, Q_2, \Delta_2)$, $C_{\mathrm{in}} = (p_{\mathrm{in}}, q_{\mathrm{in}}, \epsilon, \epsilon)$, $C_{\mathrm{fi}} = (p_{\mathrm{fi}}, q_{\mathrm{fi}}, \epsilon, \epsilon)$ of E-E-Reach[Z_1^1], and we construct a $\mathsf{PEP}^{\mathrm{partial}}_{\mathrm{codir}}$ instance $\mathscr{P} = (\Sigma, \Gamma, u, v, R, R')$ intended to express the existence of a run from C_{in} to C_{fi}.

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

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

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

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

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

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

We will now prove the correctness of this reduction, i.e., show that S has a run $C_{\text{in}} \stackrel{*}{\rightarrow} C_{\text{fi}}$ if, and only if, $\mathscr{P}$ has a solution. Before starting with the proof itself, let us illustrate some aspects of the reduction by considering a schematic example (see Figure 6.7).

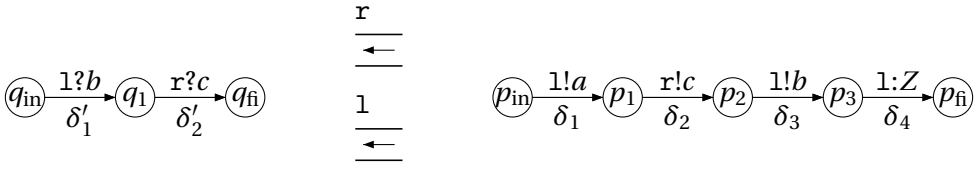


Figure 6.7: A schematic UCST$[Z_1^{\mathtt{l}}]$ instance

Let us consider $\sigma_{\text{sol}} = \delta_1\delta'_1\delta_2\delta'_2\delta_3\delta_4$ and check whether it is a solution of the $\mathscr{P}$ instance obtained by our reduction. For this, one first checks that $\sigma_{\text{sol}} \in R$, computes $u(\sigma_{\text{sol}}) = read\_\mathtt{l}(\sigma_{\text{sol}}) = b$ and check that $b \sqsubseteq v(\sigma_{\text{sol}}) = write\_\mathtt{l}(\sigma_{\text{sol}}) = ab$. There remains to check the suffixes

of σ_{sol} that belong to R', i.e., that start with a $\mathtt{l}{:}Z$ rule. Here, only $\sigma' = \delta_4$ is in R', and indeed $u(\sigma') = \epsilon \sqsubseteq v(\sigma')$. Thus σ_{sol} is a solution.

However, a solution like σ_{sol} does not directly correspond to a run of S. For instance, any run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ in the system from Figure 6.7 must use δ_3 (write b on $\mathtt{l}$) before δ'_1 (read it).

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

$$C_{\text{in}} \xrightarrow{\delta_1} C_1 \xrightarrow{\delta_2} C_2 \xrightarrow{\delta_3} C_3 = (p_3, q_{\text{in}}, c, ab) \xrightarrow{\text{los}} C_4 = (p_3, q_{\text{in}}, c, b) \xrightarrow{\delta'_1} C_5 \xrightarrow{\delta_4} C_6 \xrightarrow{\delta'_2} C_{\text{fi}} \qquad (\pi)$$

has an action in "$C_3 \xrightarrow{\text{los}} C_4$" that is not accounted for in Σ and cannot appear in solutions of $\mathscr{P}$. Also, the Σ-word $\sigma_\pi = \delta_1\delta_2\delta_3\delta'_1\delta_4\delta'_2$ obtained from π is not a solution. It belongs to $P_1 \| P_2$ but not to $E_{\mathtt{r}}^*$ (which requires that each occurrence of δ_2 is immediately followed by some $. \xrightarrow{\mathtt{r}?c} .$ rule). Note that σ_{sol} had δ_2 followed by δ'_2, but it is impossible in a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ to have δ_2 immediately followed by δ'_2.

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

(c1) $\sigma \in P_1 \| P_2$;

(c2) $\mathit{read}\_\mathtt{r}(\sigma) = \mathit{write}\_\mathtt{r}(\sigma)$;

(c3) $\mathit{read}\_\mathtt{r}(\sigma_1)$ is a prefix of $\mathit{write}\_\mathtt{r}(\sigma_1)$ for each prefix σ_1 of σ;

(c4) $\mathit{read}\_\mathtt{l}(\sigma) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma)$;

(c5) $\mathit{read}\_\mathtt{l}(\sigma_2) \sqsubseteq \mathit{write}\_\mathtt{l}(\sigma_2)$ for each factorization $\sigma = \sigma_1\delta\sigma_2$ where $\delta \in T_1$ (i.e., δ is a $\mathtt{l}{:}Z$ rule).

A pre-solution need not be a solution of $\mathscr{P}$, as it need not be in $E_{\mathtt{r}}^*$. A pre-solution need not correspond to any run of S, as it may have reads on $\mathtt{l}$ before their corresponding writes. In the reverse direction, it is easy to see that every solution of $\mathscr{P}$ is also a pre-solution. Similarly, every run of S corresponds to a pre-solution obtained by reading off all the rules along the run (ignoring losses).

A pre-solution σ has a *Receiver-advancing switch* if $\sigma = \sigma_1\delta\delta'\sigma_2$ where δ is a Sender rule, δ' is a Receiver rule, and $\sigma' = \sigma_1\delta'\delta\sigma_2$ is again a pre-solution. A *Receiver-postponing switch* is defined analogously, for δ being a Receiver rule and δ' being a Sender rule. For example, the sequence σ_π above is a pre-solution. It has a Receiver-advancing switch on δ_3 and δ'_1, and one on δ_4 and δ'_2. Note that when σ is a pre-solution, checking whether a potential Receiver-advancing or Receiver-postponing switch leads again to a pre-solution only requires checking (c3) or, respectively, (c5). Considering another example, σ_{sol}, being a solution is a pre-solution. It has two Receiver-postponing switches but only one Receiver-advancing switch since switching δ_2 and δ'_2 does not maintain (c3).

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

Claim 6.15. Any advance-stable pre-solution σ is in $E_{\mathtt{r}}^*$, and it is thus a solution of $\mathscr{P}$.

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

Claim 6.16. If $\sigma = \delta_1\ldots\delta_n$ is a postpone-stable pre-solution, S has a run of the form $C_{\mathrm{in}} \xrightarrow{\delta_1\,\mathrm{los}^*} \cdots \xrightarrow{\delta_n\,\mathrm{los}^*} C_{\mathrm{fi}}$.

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

So let us consider channel contents, starting with $\mathtt{r}$. For $i = 0,\ldots,n$, let $x_i = \mathit{read}\_\mathtt{r}(\delta_1\ldots\delta_i)$ and $y_i = \mathit{write}\_\mathtt{r}(\delta_1\ldots\delta_i)$. Since σ satisfies (c3), y_i is some $x_i x_i'$ (and $x_0' = \epsilon$). One can easily verify by induction on i that after firing $\sigma_1\ldots\sigma_i$ from C_{in}, $\mathtt{r}$ contains exactly x_i'. In fact (c3) implies that if δ_{i+1} reads on $\mathtt{r}$, it must read the first letter of x_i' (and δ_{i+1} cannot be a read on $\mathtt{r}$ when $x_i' = \epsilon$).

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

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

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

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

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

We now see that our reduction is correct. Indeed, if $C_{\text{in}} \xrightarrow{\sigma} C_{\text{fi}}$ is a run of S then σ with all occurrences of los removed is a pre-solution; and there is also an advance-stable pre-solution, i.e., a solution of $\mathscr{P}$. On the other hand, if σ is a solution of $\mathscr{P}$ then σ is a pre-solution, and there is also a postpone-stable pre-solution, which corresponds to a run $C_{\text{in}} \xrightarrow{*} C_{\text{fi}}$ of S. This finishes the proof of Lemma 6.14, and of Theorem 6.4.

6.6.2 $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ reduces to E-E-Reach[$Z_1^{\mathtt{l}}$]

We now prove a converse of Lemma 6.14, thus showing that $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ and E-E-Reach[$Z_1^{\mathtt{l}}$] are equivalent problems. Actually, $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ can be easily reduced to E-E-Reach[$Z_i^{\mathtt{c}}$] for any $i \in \{1,2\}$ and $\mathtt{c} \in \mathtt{Ch}$, but we only show a reduction for $i = 1$ and $\mathtt{c} = \mathtt{l}$ explicitly. (The other reductions would be analogous.)

Lemma 6.17. $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ reduces to E-E-Reach[$Z_1^{\mathtt{l}}$] (via a polynomial reduction).

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

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

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

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

When generating a_i, Sender writes $u(a_i)$ on $\mathtt{r}$ and $v(a_i)$ on $\mathtt{l}$. To check that $\mathtt{r}$ contains a subword of $\mathtt{l}$, Receiver behaves as in Figure 6.8 (that illustrates another reduction). So far we have

guaranteed that there is a run $(p_{\text{in}}, q_{\text{loop}}, \epsilon, \epsilon) \xrightarrow{*} (p_{\text{fi}}, q_{\text{loop}}, \epsilon, \epsilon)$ iff there is $\sigma = a_1 a_2 \ldots, a_m \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ (using the lossiness of $\mathtt{l}$ where $v(\sigma)$ has been written).

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

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

Remark 6.18 (On complexity)**.** Based on known results on the complexity of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ (see [112, 76]), our reductions prove that reachability for UCST$[Z, N]$ is $\mathbf{F}_{\omega^\omega}$-complete, using the ordinal-recursive complexity classes introduced in [111].

6.7 Two undecidable problems for UCST$[Z, N]$

The main result of this chapter is Theorem 6.4, showing the decidability of the reachability problem for UCST$[Z, N]$. In this section we argue that the emptiness and non-emptiness tests ("Z" and "N") strictly increase the expressive power of UCSes. We do this by computational arguments, namely by exhibiting two variants of the reachability problem that are undecidable for UCST$[Z, N]$. Since these variants are known to be decidable for plain UCSes (with no tests), we conclude that there is no effective procedure to transform a UCST$[Z, N]$ into an equivalent UCS in general. Subsection 6.7.1 deals with the problem of *recurrent reachability* of a control state. In Subsection 6.7.2 we consider the usual reachability problem but we assume that *messages can be lost only during writing* to $\mathtt{l}$ (i.e., we assume that channel $\mathtt{l}$ is reliable and that the unreliability is limited to the writing operation).

6.7.1 Recurrent reachability

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

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

Theorem 6.19. Recurrent reachability is decidable for UCSes, and is Σ_1^0-complete (hence undecidable) for UCST$[Z_1^{\mathtt{r}}]$.

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

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

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

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

case 2 Otherwise, if π only uses finitely many Z tests, it can be written under the form $C_{\text{in}} \xrightarrow{*} C = (p,q,u,v) \to \cdots$ such that no test occur after C. After C, any step by Sender can be advanced before Receiver steps and message losses, according to Lemma 6.6(3). Assuming that π uses infinitely many Sender steps, we conclude that S has a pq^∞ run that eventually only uses Sender rules (but no Z tests). At this point, we can forget about the contents of the channels (they are not read or tested anymore). Hence a finite witness for such pq^∞-runs is obtained by the combination of a finite run $C_{\text{in}} \xrightarrow{*} (p,q,u,v)$ and a loop $p = p_1 \xrightarrow{\delta_1} p_2 \xrightarrow{\delta_2} \cdots p_n \xrightarrow{\delta_n} p_1$ in Sender's rules that does not use any testing rule.

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

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

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

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

A *semi-Thue system* $T = (\Gamma, R)$ consists of a finite alphabet Γ and a finite set $R \subseteq \Gamma^* \times \Gamma^*$ of *rewrite rules*; we write $\alpha \to \beta$ instead of $(\alpha,\beta) \in R$. The system gives rise to a *one-step rewrite relation* $\to_R \subseteq \Gamma^* \times \Gamma^*$ as expected: $x \to_R y \overset{\text{def}}{\Leftrightarrow} x$ and y can be factored as $x = z\alpha z'$ and $y = z\beta z'$

for some rule $\alpha \to \beta$ and some strings $z, z' \in \Gamma^*$. As usual, we write $x \xrightarrow{+}_R y$ if x can be rewritten into y by a nonempty sequence of steps.

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

Fact 6.20. The question whether a given length-preserving semi-Thue system has a loop is Σ^0_1-complete.

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

Let $T = (\Gamma, R)$ be a given length-preserving semi-Thue system. We construct a UCST S, with message alphabet $\mathtt{M} \stackrel{\text{def}}{=} \Gamma \uplus \{\#\}$. The reduction is illustrated in Figure 6.8, assuming $\Gamma = \{a, b\}$. The resulting S behaves as follows:

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

1. check that (equivalently, wait until) $\mathtt{r}$ is empty;
2. send # on $\mathtt{l}$;
3. nondeterministically send a string $z \in \Gamma^*$ on both $\mathtt{l}$ and $\mathtt{r}$;
4. nondeterministically choose a rewrite rule $\alpha \to \beta$ (from R) and send α on $\mathtt{r}$ and β on $\mathtt{l}$;
5. nondeterministically send a string $z' \in \Gamma^*$ on both $\mathtt{l}$ and $\mathtt{r}$;
6. send # on $\mathtt{r}$;
7. go back to p_{loop} (and repeat 1–7).

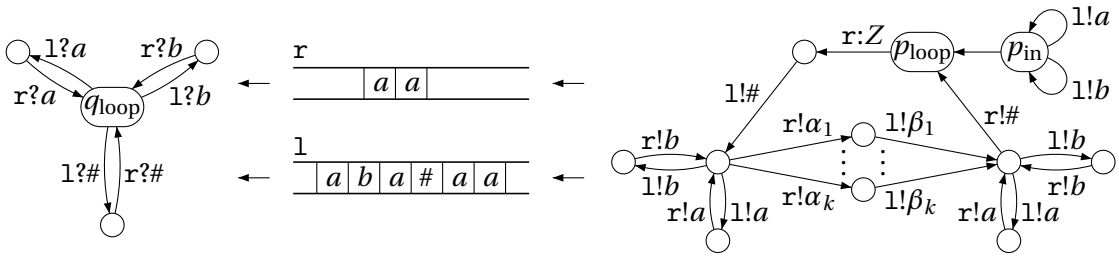


Figure 6.8: Solving the looping problem for semi-Thue systems

The above loop 1–7 can be also summarized as: check that $\mathtt{r}$ is empty, nondeterministically guess two strings x and y such that $x \to_R y$, writing $x\#$ on $\mathtt{r}$ and $\#y$ on $\mathtt{l}$.

(b) Receiver starts in state q_{loop} from where it reads any pair of identical symbols from `r` and `l`, returns to q_{loop}, and repeats this indefinitely.

Claim 6.21 (Correctness of the reduction)**.** S has an infinite run starting from $C_{\text{in}} = (p_{\text{in}}, q_{\text{loop}}, \epsilon, \epsilon)$ and visiting the control pair $(p_{\text{loop}}, q_{\text{loop}})$ infinitely often if, and only if, $x \xrightarrow{+}_R x$ for some $x \in \Gamma^*$.

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

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

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

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

6.7.2 Write-lossy semantics

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

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

Each `l`-writing rule δ of the form $p \xrightarrow{1!x} p'$ in a UCST S will give rise to *write-lossy steps* of the form $(p, q, u, v) \xrightarrow{\text{wrlo}} (p', q, u, v)$, where δ is performed but nothing is actually written. We

write $C \rightarrow_{\text{wrlo}} C'$ when there is a reliable or a write-lossy step from C to C', and use $C \rightarrow_{\text{rel}} C'$ and $C \rightarrow_{\text{los}} C'$ to denote the existence of a reliable step, and respectively, of a reliable or a lossy step. Then $\rightarrow_{\text{rel}} \subseteq \rightarrow_{\text{wrlo}} \subseteq \xrightarrow{*}_{\text{los}}$.

Now we make precise the equivalence of the two semantics when we start with the empty $\mathtt{l}$ and only use the emptiness tests:

Lemma 6.22. Assume S is a UCST[Z] system. Let $C_{\text{in}} = (p, q, u, \epsilon)$ be a configuration (where $\mathtt{l}$ is empty). Then, for any C_{fi} configuration, $C_{\text{in}} \xrightarrow{*}_{\text{los}} C_{\text{fi}}$ iff $C_{\text{in}} \xrightarrow{*}_{\text{wrlo}} C_{\text{fi}}$.

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

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

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

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

Corollary 6.23. E-G-Reachability is decidable for UCST[Z] with write-lossy semantics.

The write-lossy semantics is meaningful when modeling unreliability of the writing actions as opposed to unreliability of the channels. In the literature, write-lossy semantics is mostly used as a way of restricting the nondeterminism of message losses without losing any essential generality, relying on equivalences like Lemma 6.22 (see, e.g., [30, section 5.1]).

However, for our UCST systems, the write-lossy and the standard lossy semantics do not coincide when N tests are allowed. In fact, Theorem 6.4 does not extend to write-lossy systems.

Theorem 6.24. E-E-Reach is undecidable for UCST[$Z_1^{\mathtt{l}}, N_1^{\mathtt{l}}$] with write-lossy semantics.

Proof Idea. As in subsection 6.2.2, Sender simulates a queue automaton using tests and the help of Receiver. See Figure 6.9. Channel $\mathtt{l}$ is initially empty. To read, say, a from $\mathtt{r}$, Sender does the following: (1) write a on $\mathtt{l}$; (2) check that $\mathtt{l}$ is nonempty (hence the write was not lost); (3) check that, i.e., wait until, $\mathtt{l}$ is empty. Meanwhile, Receiver reads identical letters from $\mathtt{r}$ and $\mathtt{l}$. □

Thus, at least in the write-lossy setting, we can separate UCST[Z] and UCST[$Z_1^{\mathtt{l}}, N_1^{\mathtt{l}}$] with respect to decidability of reachability.

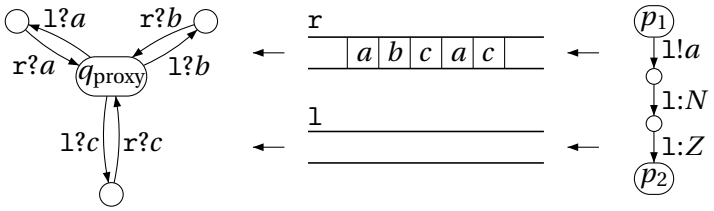


Figure 6.9: Write-lossy Sender simulates "$p_1 \xrightarrow{\mathtt{r?}a} p_2$" with N and Z tests and proxy Receiver

6.8 Concluding remarks

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

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

1. What about occurrence and non-occurrence tests, defined as $\{O_a, NO_a \mid a \in \mathtt{M}\}$ with $O_a = \mathtt{M}^*.a.\mathtt{M}^*$ and $NO_a = (\mathtt{M} \setminus \{a\})^*\mathtt{?}$ Such tests generalize N and Z tests and have been considered for channel systems used as a tool for questions on Metric Temporal Logic [24].

2. What about UCSTs with tests restricted to the lossy `l` channel? The undecidable reachability questions in Theorem 6.3 all rely on tests on the reliable `r` channel.

We have made partial progress on the decidability of non-occurrence tests: we can adapt the reduction from E-E-Reach$[Z_1^{\mathtt{l}}]$ to $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ from section 6.6 to work with non-occurrence tests NO_a. This needs a slight variant of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$. For words x and y and a letter a, say $x \sqsubseteq_a y$ if every suffix of x beginning with an a is a subword of y. Equivalently, $x \sqsubseteq_a y$ iff a does not occur in x or if a does occur and the longest suffix of x starting with an a is a subword of y. Then by a reduction very similar to that from section 6.6, the E-E-Reach problem for UCST equipped with NO_a tests by the sender on `l` can be reduced to the following variant of

$\mathsf{PEP}^{\text{partial}}_{\text{codir}}$:

> Given morphisms $u, v : \Sigma^* \to \Gamma^*$, regular languages $R, R' \subseteq \Sigma^*$, and a letter $a \in \Gamma$, does there exist $\sigma \in R$ such that $u(\sigma) \sqsubseteq v(\sigma)$ and for every suffix σ' of σ which belongs to R', $u(\sigma') \sqsubseteq_a v(\sigma')$?

This variant of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ is decidable; the proof proceeds along almost the same lines as the proof of decidability of $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ (Theorem 5.4 from section 5.2), by cutting a long solution to obtain a bound on the length of the shortest solution if one exists.

This can be easily extended to UCST with non-occurrence tests on multiple letters. Thus we can show that E-E-reachability for UCST with non-occurrence tests by sender on `1` is decidable. However this is far from satisfactory, as for general N and Z tests (and arbitrary regular initial and final channel contents) we have an intricate sequence of reductions, and it is not clear how to proceed in the presence of non-occurrence tests. For example the Turing reduction from section 6.5 which eliminates emptiness Z_1^r tests does not adapt to non-occurrence tests. Handling even "simple" [non-]occurrence tests remains a challenging problem.

Conclusion

We considered three main problems, all related to the verification of systems analyzable with theory of well-structured transition systems (section 1.3, [48])and well-quasi-orders (section 2.2), and in particular the subword relation.

We addressed fundamental state complexity questions involving closures and interiors of languages represented by finite-state automata with respect to the subword relation. We improved upon existing results by providing exact bounds for closures, and showed new results for the two-letter case, as well as for the new problem of computing interiors. These results point to more general questions: what are the right data structures for reasoning with (sets of) subwords and superwords? Further directions of research suggested by our work also include investigating interiors for other orders such as prefix, suffix, factor, and the priority order from [53], as well as decision problems involving interiors.

The Post Embedding Problem is a convenient problem at level $\mathsf{F}_{\omega^\omega}$ in the fast-growing hierarchy, and along with its new generalization $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ can also be seen as an algebraic abstraction of the reachability problem for LCSes and UCSes. We showed that $\mathsf{PEP}^{\text{partial}}_{\text{dir}}$ and its equivalent mirror $\mathsf{PEP}^{\text{partial}}_{\text{codir}}$ are decidable, by bounding the length of the smallest solution, if one exists. We also showed decidability for the universal and counting versions of the problem, and that combining directness and codirectness leads to undecidability. Further directions of research include using PEP for lower bounds where reachability on LCS was used earlier, with the goal of obtaining bounds for new problems, and having simpler proofs, possibly parameterized by the alphabet size [76]. PEP-like problems for other wqos and their use in lower bounds is another area open to exploration.

In UCSes, Sender communicates to Receiver over one lossy and one reliable channel. This model is closely related to the Post Embedding Problem. In addition, UCSes are a minimal setting to which one can reduce reachability problems for more complex combinations of lossy and reliable channels. We extended the basic UCS model with channel tests, which allow limited synchronization between Sender and Receiver. We showed through a series of reductions that reachability is decidable for UCS extended with emptiness and non-emptiness tests. It was surprisingly hard to show decidability for emptiness and nonemptiness tests, and extending these results, either positively or negatively, to more general tests remains a

challenging open problem.

Bibliography

[1] Parosh Aziz Abdulla. Well (and better) quasi-ordered transition systems. *Bulletin of Symbolic Logic*, 16(4):457–515, December 2010.

[2] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol. In *Proc. TACAS 1999*, volume 1579 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 1999.

[3] Parosh Aziz Abdulla, Christel Baier, S. Purushothaman Iyer, and Bengt Jonsson. Simulating perfect channels with probabilistic lossy channels. *Information and Computation*, 197(1-2):22–40, February 2005.

[4] Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *Formal Methods in System Design*, 25(1):39–65, July 2004.

[5] Parosh Aziz Abdulla, Giorgio Delzanno, and Laurent Van Begin. A classification of the expressive power of well-structured transition systems. *Information and Computation*, 209(3):248–279, March 2011.

[6] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[7] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proc. LICS 1993*. IEEE, 1993.

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

[9] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *Proc. CONCUR 2004*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.

[10] M. Arfi. Polynomial Operations on Rational Languages. In *Proc. STACS 1987*, volume 247 of *Lecture Notes in Computer Science*, pages 198–206. Springer, 1987.

[11] Eugene Asarin and Pieter Collins. Noisy Turing Machines. In *Automata, Languages and Programming SE - 83*, volume 3580 of *Lecture Notes in Computer Science*, pages 1031–1042. Springer, 2005.

[12] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. On the verification problem for weak memory models. *ACM SIGPLAN Notices*, 45(1):7, January 2010.

[13] Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the Reachability Analysis of Acyclic Networks of Pushdown Systems. In *Proc. CONCUR 2008*, volume 5201, pages 356–371. Springer, 2008.

[14] G. Bachmeier, M. Luttenberger, and M. Schlund. Finite Automata for the Sub- and Superword Closure of CFLs: Descriptional and Computational Complexity. arXiv:1410.2737 [cs.FL], October 2014.

[15] Ricardo A Baeza-Yates. Searching subsequences. *Theoretical Computer Science*, 78(2):363–376, January 1991.

[16] Pablo Barcelo, Diego Figueira, and Leonid Libkin. Graph Logics with Rational Relations. *Logical Methods in Computer Science*, 9(3), July 2013.

[17] Keith A Bartlett, Roger A Scantlebury, and Peter T Wilkinson. A note on reliable full-duplex transmission over half-duplex links. In *Communications of the ACM*, volume 12, pages 260–261, May 1969.

[18] Nathalie Bertrand and Philippe Schnoebelen. Computable fixpoints in well-structured symbolic model checking. *Formal Methods in System Design*, 43(2):233–267, August 2012.

[19] Jean-Camille Birget. Partial orders on words, minimal elements of regular languages, and state complexity. *Theoretical Computer Science*, 119(2):267–291, October 1993.

[20] Jean-Camille Birget. The state complexity of $\overline{\Sigma^* \overline{L}}$ and its connection with temporal logic. *Information Processing Letters*, 58(4):185–188, May 1996.

[21] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. CONCUR 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[22] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *Proc. CAV 2000*, Lecture Notes in Computer Science, pages 403–418. Springer, 2000.

[23] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. On termination and invariance for faulty channel machines. *Formal Aspects of Computing*, 24(4-6):595–607, June 2012.

[24] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The Cost of Punctuality. In *Proc. LICS 2007*, pages 109–120. IEEE, 2007.

[25] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, April 1983.

[26] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, January 1996.

[27] Pierre Chambart and Philippe Schnoebelen. Post Embedding Problem Is Not Primitive Recursive, with Applications to Channel Systems. In *Proc. FSTTCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

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

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

[30] Pierre Chambart and Philippe Schnoebelen. The Ordinal Recursive Complexity of Lossy Channel Systems. In *Proc. LICS 2008*, pages 205–216. IEEE, June 2008.

[31] Pierre Chambart and Philippe Schnoebelen. Computing blocker sets for the Regular Post Embedding Problem. In *Proc. DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2010.

[32] Pierre Chambart and Philippe Schnoebelen. Pumping and counting on the Regular Post Embedding Problem. In *Proc. ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2010.

[33] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *Proc. FOCS 1976*, pages 98–108. IEEE, October 1976.

[34] Edmund Clarke, Olna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[35] Lorenzo Clemente, Frédéric Herbreteau, Amelie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *Proc. FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96, 2013.

[36] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, UK, 1971.

[37] Bruno Courcelle. On constructing obstruction sets of words. *EATCS Bulletin*, 44:178–185, 1991.

[38] Patrick Cousot and Radhia Cousot. Abstract interpretation. In *Proc. POPL 1977*, pages 238–252. ACM Press, 1977.

[39] Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.

[40] Cees Elzinga, Sven Rahmann, and Hui Wang. Algorithms for subsequence combinatorics. *Theoretical Computer Science*, 409:394–404, 2008.

[41] E. Allen Emerson. The Beginning of Model Checking: A Personal Perspective. In *25 Years of Model Checking SE - 2*, volume 5000 of *Lecture Notes in Computer Science*, pages 27–45. Springer, 2008.

[42] M Fairtlough and S S Wainer. Hierarchies of provably recursive functions. In S Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, chapter III, pages 149–207. Elsevier Science, 1998.

[43] Alain Finkel. A generalization of the procedure of Karp and Miller to well-structured transition systems. In *Proc. ICALP 1987*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987.

[44] Alain Finkel. A new class of analyzable CFSMs with unbounded FIFO channels. In *Proc. PSTV 1988*. North-Holland, 1988.

[45] Alain Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, December 1990.

[46] Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, March 1994.

[47] Alain Finkel and Jean Goubault-Larrecq. Forward Analysis for WSTS, Part II: Complete WSTS. *Logical Methods in Computer Science*, 8(3), September 2012.

[48] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[49] Robert W Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[50] Mohamed G. Gouda. Synthesis of Communicating Finite-State Machines with Guaranteed Progress. *IEEE Transactions on Communications*, 32(7):779–788, July 1984.

[51] Hermann Gruber and Markus Holzer. Finding Lower Bounds for Nondeterministic State Complexity Is Hard. In *Proc. DLT 2006*, volume 4036 of *Lecture Notes in Computer Science*, pages 363–374. Springer, 2006.

[52] Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the size of Higman-Haines sets: effective constructions. *Fundamenta Informaticae*, pages 1–17, 2009.

[53] Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The Power of Priority Channel Systems. In *Proc. CONCUR 2013*, volume 8052 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2013.

[54] Peter Habermehl, Roland Meyer, and Harro Wimmel. The Downward-Closure of Petri Net Languages. In *Proc. ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010.

[55] Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The Ordinal-Recursive Complexity of Timed-arc Petri Nets, Data Nets, and Other Enriched Nets. In *Proc. LICS 2012*, pages 355–364. IEEE, June 2012.

[56] Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, January 1969.

[57] Pierre-Cyrille Héam. On shuffle ideals. *RAIRO-Theoretical Informatics and Applications*, 36(2002):359–384, 2002.

[58] Alexander Heußner, Tristan Le Gall, and Grégoire Sutre. Safety Verification of Communicating One-Counter Machines. In *Proc. FSTTCS 2012*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 224–235. Leibniz-Zentrum für Informatik, 2012.

[59] Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability Analysis of Communicating Pushdown Systems. *Logical Methods in Computer Science*, 8(3), September 2012.

[60] Graham Higman. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, January 1952.

[61] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[62] Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. *Theoretical Computer Science*, 327(3):319–347, November 2004.

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

[64] Gerard J. Holzmann. Economics of software verification. In *Proc. PASTE 2001*, pages 80–89. ACM Press, 2001.

[65] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[66] Pieter Hooimeijer and Margus Veanes. An Evaluation of Automata Algorithms for String Analysis. In *Proc. VMCAI 2011*, volume 6538 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2011.

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

[68] Neil Immerman. Languages that Capture Complexity Classes. *SIAM Journal on Computing*, 16(4):760–778, August 1987.

[69] International Standards Organization. Data Communications - HDLC Procedures - Elements of Procedures. Technical report, Geneva, 1979.

[70] Petr Jančar, Prateek Karandikar, and Philippe Schnoebelen. Unidirectional Channel Systems Can Be Tested. In *Proc. IFIP TCS 2012*, volume 7604 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2012.

[71] Petr Jančar, Prateek Karandikar, and Philippe Schnoebelen. On Reachability for Unidirectional Channel Systems Extended with Regular Tests. *Logical Methods in Computer Science (in press)*, 2014.

[72] Jeff Kahn. Entropy, independent sets and antichains: A new approach to Dedekind's Problem. *Proceedings Of The American Mathematical Society*, 130:371–378, 2002.

[73] Jui-Yi Kao, Narad Rampersad, and Jeffrey Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47-49):5010–5021, November 2009.

[74] Prateek Karandikar, Manfred Kufleitner, and Philippe Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, November 2014.

[75] Prateek Karandikar, Matthias Niewerth, and Philippe Schnoebelen. On the state complexity of closures and interiors of regular languages with subwords. *CoRR*, abs/1406.0, 2014.

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

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

[78] Prateek Karandikar and Philippe Schnoebelen. Generalized Post Embedding Problems. *Theory of Computing Systems*, September 2014.

[79] Prateek Karandikar and Philippe Schnoebelen. On the state complexity of closures and interiors of regular languages with subwords. In *Proc. DCFS 2014*, volume 8614 of *Lecture Notes in Computer Science*, pages 234–245. Springer, 2014.

[80] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[81] Kamilla Kátai-Urbán, Péter Pál Pach, Gabriella Pluhár, András Pongrácz, and Csaba Szabó. On the word problem for syntactic monoids of piecewise testable languages. *Semigroup Forum*, 84(2):323–332, November 2011.

[82] Christoph Kern and Mark R. Greenstreet. Formal verification in hardware design: a survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2):123–193, April 1999.

[83] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

[84] Ondřej Klíma and Libor Polák. Alternative Automata Characterization of Piecewise Testable Languages. In *Proc. DLT 2013*, volume 7907 of *Lecture Notes in Computer Science*, pages 289–300. Springer, 2013.

[85] Boris Konev, Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Dynamic topological logics over spaces with continuous functions. In *Advances in Modal Logic*, volume 6, pages 299–318. College Publications, 2006.

[86] Agi Kurucz. Combining Modal Logics. In *Handbook of Modal Logics*, volume 3, chapter 15, pages 869–926. Elsevier Science, 2006.

[87] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-Bounded Analysis of Concurrent Queue Systems. In *Proc. TACAS 2008*, volume 4963, pages 299–314. Springer, 2008.

[88] Simon S. Lam and A. Udaya Shankar. Protocol Verification via Projections. *IEEE Transactions on Software Engineering*, SE-10(4):325–342, July 1984.

[89] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2):1–27, March 2008.

[90] Sylvain Lombardy and Jacques Sakarovitch. The universal automaton. In *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 457–504. Amsterdam University Press, 2008.

[91] Zohar Manna and Amir Pnueli. Verification of Parameterized Programs. In *Specification and Validation Methods*, pages 167–230. University Press, 1995.

[92] Raymond E. Miller. The construction of self-synchronizing finite state protocols. *Distributed Computing*, 2(2):104–112, June 1987.

[93] C. St. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Mathematical Proceedings of the Cambridge Philosophical Society*, 59(04):833, October 1963.

[94] Alexander Okhotin. On the state complexity of scattered substrings and superstrings. *Fundamenta Informaticae*, 99:325–338, 2010.

[95] Joël Ouaknine and James Worrell. On Metric Temporal Logic and Faulty Turing Machines. In *Proc. FOSSACS 2006*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[96] Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1):1–27, February 2007.

[97] Jan K. Pachl. Protocol Description and Analysis Based on a State Transition Model with Channel Expressions. In *Proc. PSTV 1987*, pages 207–219. North-Holland, May 1987.

[98] Jean-Éric Pin. *Varieties of Formal Languages*. Plenum, New-York, 1986.

[99] Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, July 1997.

[100] Thomas Place, Lorijn Van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proc. MFCS 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2013.

[101] Amir Pnueli. The temporal logic of programs. In *Proc. FOCS 1977*, pages 46–57. IEEE, 1977.

[102] Emil L. Post. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics*, 65(2):197, April 1943.

[103] Hans Jürgen Prömel and Wolfgang Thumser. Fast growing functions based on Ramsey theorems. *Discrete Mathematics*, 95(1-3):341–358, December 1991.

[104] Jean Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming SE - 22*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[105] C. V. Ramamoorthy, Y. Yaw, R. Aggarwal, and J. Song. Synthesis of two-party error-recoverable protocols. In *Proc. SIGCOMM 1986*, pages 227–235. ACM Press, 1986.

[106] Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informaticae*, 116:223–236, July 2012.

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

[108] Jacques Sakarovitch and Imre Simon. Subwords. In M Lothaire, editor, *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*, chapter 6, pages 105–142. Cambridge University Press, 1983.

[109] Arto Salomaa, Kai Salomaa, and Sheng Yu. State complexity of combined operations. *Theoretical Computer Science*, 383(2-3):140–152, September 2007.

[110] David Sankoff and Joseph Kruskal. *Time Warps, String Edits, and Macromolecules*. Addison-Wesley Reading, 1983.

[111] Sylvain Schmitz. Complexity Hierarchies Beyond Elementary. *CoRR*, abs/1312.5:36, December 2013.

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

[113] Sylvain Schmitz and Philippe Schnoebelen. Algorithmic Aspects of WQO Theory. *HAL*, cel-007270, 2012.

[114] Sylvain Schmitz and Philippe Schnoebelen. The Power of Well-Structured Systems. In *Proc. CONCUR 2013*, volume 8052 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2013.

[115] Philippe Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In *Proc. MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010.

[116] Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

[117] Carl A. Sunshine. Formal Modeling of Communication Protocols. In *Kommunikation in verteilten Systemen SE - 24*, volume 40 of *Informatik-Fachberichte*, pages 406–428. Springer, 1981.

[118] Rüdiger Valk and Matthias Jantzen. The residue of vector sets with applications to decidability problems in Petri nets. *Acta Informatica*, 21(6):643–674, March 1985.

[119] Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, January 1978.

[120] Pierre Wolper and Bernard Boigelot. Verifying Systems with Infinite but Regular State Spaces. In *Proc. CAV 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 1998.

[121] Sheng Yu. State Complexity: Recent Results and Open Problems. *Fundamenta Informaticae*, 64(1–4):471–480, 2005.

[122] P. Zafiropulo, C. West, H. Rudin, D. Cowan, and D. Brand. Towards Analyzing and Synthesizing Protocols. *IEEE Transactions on Communications*, 28(4):651–661, April 1980.

[123] Georg Zetzsche. Computing downward closures for stacked counter automata. arxiv:1409.7922 [cs.FL], September 2014.

Index