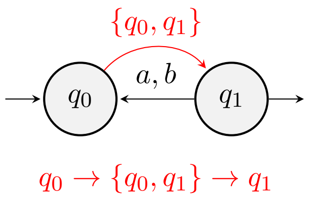
Non-Emptiness Test for Automata on Linear Orderings: an Efficient Implementation

Auteur : Braipson, Thomas
Promoteur(s) : Boigelot, Bernard
Faculté : Faculté des Sciences appliquées

Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "electronic systems and devices"
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/23218

UNIVERSITÉ DE LIÈGE
FACULTÉ DES SCIENCES APPLIQUÉES



Non-Emptiness Test for Automata on Linear Orderings: an Efficient Implementation

BRAIPSON Thomas

Thesis presented to obtain the degree of:
Master of Science in Electrical Engineering, professional focus in electronic systems and devices

Thesis supervisor:
BOIGELOT Bernard

Academic year: **2024-2025**

Abstract

Automata on linear orderings are a form of finite-state automata introduced by Véronique Bruyère and Olivier Carton, that generalise the concepts of finite-word, infinite-word, and transfinite-word automata. They accept words indexed by linear orderings, defined as mappings from the elements of such orderings to a finite alphabet. This master's thesis addresses the problem of deciding whether an arbitrary automaton on linear orderings accepts at least one word. In the scope of implementing a decision procedure for the monadic first-order theory of order over real or rational numbers, we focus on deciding whether a given automaton on linear orderings accepts at least one word indexed by the real or the rational numbers. Our work is based on a decision procedure recently obtained by Bernard Boigelot, Pascal Fontaine and Baptiste Vergain. While the critical step of the original procedure has quadratic cost in the size of the input automaton, the one that we propose is linear. Then, we implement this simplified procedure as a part of the LASH toolset, a C library dedicated to finite-state automata. By means of some examples, we demonstrate that our implementation can handle automata having more that 100,000 states in a matter of a few minutes. Besides this, we introduce a variant of automata on linear orderings that, while preserving the original expressive power, allows the presence of epsilon transitions. The theoretical usefulness of these automata is illustrated by correcting an erroneous construction in the original proof of equivalence between automata on linear orderings and their associated rational expressions.

Résumé

Les automates sur ordres linéaires sont une forme d'automates finis introduite par Véronique Bruyère et Olivier Carton. Ces automates généralisent les concepts d'automates sur mots finis, infinis et transfinis. Les mots qu'ils acceptent sont indexés par des ordres linéaires, ces mots sont des fonctions associant à chaque élément d'un de ces ordres un symbole issu d'un alphabet fini. Ce travail de fin d'études traite de la problématique consistant à décider si un automate sur ordres linéaires arbitraire accepte au moins un mot. Dans le cadre de l'implémentation d'une procédure de décision pour la théorie monadique du premier ordre de l'ordre pour les nombres réels ou rationnels, nous concentrons notre étude sur le problème consistant à décider si un automate sur ordres linéaires donné accepte au moins un mot indexé par les réels ou les rationnels. Ce travail est basé sur une procédure de décision récemment obtenue par Bernard Boigelot, Pascal Fontaine et Baptiste Vergain. Alors que l'étape critique de la procédure originale a un coût quadratique en la taille de l'automate à étudier, son pendant dans la procédure que nous développons a une complexité linéaire. Ensuite, nous implémentons cette procédure simplifiée au sein de l'outil LASH, une librairie écrite en langage C dédiée aux automates finis. Via des exemples, nous montrons que notre procédure peut manipuler des automates ayant plus de 100 000 états en l'espace de quelques minutes. En parallèle, nous introduisons une variante des automates sur ordres linéaires qui, sans en altérer l'expressivité, permet la présence de transitions epsilon. L'intérêt théorique d'une telle extension est illustré par la correction d'une construction erronée présente dans la preuve originale de l'équivalence entre les automates sur ordres linéaires et les expressions rationnelles associées.

Acknowledgment

I would like to thank my advisor, Bernard Boigelot, who introduced the exciting world of automata to me. His endless enthusiasm, availability and advice are priceless. It wish to thank Pascal Fontaine, Baptiste Vergain and Alexis Bertrand, for their interest in my work and discussions that we had together. My friend Tom Clara deserves special thanks for the many hours spent together discussing our respective and common productions. I want to thank Pascal Fontaine and Michel Rigo for accepting the invitation to join my jury. Finally, I want to thank my family and friends for their continuous support.

Contents

Introduction

Finite-state automata are mathematical objects able to represent infinite sets with finite memory. Many decision procedures for logic fragments, in particular those containing arithmetic, are based on finite-state automata. The role of these automata is to represent the class of models of the input formula. Then, to decide whether the formula has at least one model, one performs a non-emptiness test on the language accepted by the associated automaton. For instance, in 1962, Büchi introduced automata on infinite words [8] to establish the decidability of the monadic second-order theory of one successor (S1S). Another logic for which a decision procedure can be based on automata is Presburger arithmetic, that is the additive first-order theory of integers [6].

Although automata are often considered as theoretical objects, they can serve as data structures used by computer programs efficiently implementing decision procedures. This work is a part of a project consisting in building a decision procedure for the monadic first-order theory of order on real or rational numbers [2]. The motivation is to provide a solver which can be added to a satisfiability modulo theory (SMT) solver. Such a solver consists in a computer program able to decide the satisfiability of formulas mixing several logic fragments. Two major applications of such solvers are computer system verification and automated theorem proving. Verification consists in modelling a computer system (and its environment) into a formal mathematical framework whose properties can be automatically analysed. A typical property is the absence of error in the execution of the program. If such an error is detected, it consists in a bug that can then be corrected by the developers of the program. Otherwise, under the hypotheses made at the modelling step, the program is correct.

The decision procedure for the monadic first-order theory of order on real or rational numbers relies on finite-state automata which are more expressive than the usual finite-word or Büchi automata. These automata are called automata on linear orderings, as the only restriction on the words which they read is that their letters can be placed on a line [7, 1]. This line can take many forms, including the real or the rational numbers.

This work focuses on testing whether a given automaton on linear orderings accepts a non-empty language. In particular, the main objectives are the design and implementation of an efficient procedure deciding whether the language accepted by an automaton on linear orderings contains at least one word indexed by the real or the rational numbers. This is done as a refinement of the procedure recently obtained by Bernard Boigelot, Pascal Fontaine and Baptiste Vergain [2]. A second master's thesis has been carried out in parallel by Tom Clara [12]. It tackles the problem of efficiently computing the automaton resulting from the union or the intersection of two arbitrary automata on linear orderings. Since both theses study the same objects, they share some chapters.

This document is structured as follows. Chapter 1, common with Tom Clara, formally presents automata on linear orderings according to [7, 1]. Then, in Chapter 2, common with Tom Clara as well, we slightly extend the existing formalism of automata on linear orderings so as to ease theoretical developments. As an application, we correct a (minor) mistake made by Alexis Bès and Olivier Carton [1] in their proof of equivalence between automata and their associated rational expressions. The results of this chapter have been turned into an article [3], co-authored by Tom Clara, Bernard Boigelot and myself. Chapter 4 reviews the state of the art in terms of non-emptiness test for automata on linear orderings. Then, we concentrate on the non-emptiness test for words indexed by real or rational numbers. In Chapter 4, we simplify the procedure of Bernard Boigelot et al. [2] from a theoretical point of view. In Chapter 5, we provide algorithms implementing the non-emptiness test and measure their practical performances. Finally, conclusions are drawn and perspectives are outlined. Appendix A, gives technical details about the practical implementation of automata on linear orderings, and is common with Tom Clara. In Appendix B, we outline the main steps of the whole decision procedure and highlight the importance of the operations of intersection, union and non-emptiness test.

Chapter 1

Automata on linear orderings

This chapter (common with Tom Clara) introduces *automata on linear orderings*, defined for the first time in [7] and later extended in [1]. In this chapter, we present automata on linear orderings. These finite-state automata can be seen as a broad generalization of finite-word, infinite-word and transfinite-word automata.

1.1 Linear orderings

In this part, we recall elementary notions on linear orderings. Much more detail can be found in [18].

A *linear ordering* is a set J equipped with an order relation $<_J$ which is

- total (for all $j, k, \ell \in J$, either $j <_J k$ or $k <_J j$),
- irreflexive ($j \not<_J j$),
- antisymmetric ($j <_J k$ implies $k \not<_J j$), and
- transitive ($j <_J k$ and $k <_J \ell$ imply $j <_J \ell$).

When clear from the context, $<_J$ will be abbreviated by $<$. Two elements $j, k \in J$ are said to be *consecutive* if $j < k$ and if there does not exist any $\ell \in J$ such that $j < \ell < k$. In this case, j is the *predecessor* of k and k is the *successor* of j. A linear ordering K is a *subordering* of J if $K \subseteq J$ and if the order relation $<_K$ is a restriction of $<_J$ to the elements of K.

Two linear orderings J_1 and J_2 are *isomorphic* if there exists an order-preserving bijection between them, i.e., if there exists a bijective mapping $\tau : J_1 \to J_2$ such that $k <_{J_1} \ell$ iff $\tau(k) <_{J_2} \tau(\ell)$. The *order type* of a linear ordering J is the class of all linear orderings that are isomorphic to J. The order types of the usual linear orderings $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ are denoted by ω, ζ, η and λ, respectively. Even though there exists a bijection between $\mathbb{Q}$ and $\mathbb{N}$ (since $\mathbb{Q}$ is a countable set), the order types ω and η are different, because such a bijection cannot be order-preserving. This is intuitively why $\mathbb{Q}$ and $\mathbb{N}$ are two very different linear orderings (see Examples 1 and 2). For the same reason, ω and ζ are different order types as well.

A linear ordering K is *dense* if it does not contain any pair of consecutive elements. It is *dense in J* if it is a subordering of J such that for any $i, j \in J$ such that $i < j$, one has $i < k < j$ for some $k \in K$. If K does not contain any dense infinite subordering, then K is said to be *scattered.*

Given two linear orderings J and K, we denote by $J + K$ the juxtaposition of J and K, i.e., the linear ordering L obtained via the disjoint union of J and K and in which the order relation is defined as follows:

- If $\ell_1, \ell_2 \in J$, then $\ell_1 <_L \ell_2$ iff $\ell_1 <_J \ell_2$.
- If $\ell_1, \ell_2 \in K$, then $\ell_1 <_L \ell_2$ iff $\ell_1 <_K \ell_2$.
- If $\ell_1 \in J$ and $\ell_2 \in K$, then $\ell_1 <_L \ell_2$.

More generally, for any linear ordering J, the sum $\sum_{j \in J} K_j$ (where K_j is a linear ordering for each $j \in J$) denotes the linear ordering whose elements are all the pairs (k, j) where $j \in J$ and $k \in K_j$. The order relation $(k_1, j_1) <_L (k_2, j_2)$ holds iff $j_1 <_J j_2$ (i.e., K_{j_1} comes before K_{j_2} in the juxtaposition) or $j_1 = j_2$ and $k_1 <_{K_{j_1}} k_2$ (i.e., k_1 and k_2 belong to the same linear ordering $K_{j_1} = K_{j_2}$, and k_1 is lower than k_2 in this linear ordering). Finally, we denote by $-J$ the linear ordering J in which the order relation has been reversed. In other words, $-J$ and J are the same set, but equipped with different order relations: $j <_J k$ iff $k <_{-J} j$.

A *cut* of a linear ordering J is an ordered pair of sets (K, L), where K and L partition J and $k <_J \ell$ for any $k \in K$ and any $\ell \in L$. The set of all the cuts of J is denoted by $\widehat{J}$ and can be linearly ordered by the order relation $<_{\widehat{J}}$, defined as $(K, L) <_{\widehat{J}} (K', L')$ iff $K \subsetneq K'$. The least element of $\widehat{J}$ is the *first cut* $\widehat{J}_{min} = (\emptyset, J)$. Similarly, the greatest element of $\widehat{J}$ is the *last cut* $\widehat{J}_{max} = (J, \emptyset)$. A cut that is not the first one or the last one is a *non trivial* cut, and the set of non trivial cuts is denoted by $\widehat{J}^*$. A cut $(K, L) \in \widehat{J}$ is a *gap* if $K \neq \emptyset$ has no greatest element and $L \neq \emptyset$ has no least element. The linear ordering J is *complete* if it has no gap. For all linear orderings J, it can be shown that $\widehat{J}$ is complete (see [7] for details).

Given the two linear orderings J and $\widehat{J}$, a new linear ordering $J \cup \widehat{J}$ can be defined by extending the order relations of J and $\widehat{J}$: the elements of J and the elements of $\widehat{J}$ are still ordered in the same way internally, and for any $j \in J$ and any cut $c = (K, L) \in \widehat{J}$, one has $j < c$ (resp. $c < j$) iff $j \in K$ (resp. $j \in L$). The linear ordering $J \cup \widehat{J}^*$ is defined analogously ($J \cup \widehat{J}^*$ is actually obtained by removing the least and the greatest elements from $J \cup \widehat{J}$). It can be shown that both $J \cup \widehat{J}$ and $J \cup \widehat{J}^*$ are complete linear orderings. The semantics of automata on linear orderings that we will introduce in Section 1.3 strongly relies on the fact that in $J \cup \widehat{J}$, any element $j \in J$ admits one successor $c_j^+ = (K \cup \{j\}, L)$ and one predecessor $c_j^- = (K, L \cup \{j\})$, where $K = \{k \in J \mid k < j\}$ and $L = \{\ell \in J \mid j < \ell\}$.

Example 1. *The linear ordering $\mathbb{N} + (-\mathbb{N})$ can be visually represented by the following line, in which the elements of the linear ordering are sorted by increasing order:*

$$0_{\mathbb{N}}, 1_{\mathbb{N}}, 2_{\mathbb{N}}, 3_{\mathbb{N}}, \ldots \quad \ldots, 3_{-\mathbb{N}}, 2_{-\mathbb{N}}, 1_{-\mathbb{N}}, 0_{-\mathbb{N}}$$

The notations $x_{\mathbb{N}}$ and $x_{-\mathbb{N}}$ are used to distinguish the elements of $\mathbb{N}$ from the elements of $-\mathbb{N}$. The linear ordering $\mathbb{N} + (-\mathbb{N})$ is scattered because each element has a predecessor and a successor (except $0_{\mathbb{N}}$ which has no predecessor and $0_{-\mathbb{N}}$ which has no successor), hence it does not contain any dense subordering. It is not complete, because the cut $(\mathbb{N}, -\mathbb{N})$ is a gap (since $\mathbb{N}$ does not have a greatest element and $-\mathbb{N}$ does not have a least element). Adding a single additional element between $\mathbb{N}$ and $-\mathbb{N}$ would make the ordering complete, because $\mathbb{N}$ and $-\mathbb{N}$ are already complete. Note that $+$ is not commutative: $-\mathbb{N} + \mathbb{N}$ is isomorphic to $\mathbb{Z}$ and differs completely from $\mathbb{N} + (-\mathbb{N})$.

Example 2. *Throughout this example, we use the notation $\left]\, a, b \,\right[_J$ where J is a subordering of $\mathbb{R}$ and $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$. This is a shorthand for $\{x \in J \mid a <_{\mathbb{R}} x \ \wedge \ x <_{\mathbb{R}} b\}$. As usual, the orientation of the square brackets (closed or open) is used to accept or reject the left bound a or the right bound b. For instance, $\left]\, a, b \,\right]_J$ denotes the set $\{x \in J \mid a <_{\mathbb{R}} x \ \wedge \ \neg(b <_{\mathbb{R}} x)\}$.*

The set $\mathbb{R}$ of real numbers (equipped with its natural order relation) is a dense linear ordering. Cuts of $\mathbb{R}$ consist of the first cut $(\emptyset, \mathbb{R})$, the last cut $(\mathbb{R}, \emptyset)$ and two infinite sets of non trivial cuts

$$\Big\{ \Big(\left]-\infty, r\right[_{\mathbb{R}}, \left[r, +\infty\right[_{\mathbb{R}} \Big) \mid r \in \mathbb{R} \Big\}$$

and

$$\Big\{ \Big(\left]-\infty, r\right]_{\mathbb{R}}, \left]r, +\infty\right[_{\mathbb{R}} \Big) \mid r \in \mathbb{R} \Big\}.$$

None of these cuts is a gap, hence $\mathbb{R}$ is a complete linear ordering. In contrast, the set $\mathbb{Q}$ of rational numbers is not a complete linear ordering. In addition to the non trivial cuts

$$\Big\{ \Big(\left]-\infty, q\right[_{\mathbb{Q}}, \left[q, +\infty\right[_{\mathbb{Q}} \Big) \mid q \in \mathbb{Q} \Big\}$$

and

$$\Big\{ \Big(\left]-\infty, q\right]_{\mathbb{Q}}, \left]q, +\infty\right[_{\mathbb{Q}} \Big) \mid q \in \mathbb{Q} \Big\}$$

there are uncountably many cuts of the form $\Big(\left]-\infty, r\right[_{\mathbb{Q}}, \left]r, +\infty\right[_{\mathbb{Q}} \Big)$ where $r \in \mathbb{R} \setminus \mathbb{Q}$. These cuts are gaps, because if r is irrational, the set $\left]-\infty, r\right[_{\mathbb{Q}}$ has no greatest element and the set $\left]r, +\infty\right[_{\mathbb{Q}}$ has no least element. Interestingly, the set of cuts of $\mathbb{Q}$ is thus uncountable, even though $\mathbb{Q}$ is a countable set.

1.2 Words on linear orderings

We now introduce the notions related to words on linear orderings, borrowed from [7].

Given a linear ordering J, a *word of length J* (also called *word indexed by J*) is a function w from J to a finite alphabet Σ (whose elements are *letters* or *symbols*). The letter associated with the element $j \in J$ is denoted by $w(j)$ and the length of w by $|w|$. Intuitively, if $|w|$ is a finite linear ordering, w is simply a finite word, in the classical sense. Setting $|w|$ to $\mathbb{N}$ or $\mathbb{Z}$ leads to the well-known infinite or bi-infinite words, respectively. The *empty word ε* is the word of length $\emptyset$. This shows that words on linear orderings consist of a broad generalization of the concepts of finite, infinite and bi-infinite words.

Two words w_1 and w_2 are *isomorphic* if there exists an order-preserving bijection $\tau : |w_1| \to |w_2|$ such that $w_1(j) = w_2(\tau(j))$ for all $j \in |w_1|$. The isomorphism between words is obviously an equivalence relation. In the sequel, we do not distinguish two words that are isomorphic, i.e., that belong to the same equivalence class.[1] Examples 3 and 4 show that this choice is natural. Formally, this amounts to working with the equivalence classes of words rather than with the words themselves. Still, we keep using the concept of word throughout this thesis, while keeping in mind that isomorphic words are undistinguishable.

Example 3. *Consider the word $w_1 : \{1, 2, 3\} \to \{a, b\}$ defined as*

$$w_1(1) = a, \;\; w_1(2) = b, \;\; w_1(3) = a$$

and the word $w_2 : \{4, 5, 6\} \to \{a, b\}$ defined as

$$w_2(4) = a, \;\; w_2(5) = b, \;\; w_2(6) = a.$$

These two words are finite words that are isomorphic, and they should of course be considered as identical, since they can both be written aba.

Example 4. *Let $\mathbb{N}_p$ be the set of natural numbers that are multiple of $p > 0$. On the alphabet $\{a\}$, two words w_1 and w_2 of respective lengths $\mathbb{N}_2$ and $\mathbb{N}_3$ are isomorphic. These two words are infinite words that contain only the symbol a, and they should thus intuitively be considered as identical.*

[1]Strictly speaking, the length of a word should therefore be an order type rather than a linear ordering.

The *product*, or *concatenation*, of two words w_1 and w_2 is defined as the word $w = w_1 \cdot w_2$ (sometimes just written $w_1 w_2$) of length $|w_1| + |w_2|$ such that

$$w(j) = \begin{cases} w_1(j) & \text{if } j \in |w_1| \\ w_2(j) & \text{if } j \in |w_2| \end{cases}$$

This operation is the same as for finite words, and can be generalized to multiple operands: for any linear ordering J, the product $\prod_{k \in J} w_k$ is the word w of length $\sum_{k \in J} |w_k|$ such that $w(j) = w_k(j)$ iff $j \in |w_k|$. Note that nothing prevents J to be dense in this definition.

As in the case of finite or infinite words, rational expressions can be defined for languages of words on linear orderings. A language (i.e., a class of words on linear orderings) is *rational* if it can be denoted by a rational expression. In the sequel, we present these rational expressions. Note that a rational expression *represents* a language but is not equal to a language. By convenience, we thus introduce the operator $\mathscr{L}(.)$ that converts a rational expression into the language that it represents.

Let Σ be a finite alphabet. We first define the elementary rational expressions, from which all the other rational expressions can be built inductively. These elementary rational expressions are:

- ε, that denotes the language $\mathscr{L}(\varepsilon) = \{\varepsilon\}$.
- $\varnothing$, that denotes the empty language $\mathscr{L}(\varnothing) = \{\}$.
- σ, that denotes the language $\mathscr{L}(\sigma) = \{\sigma\}$, for all $\sigma \in \Sigma$.

Next, we define how a rational expression can be built from a simpler one. Let X be a rational expression. As in the case of finite words, one can define the *finite iteration* X^* as the rational expression that denotes the language

$$\mathscr{L}(X^*) = \big\{w \mid \exists\, n \in \mathbb{N} \text{ and } w_1, w_2, \dots, w_n \in \mathscr{L}(X) \;\text{ such that }\; w = w_1 w_2 ... w_n\big\}. \tag{1.1}$$

Similarly, the ω*-iteration* X^ω represents the language

$$\mathscr{L}(X^\omega) = \left\{\prod_{i \in \mathbb{N}} w_i \mid w_i \in \mathscr{L}(X)\right\}. \tag{1.2}$$

Interestingly, the finite iteration and the ω-iteration are particular cases of a general iteration $X^{\mathcal{J}}$ that denotes the language

$$\mathscr{L}\big(X^{\mathcal{J}}\big) = \left\{\prod_{j \in J} w_j \mid w_j \in \mathscr{L}(X) \;\text{ and }\; J \in \mathcal{J}\right\}$$

where $\mathcal{J}$ is a class of linear orderings. This general iteration can be used to define concisely all the iterations that are needed to define rational languages of words on linear orderings:

- Setting $\mathcal{J}$ equal to the class of all finite linear orderings yields the finite iteration X^* whose meaning is already defined in 1.1.
- Setting $\mathcal{J}$ to $\{\omega\}$, i.e., the class of all the orderings isomorphic to $\mathbb{N}$, brings back the ω-iteration X^ω, that represents the language previously described in 1.2.
- The *ordinal iteration* $X^\sharp$ can be obtained by setting $\mathcal{J}$ to the class of all ordinals.
- The reverse ω-iteration $X^{-\omega}$ can be defined by setting $\mathcal{J}$ to $\{-\omega\}$ (the class of linear orderings isomorphic to $-\mathbb{N}$).

- Finally, the reverse ordinal iteration $X^{-\sharp}$ can be defined by setting $\mathcal{J}$ to the class of all reverse ordinals (the class of all the orderings J such that $-J$ is an ordinal).

We obviously do not introduce the reverse finite iteration, because $-J$ and J are isomorphic if J is a finite linear ordering. Note that we do not allow $\mathcal{J}$ to be the class of all linear orderings, because such a fully general iteration will be a particular case of the *diamond operator*, introduced later in this section.

Example 5. *The language denoted by the rational expression $(ab)^{\omega}(ba)^{-\omega}$ contains a single word (recall that we do not distinguish isomorphic words), of length $\mathbb{N} + (-\mathbb{N})$. On the alphabet $\{a, b\}$, this is the only word of this length which starts and ends by an a and in which two identical symbols are never consecutive (i.e., one cannot find two consecutive elements of $\mathbb{N} + (-\mathbb{N})$ that are mapped to the same symbol). In contrast, the language denoted by the rational expression $(ba)^{-\omega}(ab)^{\omega}$ contains a single word of length $\mathbb{Z}$. In this word, one can find two consecutive symbols a, corresponding to the last symbol of $(ba)^{-\omega}$ and the first symbol of $(ab)^{\omega}$.*

We now introduce the rational expressions that combine two simpler rational expressions X_1 and X_2. The rational expression $X_1 \cdot X_2$ denotes the *product* or the *concatenation* of the languages denoted by X_1 and X_2:

$$\mathscr{L}(X_1 \cdot X_2) = \Big\{ w_1 \cdot w_2 \mid w_1 \in \mathscr{L}(X_1) \text{ and } w_2 \in \mathscr{L}(X_2) \Big\}.$$

The rational expression $X_1 + X_2$ denotes the *union* of the languages represented by X_1 and X_2:

$$\mathscr{L}(X_1 + X_2) = \mathscr{L}(X_1) \cup \mathscr{L}(X_2).$$

We now introduce an operator that makes it possible to create words of arbitrary length. The rational expression $X_1 \diamond X_2$ (where $\diamond$ is the *diamond operator*) represents the following language:

$$\mathscr{L}(X_1 \diamond X_2) = \left\{ \prod_{j \in J \cup \widehat{J}^*} w_j \mid J \neq \emptyset \text{ and } w_j \in \mathscr{L}(X_1) \text{ if } j \in J \text{ and } w_j \in \mathscr{L}(X_2) \text{ if } j \in \widehat{J}^* \right\}.$$

Intuitively, to generate a word in $\mathscr{L}(X_1 \diamond X_2)$ of length $J \cup \widehat{J}^*$, one needs to choose a non empty linear ordering J and to assign a word from $\mathscr{L}(X_1)$ to each element $j \in J$ and a word from $\mathscr{L}(X_2)$ to each element $j \in \widehat{J}^*$. In the particular case where $\mathscr{L}(X_2)$ contains only the empty word, $X_1 \diamond X_2$ denotes the language

$$\mathscr{L}(X_1 \diamond \varepsilon) = \mathscr{L}(X_1^{\diamond}) = \left\{ \prod_{j \in J} w_j \mid J \neq \emptyset \text{ and } w_j \in \mathscr{L}(X_1) \right\}.$$

The rational expression $X^{\diamond}$ therefore serves as a shorthand for a general iteration based on any non-empty linear ordering.

Example 6. *The language denoted by $\varepsilon + (a + b)^{\diamond}$ contains every possible word on the alphabet $\{a, b\}$.*

A last rational expression has been introduced in [1] when generalizing the theory of automata on linear orderings to dense orderings. The *shuffle* of $X_1, X_2, \ldots, X_n$ is written $sh(X_1, X_2, \ldots, X_n)$. A word w belongs to $\mathscr{L}(sh(X_1, X_2, \ldots, X_n))$ if w can be written as $\prod_{j \in J} w_j$ where the ordering J has the two following properties:

- J is a non empty, dense and complete linear ordering without first and last element.
- J can be partitioned into n sub-orderings $J_1, J_2, \ldots, J_n$ that are all dense in J, and such that $w_j \in \mathscr{L}(X_i)$ if $j \in J_i$.

Intuitively, the shuffle operator mixes up languages associated to $X_1, X_2, \ldots, X_n$ in a dense fashion.

Even though a rational expression is not equal to the set it represents, we will allow ourselves to forget this distinction, by convenience. For instance, we will often write sentences like “the language a^ω contains a single word”, even though a^ω is not a language, strictly speaking.

Example 7. *The word $w_1 : \mathbb{R} \to \{a, b\}$ such that*

$$\forall j \in \mathbb{R} \quad w_1(j) = \begin{cases} a & \text{if } j \in \mathbb{Q} \\ b & \text{if } j \in \mathbb{R} \setminus \mathbb{Q} \end{cases}$$

belongs to the language $sh(a, b)$, since:

- *Its length $\mathbb{R}$ is non empty, dense and complete. Moreover, $\mathbb{R}$ does not have a first or a last element.*
- *$\mathbb{Q}$ is dense in $\mathbb{R}$ and $\mathbb{R} \setminus \mathbb{Q}$ as well.*

The word $w_2 :]-1, 1[\to \{a, b\}$ defined as

$$\forall j \in]-1, 1[: \quad w_2(j) = \begin{cases} a & \text{if } j \in \mathbb{Q} \\ b & \text{if } j \in \mathbb{R} \setminus \mathbb{Q} \end{cases}$$

is isomorphic to w_1, and we do not make any distinction between them. Note that it is not immediate to find an order-preserving bijection between $\mathbb{R}$ and $]-1, 1[$ that maps each rational of $\mathbb{R}$ to a rational of $]-1, 1[$ and each irrational of $\mathbb{R}$ to an irrational of $]-1, 1[$, but such a bijection does exist. For instance, consider the function

$$\tau(j) = \begin{cases} \left(1 - \left(\frac{1}{2}\right)^{\lfloor j \rfloor} + (j - \lfloor j \rfloor)\left(\frac{1}{2}\right)^{\lceil j \rceil}\right) & \text{if } j \geq 0 \\ -\tau(-j) & \text{if } j < 0 \end{cases}$$

where $\lfloor x \rfloor$ denotes the greatest integer lower or equal to x, and $\lceil x \rceil$ the least integer larger or equal to x. This function τ is piecewise linear, and each linear piece has a rational slope and a rational intercept. This guarantees that the image of a rational (resp. an irrational) is rational (resp. irrational). The function τ is depicted in Figure 1, in which the colours are used to highlight the different linear pieces of the function.

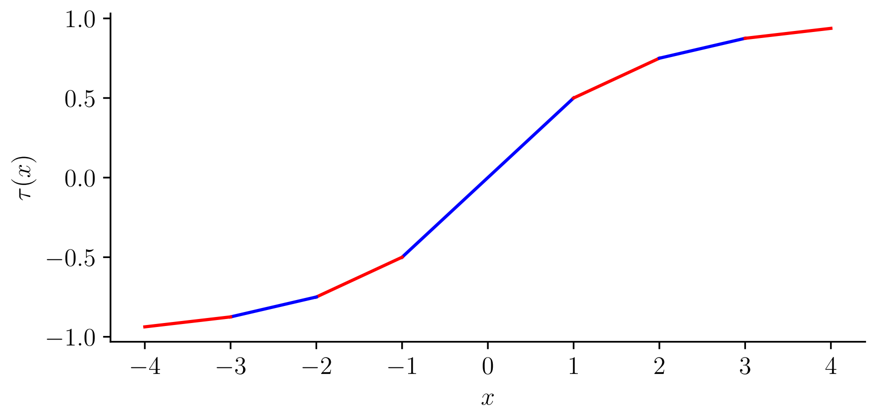


Figure 1: Bijection τ between $\mathbb{R}$ and $]1, 1[$. This bijection preserves the order, and maps each rational of $\mathbb{R}$ to a rational of $]-1, 1[$ and each irrational of $\mathbb{R}$ to an irrational of $]-1, 1[$.

Note that the language $sh(a,b)$ contains words whose length is not isomorphic to $\mathbb{R}$. For instance, the word

$$w_3 = \prod_{j\in\mathbb{R}} a \cdot w_1 \cdot a$$

has the length $\sum_{j\in\mathbb{R}}\{1\} + \mathbb{R} + \{1\}$, which is not isomorphic to $\mathbb{R}$ and yet does belong to $sh(a,b)$. The presence of the singletons $\{1\}$ around each copy of $\mathbb{R}$ is necessary to make $|w_3|$ complete.

Example 8. *The language $sh(a,b)$ does not contain any word of length $\mathbb{Q}$, since $\mathbb{Q}$ is not a complete linear ordering. However, the language $sh(a,\varepsilon)$ contains the word*

$$w = \prod_{j\in\mathbb{R}} w_j$$

where

$$w_j = \begin{cases} a & \text{if } j \in \mathbb{Q} \\ \varepsilon & \text{if } j \in \mathbb{R}\setminus\mathbb{Q} \end{cases} \qquad \forall j \in \mathbb{R}.$$

This word can be rewritten as

$$w = \prod_{j\in\mathbb{Q}} a \ ,$$

whose length is obviously $\mathbb{Q}$.

1.3 Automata on linear orderings

In this section, we present automata on linear orderings as defined by [7] and extended by [1]. They are a form of finite-state automata able to read words indexed by linear orderings as defined in Section 1.2. Their expressive power has been shown to coincide with the one of rational expressions [7, 1].

Definition 1. *An* automaton on linear orderings *is a tuple (Q,Σ,Δ,I,F) where:*

- *Q is a finite set of* states.
- *Σ is a finite* alphabet.
- *$\Delta \subseteq (Q\times\Sigma\times Q)\cup\left(Q\times 2^Q\right)\cup\left(2^Q\times Q\right)$ is a* transition relation.
- *$I \subseteq Q$ is a set of* initial states.
- *$F \subseteq Q$ is a set of* final states.

The transition relation Δ contains transitions of three types. Those belonging to the set $Q\times\Sigma\times Q$ are called *successor transitions* and can be written as $q_1 \xrightarrow{a} q_2$, for $q_1, q_2 \in Q$ and $a \in \Sigma$. We respectively call q_1, q_2 and a the *origin state*, the *destination state*, and the *label* of the transition. Transitions belonging to $Q\times 2^Q$ are called *right-limit transitions* and are noted $q \to P$, for $q \in Q$ and $P \subseteq Q$. In that case, q is the *origin state* of the transition. Symmetrically, *left-limit transitions* are those that belong to $2^Q \times Q$ and can be written as $P \to q$. In that case, q is the *destination state* of the transition.

We now present the semantics of such automata. Let $\mathcal{A}$ be an automaton on linear orderings.

Definition 2. *A* run *of $\mathcal{A}$ reading a word w of length J is a mapping $\rho : \widehat{J} \to Q$ such that:*

- *$\rho(\widehat{J}_{min}) \in I$.*
- *$\rho(\widehat{J}_{max}) \in F$.*

- *For every pair of cuts $c_1 = (K_1, L_1)$ and $c_2 = (K_2, L_2)$ such that c_2 is the successor of c_1 in $\widehat{J}$, $(\rho(c_1), w(j), \rho(c_2)) \in \Delta$, where $j \in J$ is such that $K_2 = K_1 \cup \{j\}$.*
- *For every cut $c \neq \widehat{J}_{min}$ which does not have any predecessor, $(\lim_{c^-} \rho, \rho(c)) \in \Delta$, where $\lim_{c^-} \rho$ is the* left-limit set *of ρ in c, defined as*
$$\lim_{c^-} \rho = \big\{ q \in Q \mid (\forall c_1 < c)(\exists c_2)(c_1 < c_2 < c \wedge \rho(c_2) = q) \big\}.$$
- *For every cut $c \neq \widehat{J}_{max}$ which does not have any successor, $(\rho(c), \lim_{c^+} \rho) \in \Delta$, where $\lim_{c^+} \rho$ is the* right-limit set *of ρ in c, defined as*
$$\lim_{c^+} \rho = \big\{ q \in Q \mid (\forall c_1 > c)(\exists c_2)(c < c_2 < c_1 \wedge \rho(c_2) = q) \big\}.$$

The word w is *accepted* by the automaton $\mathcal{A}$ iff there exists a run reading w in $\mathcal{A}$. The class of all words accepted by some automaton forms its *accepted language*. A state $q \in Q$ is *reachable* if it is reachable from an initial state, i.e., if there exists a word that can be read by starting in an initial state and by ending in q. Similarly, a state is *co-reachable* if a final state can be reached from it.

Intuitively, a run maps cuts to states, starting from an initial state and ending in a final one. Each pair of successive cuts corresponds to a unique element of the word's length. The run reads the letter associated to that element by taking a successor transition labelled accordingly.

Whenever a cut (different from the last one) does not have a successor, a right limit transition is used to reach the state to which it is mapped. The semantics is very close to a Muller acceptance condition [15]. That is, a jump can be followed after visiting exactly infinitely often some set of states. Left-limit transitions are symmetric.

Before illustrating the previous notions with the help of some examples, we define how automata on linear orderings are represented graphically. As usual with finite-state automata, we make use of a directed graph, whose vertices are states and edges are successor transitions. If there are several successor transitions sharing the same origin and destination states, they are grouped together into a single edge labelled by comma separated letters. Limit transitions are represented as annotations and can be grouped as well. For instance, given $q_0, q_1, q_2 \in Q$ and $P \subseteq Q$, the notation $q_0 \to P \to q_1, q_2$ is shorthand for $q_0 \to P$, $P \to q_1$, and $P \to q_2$. We will often say that P is the limit set of these limit transitions, even though this might be misleading (since a limit set must be a left-limit set $\lim_{c^-} \rho$ or a right-limit set $\lim_{c^+} \rho$ of the run ρ in c, as defined in Definition 2). Depending on the context, a limit set can therefore be the set of states associated to a limit transition in an automaton on linear orderings, or the limit set of a run ρ at a cut c.

Example 9. *The automaton in Figure 2a accepts the language $(ab)^\omega$, that contains a single word $w : \mathbb{N} \to \{a, b\}$ such that $w(n) = a$ if n is even and $w(n) = b$ if n is odd. Indeed, from the only initial state q_0 a letter a must be read, immediately followed by a letter b, itself immediately followed by the same sequence. In order to reach the only final state q_2, one must repeat that pattern infinitely often in order to follow the only limit transition $\{q_0, q_1\} \to q_2$. After this state has been reached, no other transition can be taken. If the reading of the whole word has come to an end, it is accepted. In this very simple setting this automaton behaves like a Muller automaton, as previously hinted.*

Example 10. *The automaton of Figure 2b can be understood as two copies of the previous automaton. The second copy has its transitions flipped and is concatenated to the first one. Therefore, its accepted language is $(ab)^\omega (ba)^{-\omega}$. Recall from Example 1 that the ordering indexing that language has a gap. This gap can be easily identified by the presence of state q_2 which is the origin of a right-limit transition and the destination of a left-limit transition. The non trivial cut $(\mathbb{N}, -\mathbb{N})$ is mapped to that state as it neither has a predecessor nor a successor.*

Example 11. *The automaton in Figure 2c accepts the language $(a^\omega)^{-\omega}$. Let us first analyse states q_1 and q_2, together with the associated successor transition from q_1 to itself and the left-limit transition from limit set $\{q_1\}$ to state q_2. For reasons very similar to those discussed in Example 9, this part reads words of length $\mathbb{N}$ whose elements are all mapped to the letter a (i.e., this part reads the language a^ω). Then, to reach that part from the initial state, one must take a right-limit transition towards a limit set containing q_1 and q_2. Exactly as visiting the loop formed by states q_3 and q_4 allowed to take the left-limit transition leading to a reverse ω-iteration in the previous example, the only way for a run to take this limit transition is then to visit infinitely often states q_1 and q_2 by reading a^ω-words infinitely often. These nested limit sets therefore lead to the doubly ω-iterated language announced: $(a^\omega)^{-\omega}$.*

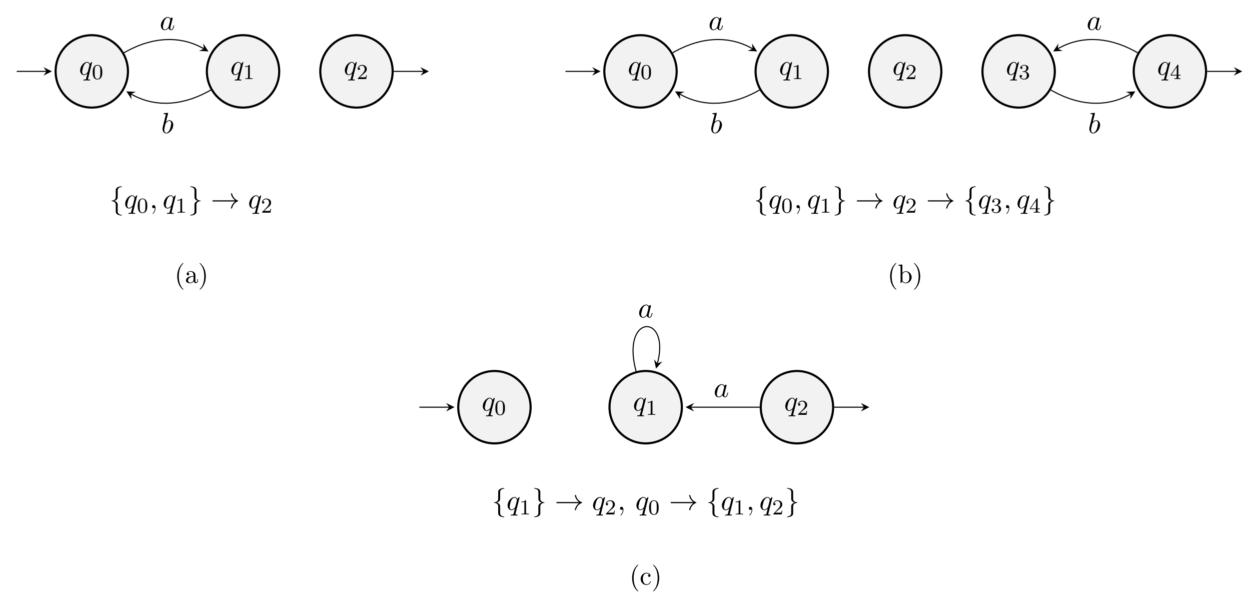


Figure 2: Automata on linear orderings accepting words of scattered lengths.

Example 12. *The automaton of Figure 3a accepts the language $sh(a, b)$. Recall from Example 7 that this language forms a dense and complete mixture of letters a and b, without a first or a last element. This is exactly what is enforced by this automaton. Indeed, from the initial state, one must take a right-limit transition, which requires to visit exactly infinitely often states q_1, q_2, q_3 and q_4. Moreover, states q_1 and q_3 can only be reached by following left-limit transitions. From these states, the only possible action is to follow a successor transition respectively labelled by a or b, which lead to states q_2 and q_3 respectively, themselves leading to the limit set by means of right-limit transitions. From this short analysis, several interesting things can be noticed:*

- *The ordering is complete since there is no gap.*
- *No letter admits a predecessor or a successor.*
- *The ordering does not have a first nor a last element.*
- *Readings of letters a and b have to be densely interleaved in order to satisfy requirements of limit transitions.*

This coincides with the definition of a shuffle.

Example 13. *Conversely, the automaton in Figure 3b accepts the language $sh(a, \varepsilon)$. Recall from Example 8 that such a language contains words indexed by $\mathbb{Q}$, which densely contain infinitely many gaps. The main difference with the previous example is that the successor transition formerly reading the letter b is now replaced by a single state, whose role is to be mapped to gaps.*

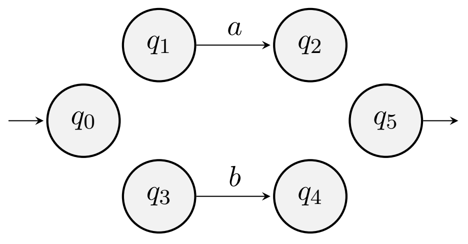
Example 14. *The automaton in Figure 3c accepts the language $sh\,(a+b)$. To give the intuition behind this, we show that every word consisting of a mapping from the set of real numbers to the alphabet $\{a,b\}$ is accepted by this automaton. Consider the function $\rho : \widehat{\mathbb{R}} \to \{q_0, q_1\}$ defined as*

$$\rho : \begin{cases} (\emptyset, \mathbb{R}) \mapsto q_0 \\ (\mathbb{R}, \emptyset) \mapsto q_1 \\ \Big(\left]-\infty, r\right[_{\mathbb{R}}, \left[r, +\infty\right[_{\mathbb{R}} \Big) \mapsto q_1, \quad \forall r \in \mathbb{R} \\ \Big(\left]-\infty, r\right]_{\mathbb{R}}, \left]r, +\infty\right[_{\mathbb{R}} \Big) \mapsto q_0, \quad \forall r \in \mathbb{R}. \end{cases}$$

Let us list properties of this function, to show that it complies with Definition 2.

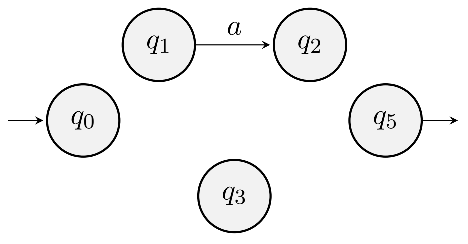
- *The first cut is mapped to an initial state and the last cut is mapped to a final state.*
- *Each pair of cuts c_1, c_2 where c_1 is the predecessor of c_2 is mapped to states connected by a successor transition reading the letter a or the letter b.*
- *Every cut of the form $\Big(\left]-\infty, r\right[_{\mathbb{R}}, \left[r, +\infty\right[_{\mathbb{R}} \Big)$ is not the first one, does not have a predecessor and every cut before it is mapped to q_0 or q_1.*
- *Every cut of the form $\Big(\left]-\infty, r\right]_{\mathbb{R}}, \left]r, +\infty\right[_{\mathbb{R}} \Big)$ is not the last one, does not have a successor and every cut after it is mapped to q_0 or q_1.*

Intuitively, this automaton consists of the one in Figure 3a, where states indexed by an even number are merged into state q_0 and symmetrically, states indexed by an odd number are merged into state q_1.



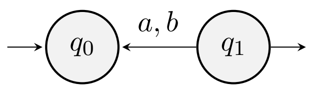
$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3, q_5$

(a)



$q_0, q_2, q_3 \to \{q_1, q_2, q_3\} \to q_1, q_3, q_5$

(b)



$q_0 \to \{q_0, q_1\} \to q_1$

(c)

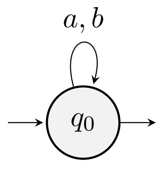
Figure 3: Automata on linear orderings accepting words of infinite, dense lengths.

Example 15. *The automaton in Figure 4 accepts any word indexed by a linear ordering on the alphabet $\{a,b\}$. Indeed, let us take a linear ordering J and show that the function $\rho : \widehat{J} \to \{q_0\}$ is a run reading a word of length J on the alphabet $\{a,b\}$, according to Definition 2.*

- *The first cut of J is mapped to an initial state and the last cut of J is mapped to a final state.*

- *For any pair of cuts $c_1, c_2 \in \widehat{J}$ such that c_1 is the predecessor of c_2, a successor transition from q_0 to itself can be followed to read the letter a or b.*
- *Every cut which is not the first one and which does not have a predecessor can be mapped to q_0 since every cut before it is mapped to q_0 as well, hence the left-limit transition $\{q_0\} \to q_0$ can be followed.*
- *Every cut which is not the last one and which does not have a successor can be mapped to state q_0 since every cut after it are mapped to q_0 as well, hence the right-limit transition $q_0 \to \{q_0\}$ can be followed.*

Therefore, this automaton accepts every word on the alphabet $\{a, b\}$.



$$q_0 \to \{q_0\} \to q_0$$

Figure 4: Automaton accepting every word on the alphabet $\{a, b\}$.

Chapter 2

Epsilon automata on linear orderings

The content of this chapter has been turned into an article co-authored by Bernard Boigelot, Tom Clara and myself [3]. In this chapter, we define a variant of automata on linear orderings that may contain epsilon transitions (i.e., transitions with an empty label) in their transition relation. With the appropriate choice of semantics, this new form of automata (that we call epsilon automata on linear orderings) keeps the same expressive power as automata on linear orderings (i.e., automata introduced in Definition 1). The use of epsilon transitions can significantly simplify some theoretical developments involving automata on linear orderings. The main motivation behind this work actually comes from the discovery of a (minor) error in [1] that, in our opinion, results from a construction made unnecessarily difficult by the lack of epsilon transitions. We illustrate the usefulness of epsilon automata by using them for correcting this erroneous construction.

2.1 Epsilon transitions

In this section, we motivate the need for epsilon transitions in automata on linear orderings, and introduce them with the help of an extended syntax and semantics.

2.1.1 Motivation

We mentioned in Chapter 1 that the languages that can be accepted by automata on linear orderings correspond exactly to those that can be described by rational expressions. This result has first been proved in [7] with a restriction to *countable and scattered* words, i.e., words w for which $|w|$ is a countable ordering that does not contain a dense infinite sub-ordering. It is then extended to unrestricted words in [1].

Recall from Chapter 1 that the rational expression $X_1 \diamond X_2$ represents the language

$$\mathscr{L}(X_1 \diamond X_2) = \left\{ \prod_{j \in J \cup \widehat{J}^*} w_j \mid J \neq \emptyset \text{ and } w_j \in \mathscr{L}(X_1) \text{ if } j \in J \text{ and } w_j \in \mathscr{L}(X_2) \text{ if } j \in \widehat{J}^* \right\}.$$

In the remainder of this chapter, it will be more convenient to only deal with languages, rather than with rational expressions. Therefore, we start by formally redefining the symbol $\diamond$ as a binary operator acting on languages of words indexed by linear orderings.

Definition 3. *Let L_1, L_2 be languages of words indexed by linear orderings, sharing the same alphabet Σ. The language $L = L_1 \diamond L_2$ contains all the words of the form $w = \prod_{j \in J \cup \widehat{J}^*} w_j$, where J is any non-empty linear ordering, and one has $w_j \in L_1$ for each $j \in J$ and $w_j \in L_2$ for each $j \in \widehat{J}^*$.*

Intuitively, $L_1 \diamond L_2$ is obtained by considering all the non-empty linear orderings J, and by replacing all their elements by words from L_1 and all their non-extreme cuts by words from L_2.

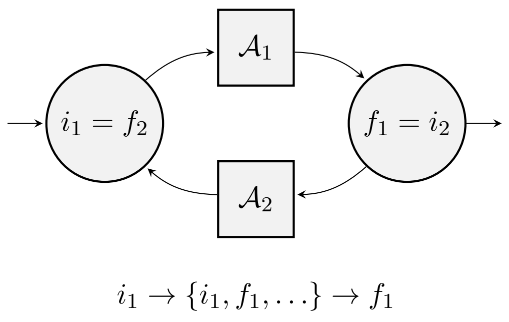


Figure 5: Automaton accepting $L_1 \diamond L_2$.

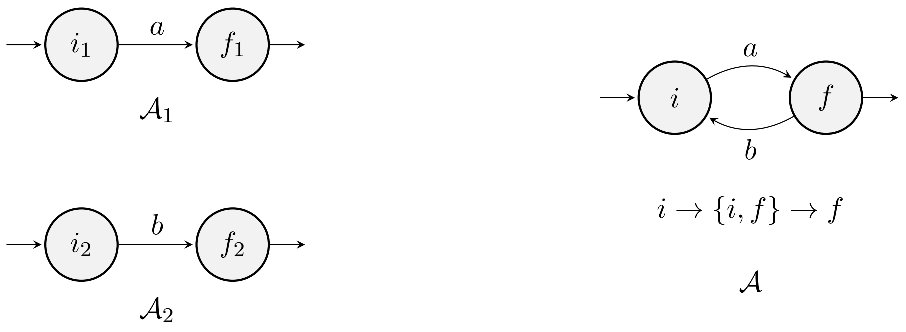


Figure 6: Automata accepting $\{a\}$, $\{b\}$, and (erroneously) $\{a\} \diamond \{b\}$.

A construction building an automaton $\mathcal{A}$ accepting $L_1 \diamond L_2$ from automata $\mathcal{A}_1$ and $\mathcal{A}_2$ accepting respectively L_1 and L_2 is outlined in [7, 1]. It consists in first expressing each $\mathcal{A}_k$, for $k \in \{1, 2\}$, as a *normalised automaton*, which has a single initial state i_k and a single final state f_k such that $i_k \neq f_k$. In addition, i_k (resp. f_k) cannot be the destination (resp. the origin) of a successor or limit transition.

The next step is to combine $\mathcal{A}_1$ and $\mathcal{A}_2$ to form the automaton $\mathcal{A}$. The combination scheme is shown in Figure 5, in the particular case where neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ accepts the empty word. In this construction, the limit transitions of the resulting automaton are those of $\mathcal{A}_1$ and $\mathcal{A}_2$, with additional transitions $i_1 \to P$ and $P \to f_1$ for all supersets P of $\{i_1, f_1\}$.

This construction is proved to be correct in [1], for the particular case of words restricted to be indexed by countable and scattered linear orderings. The claim made in [7] that it also applies to the general case is unfortunately invalid. Consider for instance the automata in Figure 6, accepting the languages $L_1 = \{a\}$ and $L_2 = \{b\}$. Let us show that the resulting automaton $\mathcal{A}$ accepts a language that differs from $L_1 \diamond L_2$. Indeed, $\mathcal{A}$ accepts in particular the word $w = \prod_{j \in \mathbb{R}} b$, i.e., the word indexed by $\mathbb{R}$ that only contains occurrences of the symbol b: a run ρ reading w can be obtained by setting $\rho((\emptyset, \mathbb{R})) = i$, $\rho((\mathbb{R}, \emptyset)) = f$, and for every $j \in \mathbb{R}$, $\rho((\mathbb{R}_{<j}, \mathbb{R}_{\geq j})) = f$ and $\rho((\mathbb{R}_{\leq j}, \mathbb{R}_{>j})) = i$ (remember Example 14). It is clear however that w does not belong to $\{a\} \diamond \{b\}$, since a occurs in every word of this language.

The error is caused by the fact that the construction in Figure 5 makes it possible to visit infinitely often all the states of $\mathcal{A}$ without ever following the successor transition labelled by a. This situation only happens when the limit transitions added by the construction can be followed without visiting a state of $\mathcal{A}_1$ distinct from i_1 and f_1.

A simple correction of this problem would thus consist in making sure that an internal state of $\mathcal{A}_1$ must necessarily be visited in order to enable the added limit transitions. However, this cannot be achieved with our current definition of automata on linear orderings, since it follows from Definitions 1 and 2 that, in an automaton accepting $\{a\}$, there must be a successor transition from an initial state to a final one, labelled by a. This motivates the need for a more general form of automaton on linear orderings, offering the possibility of defining internal states linked by transitions with an empty label. We introduce such automata in the next section. The correction of the construction of an automaton accepting $L_1 \diamond L_2$ is addressed in Section 2.2.

2.1.2 Epsilon automata on linear orderings

We define our extended form of automata over linear orderings as follows.

Definition 4. *An* epsilon automaton on linear orderings *is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where the set of states Q, the alphabet Σ, the sets of initial and final states I and F are as in Definition 1, and the relation transition Δ is of the form $\Delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times Q) \cup (Q \times 2^Q) \cup (2^Q \times Q)$.*

Compared with Definition 1, the only difference is the possibility of having successor transitions (q_1, ε, q_2) with an empty label ε.

Designing an appropriate semantics for epsilon automata is not immediate. Following a successor transition labelled by ε should naturally make a run move from a state to another without reading any symbol in its accepted word. The question is to decide whether following an infinite combination of epsilon transitions should enable limit transitions. In the positive case, adding the transition (q_1, ε, q_1) to the automaton in Figure 7 would make it also accept all finite words over the alphabet $\{a, b\}$. In the negative case, its accepted language would not be affected. Our choice is to opt for the latter approach,

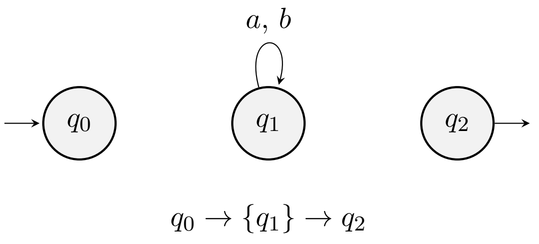


Figure 7: Automaton accepting all the bi-infinite words on the alphabet $\{a, b\}$.

that conservatively minimises the impact of epsilon transitions on the semantics of automata. This leads to a semantics in which the states visited by a run reading a word w are still associated to the cuts of $|w|$, but with the property that following epsilon transitions does not modify the current cut.

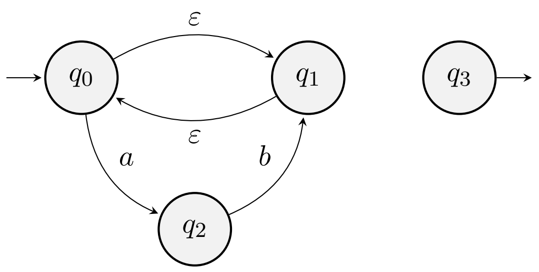
Formally, given an epsilon automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, we define the set $S \subseteq Q \times 2^Q \times Q$ of all the triples (q_1, U, q_2) such that $\mathcal{A}$ admits a finite path from q_1 to q_2 visiting exactly the states in U, entirely composed of successor transitions labelled by ε. In other words, such a path must be of the form $s_0 \xrightarrow{\varepsilon} s_1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s_n$ with $n \geq 0$, $s_0 = q_1$, $s_n = q_2$, $U = \{s_0, s_1, \ldots, s_n\}$, and for every $0 \leq k < n$, $(s_k, \varepsilon, s_{k+1}) \in \Delta$. For each $\sigma = (q_1, U, q_2) \in S$, we define $\mathit{first}(\sigma) = q_1$, $\mathit{last}(\sigma) = q_2$, and $\mathit{states}(\sigma) = U$.

Definition 5. *A* run *of $\mathcal{A}$ reading a word w is a mapping $\rho : \widehat{J} \to S$, where $J = |w|$, that satisfies the following conditions:*

- $\mathit{first}(\rho(\widehat{J}_{min})) \in I$.
- $\mathit{last}(\rho(\widehat{J}_{max})) \in F$.

- *For every* $c_1 = (K_1, L_1)$, $c_2 = (K_2, L_2) \in \widehat{J}$ *such that* $K_2 = K_1 \cup \{j\}$ *for some* $j \in J$, *one has* $(\mathit{last}(\rho(c_1)), w(j), \mathit{first}(\rho(c_2))) \in \Delta$.
- *For every* $c \neq \widehat{J}_{min} \in \widehat{J}$ *that does not have a predecessor, one has* $\lim_{c^-} \rho \to \mathit{first}(\rho(c)) \in \Delta$, *where* $\lim_{c^-} \rho = \{q \in Q \mid (\forall c_1 < c)(\exists c_2)(c_1 < c_2 < c \wedge q \in \mathit{states}(\rho(c_2)))\}$.
- *For every* $c \neq \widehat{J}_{max} \in \widehat{J}$ *that does not have a successor, one has* $\mathit{last}(\rho(c)) \to \lim_{c^+} \rho \in \Delta$, *where* $\lim_{c^+} \rho = \{q \in Q \mid (\forall c_1 > c)(\exists c_2)(c < c_2 < c_1 \wedge q \in \mathit{states}(\rho(c_2)))\}$.

Intuitively, instead of mapping the cuts of $|w|$ onto states of the automaton as in Definition 2, those cuts are now associated with finite paths of epsilon successor transitions, in which the only relevant information is their origin and destination states, together with the set of all the states that they visit.



$$\{q_0, q_1\} \to q_3,\ \{q_0, q_1, q_2\} \to q_3$$

Figure 8: Example of epsilon automaton on linear orderings.

Example 16. *An example of epsilon automaton on linear orderings is given in Figure 8. This automaton accepts the language* $(ab)^\omega$, *composed of a single word* w *mapping* $\mathbb{N}$ *onto* $\{a, b\}$, *such that* $w(j) = a$ *for even values of* j, *and* $w(j) = b$ *for odd* j. *Note that this automaton does not accept the empty word. Indeed, the empty word has a single cut* $(\emptyset, \emptyset)$, *and this cut cannot be mapped to a triple* σ *such that* $first(\sigma)$ *is initial and* $last(\sigma)$ *is final, since the final state* q_3 *cannot be reached from the initial state* q_0 *by following only* ε*-transitions.*

2.1.3 Expressiveness

We now establish that, with the choices made in Section 2.1.2, adding epsilon transitions to automata on linear orderings does not increase their expressive power.

Theorem 1. *A language of words indexed by linear orderings can be accepted by an epsilon automaton on linear orderings iff it can be accepted by an automaton on linear orderings.*

Proof. One direction is immediate: by Definitions 1 and 4, an automaton on linear orderings $\mathcal{A}$ can also be seen as an epsilon automaton $\mathcal{A}_\varepsilon$ without any epsilon transition. Every run ρ of $\mathcal{A}$ can be turned into a run of $\mathcal{A}_\varepsilon$ reading the same word w, by replacing $\rho(c)$ by $(\rho(c), \{\rho(c)\}, \rho(c))$ for each $c \in \widehat{|w|}$.

For the other direction, we prove a stronger result by providing an algorithm that turns an epsilon automaton $\mathcal{A}_\varepsilon$ into an equivalent automaton $\mathcal{A}$ on linear orderings. Let $\mathcal{A}_\varepsilon = (Q_\varepsilon, \Sigma_\varepsilon, \Delta_\varepsilon, I_\varepsilon, F_\varepsilon)$. The automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ is built as follows:

- $Q \subseteq Q_\varepsilon \times 2^{Q_\varepsilon} \times Q_\varepsilon$ is the set of all triples (q_1, U, q_2) such that $\mathcal{A}_\varepsilon$ admits a finite path of epsilon successor transitions from q_1 to q_2, visiting exactly the states that belong to U. (This corresponds to the definition of the set S in Section 2.1.2).
- $\Sigma = \Sigma_\varepsilon$.

- Δ contains
  - all $(\sigma_1, a, \sigma_2) \in Q \times \Sigma \times Q$ such that $(\mathit{last}(\sigma_1), a, \mathit{first}(\sigma_2)) \in \Delta_\varepsilon$,
  - all $(P, \sigma) \in 2^Q \times Q$ such that $\left(\bigcup_{\sigma' \in P} \mathit{states}(\sigma'), \mathit{first}(\sigma)\right) \in \Delta_\varepsilon$,
  - all $(\sigma, P) \in Q \times 2^Q$ such that $\left(\mathit{last}(\sigma), \bigcup_{\sigma' \in P} \mathit{states}(\sigma')\right) \in \Delta_\varepsilon$.
- $I = \{\sigma \in Q \mid \mathit{first}(\sigma) \in I_\varepsilon\}$.
- $F = \{\sigma \in Q \mid \mathit{last}(\sigma) \in F_\varepsilon\}$.

It follows from Definition 5 that every run of $\mathcal{A}_\varepsilon$ describes a run of $\mathcal{A}$ reading the same word. □

The algorithm given in the proof of Theorem 1 can incur an exponential blowup in the number of states of the automaton, and a double exponential blowup in the number of limit transitions. This is not problematic, since we expect epsilon automata on linear orderings to be mainly useful for theoretical developments.

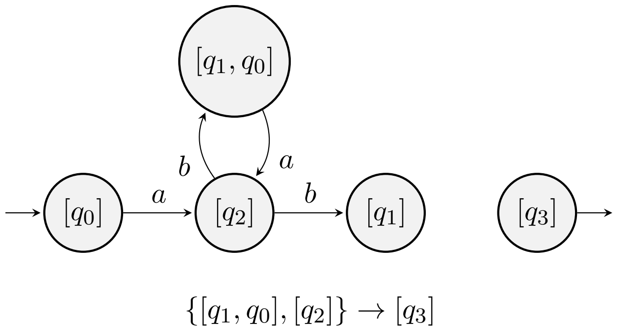
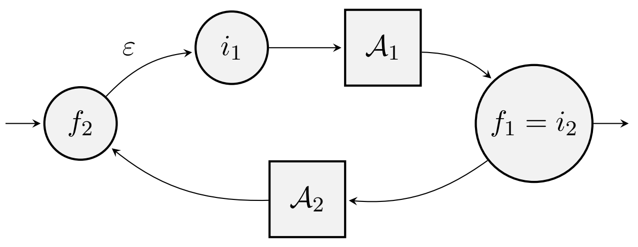


Figure 9: Automaton after eliminating epsilon transitions.

We illustrate this result by showing in Figure 9 the construction of an automaton on linear orderings equivalent to the one in Figure 8. We use the notation $[q_i]$ as shorthand for $(q_i, \{q_i\}, q_i)$, for $i \in \{0, 1, 2, 3\}$, and write $[q_1, q_0]$ in place of $(q_1, \{q_0, q_1\}, q_0)$. To keep the picture simple, some states and transitions that can not be visited or followed by any run are omitted.

2.2 Application

We are now ready to correct the construction of an automaton accepting $L_1 \diamond L_2$ with the help of epsilon automata. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be automata (either epsilon automata or plain automata on linear orderings) accepting respectively L_1 and L_2. Like in Section 2.1.1, we assume w.l.o.g. that $\mathcal{A}_1$ and $\mathcal{A}_2$ are normalised. Two remarks are in order. Firstly, epsilon transitions make it easy to normalise an arbitrary automaton: this operation amounts to replacing the initial states by a fresh state with outgoing epsilon transitions to them, and the final states by a fresh state with incoming epsilon transitions from them. Secondly, it is possible for a normalised epsilon automaton to accept the empty word: all that is needed is an epsilon transition linking its initial to its final states. This greatly simplifies the developments compared to [7], where languages that contain the empty word have to be handled as particular cases.



$$f_2 \to \{i_1, f_1, f_2, \ldots\} \to i_2$$

Figure 10: Automaton accepting $L_1 \diamond L_2$.

The construction of an automaton $\mathcal{A}$ accepting $L = L_1 \diamond L_2$ is shown in Figure 10. It consists in

- merging the initial state i_2 of $\mathcal{A}_2$ with the final state f_1 of $\mathcal{A}_1$ into a single state, which becomes the only final state of $\mathcal{A}$,
- making f_2 the only initial state of $\mathcal{A}$,
- adding an epsilon transition from f_2 to i_1,
- adding right-limit transitions from f_2 and left-limit transitions to i_2, for all sets of states containing i_1, i_2 and f_2.

Compared to the construction in Figure 5, the difference is that the presence of the epsilon transition (f_2, ε, i_1) now forces the state i_1, and therefore the branch associated to $\mathcal{A}_1$, to be visited in order to enable one of the added limit transitions. Let us show that this new construction is correct.

Theorem 2. *The automaton in Figure 10 accepts $L = L_1 \diamond L_2$.*

Proof. We first show that every word in $L_1 \diamond L_2$ is accepted by the constructed automaton $\mathcal{A}$. For any such word w, there exist a non-empty linear ordering J and words $w_j \in L_1$ for each $j \in J$ and $w_j \in L_2$ for each $j \in \widehat{J}^*$ such that $w = \prod_{j \in J \cup \widehat{J}^*} w_j$. For each $j \in J$ (resp. $j \in \widehat{J}^*$), let ρ_j be a run of $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) that reads w_j. In order to obtain a run ρ of $\mathcal{A}$ reading w, one has to map every cut c of $|w|$ onto a triple $(q_1, U, q_2) \in Q \times 2^Q \times Q$, where Q is the set of states of $\mathcal{A}$. There are several cases to consider:

- If c is an internal cut of $|w_j|$ for some $j \in J \cup \widehat{J}^*$, i.e., $c \in \widehat{|w_j|}^*$, then we set $\rho(c) = \rho_j(c)$.
- Otherwise,
  - If $c = \widehat{|w|}_{min}$, $c = \widehat{|w_j|}_{min}$ for some $j \in J$, or $c = \widehat{|w_k|}_{max}$ for some $k \in \widehat{J}$, then we set $\rho(c) = \{f_2, \{f_2, i_1\}, i_1\}$.
  - If $c = \widehat{|w|}_{max}$, $c = \widehat{|w_j|}_{max}$ for some $j \in J$ or $c = \widehat{|w_k|}_{min}$ for some $k \in \widehat{J}$, then we set $\rho(c) = \{i_2, \{i_2\}, i_2\}$.

The ordering $J \cup \widehat{J}^*$ is complete, hence this list of cases is exhaustive. Let us show that the mapping ρ is a run reading w. We call a *segment* $[c_1, c_2]$ of ρ, with $c_1, c_2 \in \widehat{|w|}$, its restriction to the cuts $c \in \widehat{|w|}$ such that $c_1 \le c \le c_2$. Such a segment originates in the state *first*$(\rho(c_1))$ and ends in *last*$(\rho(c_2))$. It reads a subword of the word w read by ρ.

The run ρ starts at the initial state f_2 and ends at the final state i_2. It is composed of segments π_j from i_1 to $f_1 = i_2$ reading subwords w_j, with $j \in J$, by following runs of $\mathcal{A}_1$, and segments π_k from i_2 to f_2 reading subwords w_k, with $k \in \widehat{J}^*$, by following runs of $\mathcal{A}_2$. These segments are joined together according to the ordering $J \cup \widehat{J}^*$: for each $k \in \widehat{J}^*$, there are two possibilities for continuing ρ after the segment π_k:

- If k has a successor j in $J \cup \widehat{J}^*$, then the last state f_2 reached by π_k is linked to the first state i_1 of π_j by the epsilon transition (f_2, ε, i_1).
- Otherwise, one of the right-limit transitions $f_2 \to \{i_1, f_1, f_2, \ldots\}$ added by the construction can be followed from f_2, in order to reach further segments.

The same mechanisms allow to start the run ρ from its initial state f_2. Similarly, for each $k \in \widehat{J}^*$, the segment π_k can be reached either if its first state i_2 corresponds to the last state of the segment π_j associated to the predecessor j of k, or after following a left-limit transition to i_2 added by the construction. By the same principles, the run ends in the final state i_2.

The second part of the proof is to show that every word accepted by $\mathcal{A}$ belongs to the language $L_1 \diamond L_2$. Let ρ be a run of $\mathcal{A}$. This run can be decomposed into a set Γ_1 of segments that originate in f_2 and end in i_2 after going through i_1 and then $\mathcal{A}_1$, and a set Γ_2 of segments that originate in i_2 and end in f_2 after going through $\mathcal{A}_2$. Each segment in Γ_1 (resp. Γ_2) reads a word in L_1 (resp. L_2). The set $\Gamma_1 \cup \Gamma_2$ can be turned into a linear ordering by setting $\pi < \pi'$ iff the segment π comes before the segment π' in the run ρ. We now enumerate three properties that come directly from the structure of $\mathcal{A}$. Firstly, in the ordering $\Gamma_1 \cup \Gamma_2$, each element of Γ_1 that is not the first or the last one has a successor and a predecessor that both belong to Γ_2. Secondly, the ordering $\Gamma_1 \cup \Gamma_2$ is complete. Finally, for all $\pi_2 < \pi_2' \in \Gamma_2$, there exists $\pi_1 \in \Gamma_1$ such that $\pi_2 < \pi_1 < \pi_2'$, since π_2 and π_2' cannot be consecutive in $\Gamma_1 \cup \Gamma_2$, and $\Gamma_1 \cup \Gamma_2$ does not admit an infinite dense subordering that only contains elements of Γ_2. This last property is ensured by the presence of the state i_1 and the transition (f_2, ε, i_1).

We now show that those properties imply that the ordering $\Gamma_1 \cup \Gamma_2$ is isomorphic to $\Gamma_1 \cup \widehat{\Gamma_1}^*$, i.e., that there exists a one-to-one correspondence between these orderings that preserves order. This together with Definition 3 establishes that the word read by ρ belongs to $L_1 \diamond L_2$. Consider the mapping $\tau : \Gamma_1 \cup \Gamma_2 \to \Gamma_1 \cup \widehat{\Gamma_1}^*$ defined as $\tau(\pi_1) = \pi_1$ for all $\pi_1 \in \Gamma_1$ and $\tau(\pi_2) = \big(\{\pi_1 \in \Gamma_1 : \pi_1 < \pi_2\}, \{\pi_1 \in \Gamma_1 : \pi_2 < \pi_1\}\big)$ for all $\pi_2 \in \Gamma_2$. The fact that this mapping is an order-preserving bijection is a consequence of the following properties:

- Each element of Γ_1 does not belong to Γ_2, and is mapped by τ onto itself.
- For all segments $\pi_2, \pi_2' \in \Gamma_2$ such that $\pi_2 < \pi_2'$, one has $\tau(\pi_2) < \tau(\pi_2')$. This follows from the definition of τ and the existence of a segment π_1 between any two elements of Γ_2.
- For all $(K, L) \in \widehat{\Gamma_1}^*$, there exists $\pi_2 \in \Gamma_2$ such that $\tau(\pi_2) = (K, L)$, and $\pi_1 < \pi_2 < \pi_1'$ for all $\pi_1 \in K$ and $\pi_1' \in L$. Indeed, if K (resp. L) has a greatest element K_{max} (resp. a least element L_{min}), then π_2 can be chosen equal to the successor of K_{max} (resp. the predecessor of L_{min}) in $\Gamma_1 \cup \Gamma_2$. Otherwise, (K, L) is a gap in Γ_1, and the completeness property of $\Gamma_1 \cup \Gamma_2$ ensures the existence of π_2.

□

Chapter 3

Non-emptiness test: state of the art

This chapter presents the state of the art for the *non-emptiness test* for automata, that is, deciding whether a given automaton accepts a non-empty language. We start by recalling procedures suited for finite and infinite words, then discuss the case of arbitrary linear orderings, and eventually summarise results related to words of length $\mathbb{R}$ or $\mathbb{Q}$. From now on, the discussion focuses on automata on linear orderings without any epsilon transition.

3.1 Finite words

We first start by the simplest setting of automata on linear orderings: those that do not contain limit transitions. Such automata are equivalent to non-deterministic finite-state automata on finite words. At least one word is accepted by such an automaton if it admits a path from an initial state to a final one. A decision procedure therefore consists in computing the set of states that are *reachable* from the set of initial states [13]. If this set of reachable states does not contain any final state, the language accepted by the automaton is empty, otherwise it contains at least one word.

3.2 Infinite words

Now, let us discuss how to decide whether a Muller automaton accepts a non-empty language. Such an automaton reads infinite words, in the classical sense, namely words of length $\mathbb{N}$. An automaton on linear orderings reduces to a Muller automaton when

- there is no right-limit transition,
- each final state is reachable by a left-limit transition and does not have any incoming or outgoing successor transition,
- left-limit transitions have for destinations final states only.

Figure 2a shows an example of such an automaton.

The non-emptiness test for Muller automata therefore consists in checking whether there is a left-limit set which is reachable from an initial state and whose states are strongly connected [16]. In the positive case, a finite word can be read from the initial state to the limit set, in which states are visited infinitely often, before taking the associated left-limit transition leading to a final state.

3.3 Words of countable scattered length

A procedure deciding whether an automaton on countable scattered linear orderings accepts a non-empty language is developed in [11]. It proceeds by annotating the input automaton as a finite sequence of automata by adding a new successor transitions in place of each limit transition that can be followed by some run. Once every limit transition has been processed, state reachability reduces to the finite-word case.

The procedure works as follows for an input automaton on linear orderings $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$:

1. A new automaton $\mathcal{A}_0 = \left(Q, 2^Q, \Delta_0, I, F\right)$ is created, where

$$\Delta_0 = \left\{(q, \{q, p\}, p) \mid (q, a, p) \in \Delta \text{ and } q, p \in Q \text{ and } a \in \Sigma\right\}$$

   It shares the same states as $\mathcal{A}$ and its transitions are successor ones, labelled by the set of their origin and destination states.

2. Limit sets are processed by increasing cardinality. Every limit transition that can be followed by a run reading an ω-iteration (resp. reverse ω-iteration) is replaced by successor transitions labelled by the set of visited states. More precisely, Automata $\mathcal{A}_1$ up to $\mathcal{A}_M$, where M is the greatest limit set cardinality, are recursively computed as follows:

$$\mathcal{A}_i = \left(Q, 2^Q, \Delta_{i-1} \cup \Delta'_i, I, F\right),$$

   where a transition $(q, P \cup \{q\}, p)$ belongs to Δ'_i iff

   - there exists a right-limit transition $q \to P$, with $q \in Q$, $P \subseteq Q$, $p \in P$ and $|P| = i$, and
   - there exist some successor transitions $(q_0, P_0, p_0), (q_1, P_1, p_1), \ldots, (q_n, P_n, p_n)$ in $\mathcal{A}_{i-1}$. Such that:
     - $p_i = q_j$, where $j \equiv_{\text{ mod } n} i + 1$,
     - $\bigcup_{i \in [0,n]} P_i = P$.

   Left-limit transitions are treated symmetrically.

Example 17. *This procedure is illustrated by studying again the automaton in Figure 2c. First, labels of its successor transitions are replaced by their origin and destination states (Figure 11a). Then, since there is a path from q_1 to itself labelled by $\{q_1\}$, the left-limit transition $\{q_1\} \to q_2$ is turned into a new successor transition from q_1 to q_2 (Figure 11b). Thanks to this transition, q_1 and q_2 are now strongly connected by successor transitions labelled by $\{q_1, q_2\}$. Then, the right-limit transition $q_0 \to \{q_1, q_2\}$ is turned into two successor transitions connecting state q_0 to those belonging to the limit set (Figure 11c). State q_2 being final and reachable from an initial state, one concludes that this automaton accepts a non-empty language.*

The motivation for processing limit sets by increasing order of cardinality comes from the Muller-like semantics of limit transitions. Since a limit transition $q \to P$ or $P \to q$ can be followed if its associated limit set P can be exactly visited infinitely often, only successor transitions and limit sets strictly included in P are of interest.

Note that there is no need to handle ordinal repetitions or words built by the diamond operator. Indeed, if X_1 and X_2 are rational expressions, then the languages respectively denoted by $X_1^\sharp$, $X_1^{-\sharp}$ and $X_1 \diamond X_2$ necessarily contain $\mathscr{L}(X_1)$.

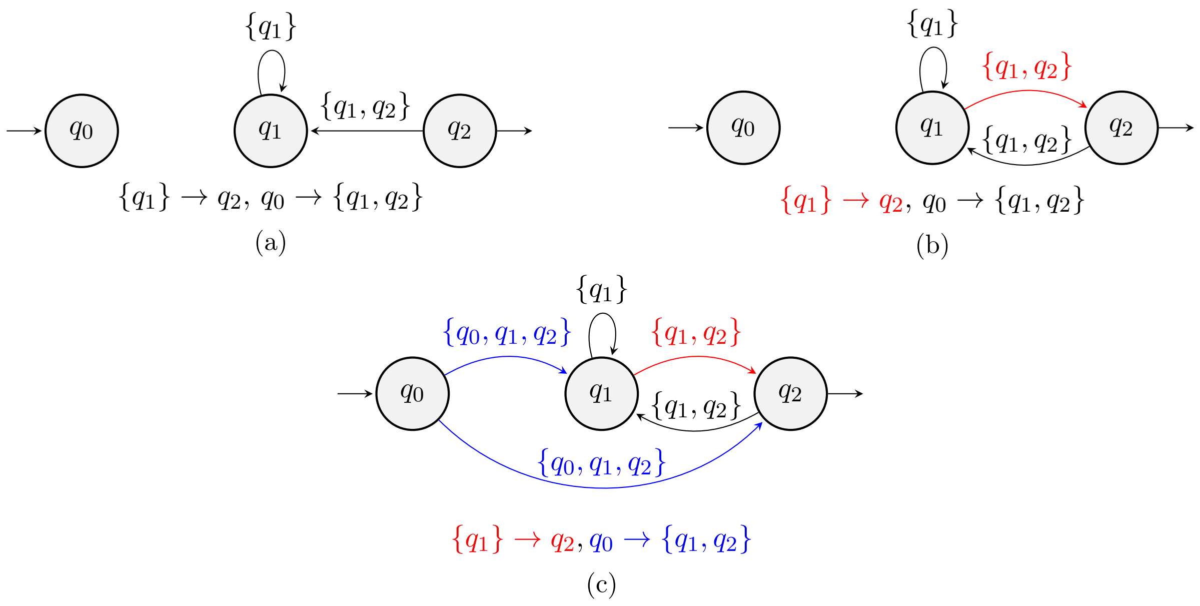


Figure 11: Illustration of the scattered non-emptiness test.

3.4 Words indexed by any linear ordering

When restrictions on orderings are lifted, one should also check whether each limit set can be visited by means of a shuffle. As a straightforward generalisation of results obtained by [2, 4], a limit set P can be visited by a shuffle if there exist states $\{q_0, q_1, \ldots, q_n\} \subseteq P$ and $\{p_0, p_1, \ldots, p_m\} \subseteq P$ fulfilling all the following criteria:

1. $q_0, q_1, \ldots, q_n \to P$ are right-limit transitions in $\mathcal{A}$.
2. $P \to p_0, p_1, \ldots, p_m$ are left-limit transitions in $\mathcal{A}$.
3. There exist some paths in $\mathcal{A}_{i-1}$, such that
   - each of their origins belongs to the set $\{p_0, p_1, \ldots, p_m\}$,
   - each of their destinations belongs to the set $\{q_0, q_1, \ldots, q_n\}$,
   - the label union of the transitions which they follow is equal to P.

In the positive case, for all limit transition pairs of the form $q \to P \to p$ one creates the successor transition $(q, P \cup \{q, p\}, p)$.

3.5 Words of length $\mathbb{R}$ or $\mathbb{Q}$

In this section, we summarise results explained in [2, 4] when it comes to deciding whether a given automaton on linear orderings accepts at least one word of length $\mathbb{R}$, or $\mathbb{Q}$. First, we highlight properties of $\mathbb{R}$ and $\mathbb{Q}$ and then explain how ideas from Sections 3.3 and 3.4 can be fit to those properties. Sections 3.5.2 and 3.5.3 explain the procedure in the case of words indexed by $\mathbb{R}$. Section 3.5.4 outlines differences to be made in the case of words of length $\mathbb{Q}$.

The technique again consists in computing a sequence of finite-word automata. Unlike in the case of words of scattered length [11], for words indexed by $\mathbb{R}$ or $\mathbb{Q}$ [2, 4], the automaton $\mathcal{A}_0$ is not defined and references are made to $\mathcal{A}$. The reason for this change is that it is necessary to make a difference between original successor transitions and annotations. Automata $\mathcal{A}_1$ up to $\mathcal{A}_M$, where M is the greatest limit set cardinality, have the following form:

$$\mathcal{A}_i = \left(Q, 2^Q, \Delta_{i-1} \cup \Delta'_i, I, F\right)$$

where $\Delta_0 = \emptyset$ and Δ'_i is computed according to the rules presented in Sections 3.5.2 and 3.5.3. An automaton accepts at least one word of length $\mathbb{R}$ (resp. $\mathbb{Q}$) iff there exists an open $\mathbb{R}$- (resp. $\mathbb{Q}$-) path connecting an initial state to a final one. The concept of open path is formally introduced in Section 3.5.2, and intuitively corresponds to a sequence of successor and limit transitions that can be followed within a run reading a word of length $\mathbb{R}$ (resp. $\mathbb{Q}$).

3.5.1 $\mathbb{R}$ and $\mathbb{Q}$ as linear orderings

The class of all linear orderings isomorphic to $\mathbb{R}$ forms the order type λ. A characterisation of this equivalence class is the following:

Theorem 3 (Rosenstein [18]). *If a linear ordering*

1. *does not have a least or a greatest element,*
2. *is complete, and*
3. *is* separable*,*

then it has the order type λ.

An ordering is said to be *separable* if it contains a dense countable subordering.

Example 18. *Let R_1 and R_2 be two disjoint real open intervals and S be the singleton $\{s\}$. The linear ordering $R_1 + S + R_2$ has order the type λ. Intuitively, connecting two open real intervals which share a lower and an upper bound, by this bound results in another open interval of $\mathbb{R}$.*

Example 19. *Let $]0,1]$ be the interval $\{r \in \mathbb{R} \mid 0 < r \leq 1\}$. The set $]0,1] \times \mathbb{N}$ equipped with the order relation $(r,n) < (r',n')$ iff $n < n'$ or $(n = n' \wedge r < r')$ forms a linear ordering. Intuitively, it is an infinite concatenation of real intervals, which are left-open and right-closed.*
This ordering has order type λ. Indeed:

- *It does not have a least element since the first real interval does not have a least element.*
- *It does not have a greatest element since it does not have a greatest real interval.*
- *It is complete since $\mathbb{N}$ and each real interval are themselves complete. Moreover, the connection between two successive real intervals does not form a gap, as the lower bound of each interval that is not the first one matches the greatest element of its predecessor.*
- *It is separable since there are countably many separable intervals.*

Example 20. *Let $[0,1]$ be the interval $\{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$. Consider now the set $[0,1] \times \mathbb{R}$ equipped with the order relation $(r,n) < (r',n')$ iff $n < n'$ or $(n = n' \wedge r < r')$. Albeit being complete, not having a first or a last element, this linear ordering does not have the order type λ. This is due to the absence of a dense subordering of order type η. A proof of this property can be found in [2, 4].*

The class of all linear orderings isomorphic to $\mathbb{Q}$ is the order type η. Such orderings have countably infinitely many elements, do not have a least or a greatest element and are dense.

While Properties 1 and 2 can be easily enforced by an automaton, Property 3 cannot. Specific rules therefore are required, they are presented in Sections 3.5.2 and 3.5.3.

3.5.2 Shuffle rule

We first define some notions required in the developments of this section.

Definition 6. *An* open $\mathbb{R}$-path *in $\mathcal{A}_i$ is a non-empty set of successor transitions that either consist in*

- *a single transition of $\mathcal{A}_i$, or*
- *two open $\mathbb{R}$-paths π_1, π_2 in $\mathcal{A}_i$ and a successor transition σ of $\mathcal{A}$ such that the destination of π_1 is the origin of σ, which itself has for destination the origin of π_2.*

Definition 7. *A* closed $\mathbb{R}$-path *in $\mathcal{A}_i$ is a set of successor transitions composed of a successor transition of $\mathcal{A}$ leading to the origin of an open $\mathbb{R}$-path in $\mathcal{A}_i$ whose destination is the origin of a successor transition of $\mathcal{A}$.*

Intuitively, an open (resp. closed) $\mathbb{R}$-path corresponds to a run reading a word indexed by an open (resp. closed) interval of $\mathbb{R}$ with infinitely many elements.

Now, we explain the procedure which annotates a limit set P when it can be visited by a run reading a word of length $\mathbb{R}$, associated to a shuffle. It consists in converting each limit transition pair of the form $q \to P \to p$, into a successor transition of Δ'_i when

- $|P| = i$, and
- there exists a non-empty set of successor transitions, $(q_0, a_0, p_0), (q_1, a_1, p_1), \dots (q_n, a_n, p_n)$, in the automaton $\mathcal{A}$, and a possibly empty set of closed $\mathbb{R}$-paths in $\mathcal{A}_{i-1}$ starting in a state set S and ending in a state set E, whose label union is P_1, such that:
  - $\bigcup_{i\in[0,n]} \{q_i, p_i\} \cup P_1 \cup S \cup E = P$,
  - $p \to P$ is a right-limit transition for all $p \in \bigcup_{i\in[0,n]}\{p_i\} \cup E$, and
  - $P \to q$ is a left-limit transition for all $q \in \bigcup_{i\in[0,n]}\{q_i\} \cup S$.

This rule is a restriction of the general case, now taking into account the fact that $\mathbb{R}$ is dense, by imposing shuffles to contain words whose length is dense and which have a first and a last element. It also makes sure that word lengths are separable by imposing the presence of a word of length $\mathbf{1}$ in each shuffle.

Let us explain the intuition of how a shuffle of $n \neq 0$ words of order type $\mathbf{1} + \lambda + \mathbf{1}$ and $m \neq 0$ words of order type $\mathbf{1}$ contains words whose length has the order type λ. We partition the real interval $]0, 1[$ (itself being isomorphic to $\mathbb{R}$) as in the construction of a Cantor ternary set [9, 10]. Namely, we start with a set of points P composed of the whole interval and an empty set of closed intervals I. Then, we recursively remove from P a third of any interval in its middle and add this to I. At the limit, there are countably infinitely many intervals and uncountably many single points. Figure 12 illustrates the construction, blue segments are elements of I and P contains the remaining points of the interval $]0, 1[$.

Since there are countably many steps and finitely many blue segments at each step, there are countably many intervals in I. On the other hand, there are as many points in P as binary ω-words (each point either lies on the left of a blue segment or on its right at each step). Therefore, set P contains uncountably many points. If the set $P \cup I$ is equipped with the following order relation:

- $I_1 <_{P\cup I} I_2$ for $I_1, I_2 \in I$ iff $i_1 <_{\mathbb{R}} i_2$, $\forall i_1 \in I_1, i_2 \in I_2$,
- $p_1 <_{P\cup I} p_2$ for $p_1, p_2 \in P$ iff $p_1 <_{\mathbb{R}} p_2$,
- $I_1 <_{P\cup I} p_1$ for $I_1 \in I$ and $p_1 \in P$ iff $i_1 < p_1$, $\forall i_1 \in I_1$,

P and I are dense in $P \cup I$.

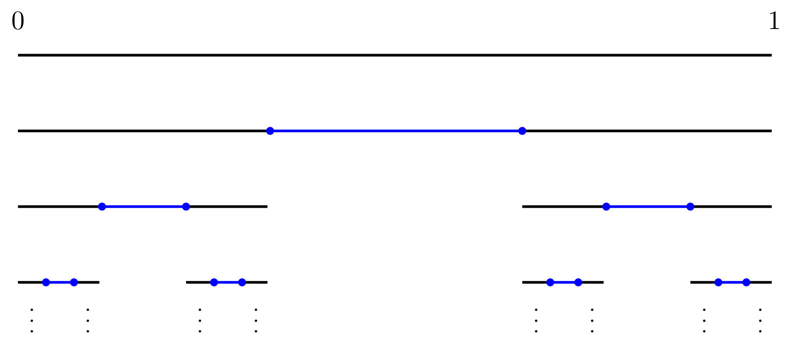
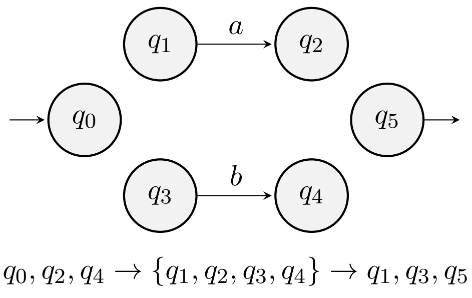


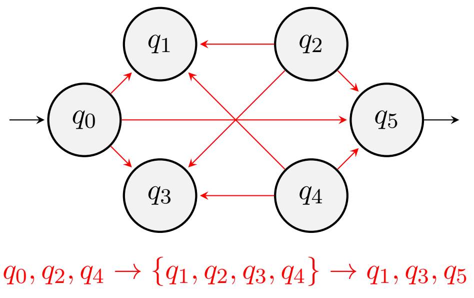
Figure 12: Cantor ternary set construction.

After partitioning P into m suborderings that are each dense in P, each of those suborderings is assigned a different word of length $\mathbf{1}$. Symmetrically, I is partitioned into n suborderings that are dense in I, and each of those suborderings is assigned a word of order type $\mathbf{1} + \lambda + \mathbf{1}$. It is easy to check that this word of length $\mathbb{R}$ satisfies the definition of a shuffle.

Example 21. *We illustrate this rule by studying again the automaton in Figure 3a. First, a copy of the input automaton is kept, in order to remember which transitions are original successor transitions (Figure 13a). Then, since the automaton does not contain limit sets of cardinality less than 4, the associated automata $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are all equal and have an empty transition relation. The automaton $\mathcal{A}_4$ is now constructed. There are paths, respectively composed of a single successor transition form q_1 and q_3 to q_2 and q_4 in $\mathcal{A}$, and the limit transitions $q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3$. Therefore, all transitions connecting pairs of states (q, p) such that $q \to \{q_1, q_2, q_3, q_4\} \to p$ is a limit transition are added with the label $\{q_1, q_2, q_3, q_4\} \cup \{q, p\}$. For the sake of readability, these labels are omitted in Figure 13b. Since there is an open $\mathbb{R}$-path form the initial state q_1 to the final state q_5, one concludes that the automaton $\mathcal{A}$ accepts at least one word of length $\mathbb{R}$.*


q_1
a
q_2
q_0
q_5
q_3
b
q_4
$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3, q_5$

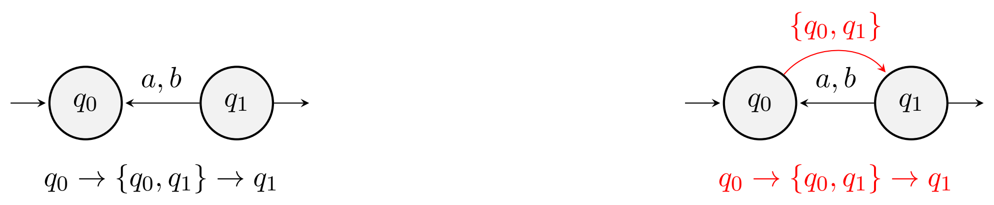
(a) Input automaton, $\mathcal{A}$.


q_1
q_2
q_0
q_5
q_3
q_4
$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3, q_5$

(b) Output automaton, $\mathcal{A}_4$.

Figure 13: Illustration of the shuffle rule on $sh\,(a, b)$.

Example 22. *Similarly, we can apply this rule to the automaton in Figure 14a. While in Example 14 showing that this automaton accepts a word of length $\mathbb{R}$ was somewhat involved, it is immediate by applying the rule. The only pair of limit transitions is associated with a limit set composed of the origin of the right-limit transition, q_0, and the destination of the left-limit transition, q_1. In addition to that, there exists a successor transition from q_1 to q_0. Therefore, the limit transition pair $q_0 \to \{q_0, q_1\} \to q_1$ is converted into the successor transition $(q_0, \{q_0, q_1\}, q_1)$. This yields the automaton in Figure 14b.*


q_0
a, b
q_1
$q_0 \to \{q_0, q_1\} \to q_1$
$\{q_0, q_1\}$
q_0
a, b
q_1
$q_0 \to \{q_0, q_1\} \to q_1$

(a) (b)

Figure 14: Illustration of the shuffle rule on $sh\,(a + b)$.

3.5.3 Infinite repetition rule

As it is illustrated by Example 19, the (reverse) ω-iteration of a real interval that is open on one side and closed on the other is isomorphic to $\mathbb{R}$. The rule explained in this section annotates limit transitions that can be followed by means of such iterations. The successor transition $(q, P \cup \{q\}, p_1)$ is added to Δ'_i when

- $|P| = i$,
- $q \to P$ is a right-limit transition, and
- $p_1 \in P$ and there exist $p_2 \in P$ such that,
  - there exists a successor transition in $\mathcal{A}$ from p_1 to p_2, and
  - there exists an open $\mathbb{R}$-path in $\mathcal{A}_{i-1}$, from p_2 to p_1 whose label union is equal to P.

Left-limit transitions are handled symmetrically.

3.5.4 Adaptations to words of length $\mathbb{Q}$

Since $\mathbb{Q}$ has uncountably many gaps that are densely nested between its elements, a new definition of the concept of path is required.

Definition 8. *An* open $\mathbb{Q}$-path *in $\mathcal{A}_i$ is a non-empty set of successor transitions composed of either*

- *a single transition belonging to $\mathcal{A}_i$, or*
- *two open $\mathbb{Q}$-paths π_1 and π_2 in $\mathcal{A}_i$ such that the destination of π_1 is the origin of π_2, or*
- *two open $\mathbb{Q}$-paths π_1 and π_2 in $\mathcal{A}_i$ and a successor transition σ of $\mathcal{A}$ such that the destination of π_1 is the origin of σ, which itself has for destination the origin of π_2.*

Intuitively, an open $\mathbb{Q}$-path is the concatenation of two open intervals of $\mathbb{Q}$, joined either by a rational number or by an irrational number which forms a gap. Similarly, we call *interval $\mathbb{Q}$-path*, a path which can be followed by a run reading a word of a length which is an interval of $\mathbb{Q}$, that, is a path of order type $\mathbf{1}$, $\mathbf{1} + \lambda$, $\lambda + \mathbf{1}$ or $\mathbf{1} + \lambda + \mathbf{1}$.

The shuffle rule is then adapted as follows: the pair of limit transitions $q \to P \to p$, where $q, p \in Q$ and $P \subseteq Q$ is translated into a successor transition of Δ'_i when

- $|P| = i$,
- there exists $p_1 \in P$ such that $p_1 \to P \to p_1$ are limit transitions, and

- there exists a non-empty set of interval $\mathbb{Q}$-paths starting in a state set S and ending in a state set E, whose label union is P_1, such that
  - $\{p_1\} \cup P_1 = P$,
  - $q \to P$ is a right-limit transition for all $q \in E$, and
  - $P \to q$ is a left-limit transition for all $q \in S$.

The infinite repetition rule requires the existence of an interval $\mathbb{Q}$-path, which is not composed of a single successor transition of $\mathcal{A}$, from a state to itself and that visits exactly the limit set involved in the limit transition.

Chapter 4

Making the non-emptiness test more efficient

In this chapter, we explain how the procedure deciding whether an automaton on linear orderings accepts at least one word of length $\mathbb{R}$ can be adapted in order to yield an efficient implementation. We first restate the procedure of [2, 4] so as to avoid redundant operations. Then, we introduce a method for reducing the number of added successor transitions, to go from a quadratic to a linear behaviour, while preserving reachability properties.

4.1 Preliminaries

Chapter 3 presented a procedure deciding whether a given automaton on linear orderings $\mathcal{A}$ accepts at least one word of length $\mathbb{R}$. It builds a sequence of finite-word automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M$, where M is the greatest cardinality of limit sets in $\mathcal{A}$. This procedure was illustrated by Example 21. This example already contained an optimisation, with respect to the original paper. Namely, all pairs of limit transitions associated to the only limit set of the automaton were translated into successor transitions after analysing the properties of that limit set. This idea is exploited in this chapter, by focusing on limit sets rather than limit transitions. This change not only prevents from uselessly recomputing things but it also gives a natural opportunity to decouple the two following questions:

1. Can a given limit set be visited, hence enabling its associated limit transitions?
2. Does a given enabled limit transitions need to be turned into a successor transition in the automaton sequence $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M$?

4.2 Improving the complexity

In this section, we study the complexity of the shuffle and infinite repetition rules in terms of added successor transitions as a function of the amount of limit transitions and size of their limit set.

4.2.1 Original construction

Before trying to improve the original construction, we study study it.

Let $q_0, q_1, \ldots, q_n \to P \to p_0, p_1, \ldots, p_m$ be a set of limit transitions. If P satisfies the requirements of the shuffle rule, one adds the transition set $\big\{(q_i, P \cup \{q_i, p_j\}, p_j) \mid i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}\big\}$. The complexity of the shuffle rule therefore is $\mathcal{O}(n \cdot m) = \mathcal{O}\left(|Q|^2\right)$.

Let $q_0, q_1, \dots, q_n \to P$ be a set of right-limit transitions. If P can be visited by an infinite repetition, one adds a subset of the transition set $\left\{(q_i, P \cup \{q_i\}, p) \mid i \in \{0, 1, \dots, n\} \text{ and } p \in P\right\}$. In the worst case, this whole set is added. This worst-case scenario is illustrated by Figure 15, in which blue edges are open $\mathbb{R}$-paths and black ones are successor transitions. Since every state of the limit set has an outgoing successor transition belonging to the cycle, a transition from every state q_i to every state p_j needs to be created. The complexity of the infinite repetition rule therefore is $\mathcal{O}(n \cdot |P|) = \mathcal{O}\left(|Q|^2\right)$. From this analysis, we conclude that in the worst case, each limit set produces $\mathcal{O}\left(|Q|^2\right)$ transitions. Since there can be as many as $2^{|Q|}$ limit sets, reducing the amount of transitions is a useful step towards an efficient implementation of the non-emptiness test.

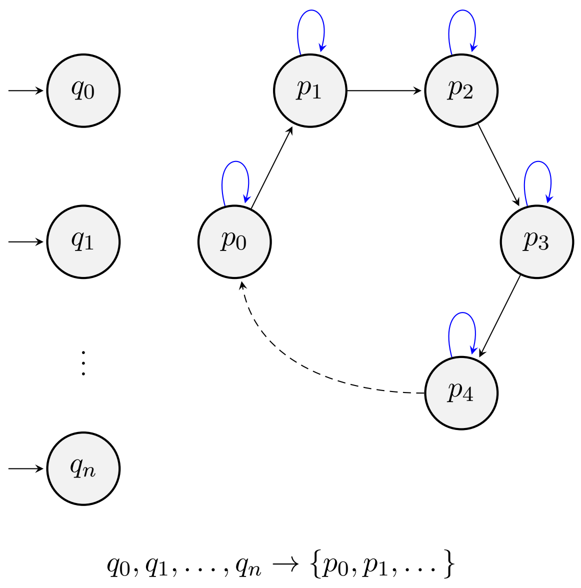


Figure 15: Worst-case scenario for the infinite repetition rule.

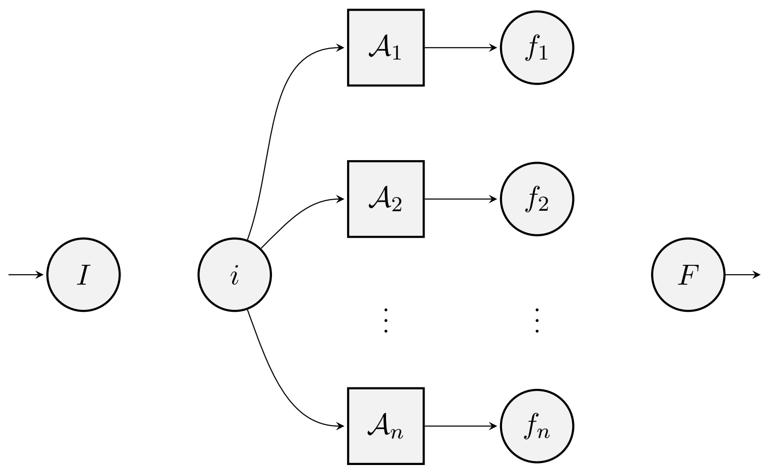
4.2.2 Compact shuffles

A first observation is that some automata better behave than others, while accepting the same language.

Definition 9. Compact shuffle automaton. *Let $L_1, L_2, \dots, L_n$ be rational languages, respectively denoted by the rational expressions $X_1, X_2, \dots, X_n$ and $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ normalised automata respectively accepting them. A compact shuffle automaton accepting the language $\mathscr{L}(sh(X_1, X_2, \dots, X_n))$ is built as follows. One adds two new states I and F, respectively being the only initial and final states of the new automaton. Then, the initial state of each automaton $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ is merged into a single state i. Finally, one adds all limit transitions of the form $I, f_1, f_2, \dots, f_n \to \{f_1, f_2, \dots, f_n, i\} \cup P \to i, F$, where P is a superset of the states of the automata $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$.*

This construction is illustrated in Figure 16. It is very close to what was proposed by [1] as an automaton accepting a shuffle (see Figure 3a for example). The main difference consists in merging the initial state of each automaton $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ into a single state i. (This can be seen as a form of determinisation. Instead of guessing which word is to be read after a left-limit transition, every left-limit transition of the shuffle leads to the state i, from which a "deterministic" choice is made.)

When the shuffle rule studies a compact shuffle automaton, it adds a linear number of transitions in terms of n (provided that there is a constant number of limit transition involving states which do not belong to P). It is worth mentioning that in the scope of implementing a decision procedure for some



$$I, f_1, f_2, \ldots, f_n \to \{f_1, f_2, \ldots, f_n, i\} \cup P \to i, F$$

$$\vdots$$

Figure 16: Compact shuffle automaton.

logics, automata are not the input but an internal tool, for representing sets of models of formulas. In this case, it is natural to use compact shuffles instead of standard ones.

4.2.3 New construction

In this section, we propose to restrict the amount of transitions added when a limit set can be visited by a run reading a word of length $\mathbb{R}$. The idea is to turn such limit sets into strongly connected components, in the sense of $\mathbb{R}$-paths. Indeed, if a set of states is strongly connected, then adding extra transitions within this set cannot alter its reachability properties. We formally define this notion of strongly connected component in Definition 10.

Definition 10. *A* strongly connected component in the sense of $\mathbb{R}$-paths *is a set of states P such that*

- *each state is reachable from itself by following*
  - *a successor transition of $\mathcal{A}$, and then an open $\mathbb{R}$-path visiting exactly P, or*
  - *an open $\mathbb{R}$-path visiting exactly P, and then a successor transition of $\mathcal{A}$, and*
- *for every state pair $p_1, p_2 \in P$, p_1 is reachable from p_2 by following an open $\mathbb{R}$-path, a closed $\mathbb{R}$-path, a successor transition of $\mathcal{A}$ followed by an open $\mathbb{R}$-path, or an open $\mathbb{R}$-path followed by a successor transition of $\mathcal{A}$, such that the involved $\mathbb{R}$-path visits a subset of P.*

Intuitively, in a strongly connected component in the sense of $\mathbb{R}$-paths states can be visited by a run reading a word indexed by an infinite interval of $\mathbb{R}$, and and every state is reachable from itself by following a run reading a word indexed by an interval of $\mathbb{R}$ that is closed on one side and open on the other.

New shuffle rule

For every limit set P satisfying the following conditions,

- $|P| = i$,

- there exists a non-empty set of successor transitions, $(q_0, a_0, p_0), (q_1, a_1, p_1), \ldots, (q_n, a_n, p_n)$, in the automaton $\mathcal{A}$, and a possibly empty set of closed $\mathbb{R}$-paths in $\mathcal{A}_{i-1}$ starting in a state set S and ending in a state set E, whose label union is P_1, such that:
  - $\bigcup\limits_{i\in[0,n]} \{q_i, p_i\} \cup P_1 \cup S \cup E = P$,
  - $p \to P$ is a right-limit transition for all $p \in \bigcup_{i\in[0,n]}\{p_i\} \cup E$, and
  - $P \to q$ is a left-limit transition for all $q \in \bigcup_{i\in[0,n]}\{q_i\} \cup S$,

one adds successor transitions to Δ'_i. Every right-limit transition $p \to P$ in $\mathcal{A}$ is converted into the successor transition

$$(p, P \cup \{p\}, q_0).$$

Symmetrically, every left-limit transition $P \to q$ in $\mathcal{A}$ is converted into

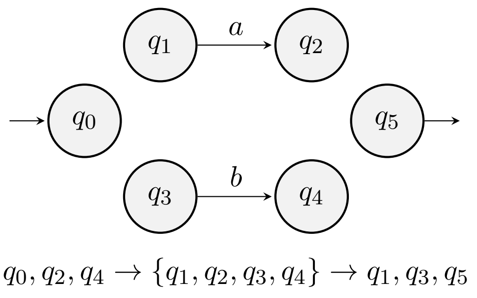
$$(p_0, P \cup \{q\}, q).$$

Lemma 1. *To every transition added by the original shuffle rule corresponds an open $\mathbb{R}$-path in the automaton built by the new shuffle rule.*

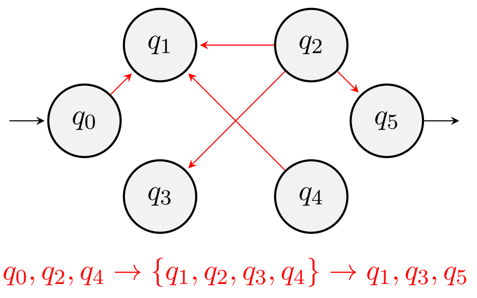
Proof. Since every transition of the new construction is added by the original one, we have to show that reachability is preserved by the new rule. To do so, we construct an open $\mathbb{R}$-path in place of each original transition. State q_0 is reachable from every state p such that $p \to P$ is a right-limit transition, by following transitions added by the new rule, which form open $\mathbb{R}$-paths. Symmetrically, every state q such that $P \to q$ is a left-limit transition is reachable from state p_0 by following a freshly added transition, forming an open $\mathbb{R}$-path as well. In addition to that, there is a successor transition from q_0 to p_0 in $\mathcal{A}$. Therefore for every pair of limit transitions $p \to P \to q$, there exists an open $\mathbb{R}$-path from p to q in $\mathcal{A}_i$. □

Corollary 1.1. *A limit set that can be visited by a shuffle is turned into a strongly connected component in the sense of $\mathbb{R}$-paths by the shuffle rule.*

Example 23. *We illustrate this new construction by studying again the automaton in Figure 3a. After keeping a copy of $\mathcal{A}$ (Figure 17a), we arbitrarily chose (q_1, a, q_2) as the successor transition to which every new transition will be added. Then, q_0, q_2 and q_3 are connected to q_1, while q_2 is connected to q_1, q_3 and q_5, this is displayed in Figure 17b. There is still an open $\mathbb{R}$-path from q_0 to q_5, it now sequentially visits q_0, q_1, q_2 and q_5, by combining annotations and an original transition. One concludes that this automaton accepts at least one word of length $\mathbb{R}$.*


q_1
a
q_2
q_0
q_5
q_3
b
q_4
$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3, q_5$

(a) Input automaton, $\mathcal{A}$.


q_1
q_2
q_0
q_5
q_3
q_4
$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3, q_5$

(b) Output automaton, $\mathcal{A}_4$.

Figure 17: Illustration of the simplified shuffle rule.

New infinite repetition rule

For every limit set P of cardinality i, one arbitrarily choses one state $p_1 \in P$.

- If there exists $p_2 \in P$ such that $\mathcal{A}$ admits a successor transition from p_1 to p_2, and $\mathcal{A}_{i-1}$ admits an open $\mathbb{R}$-path from p_2 to p_1 labelled by P,
  - for each right-limit transition $q \to P$, one adds the transition
  $$(q, P \cup \{q\}, p_1)\,, \text{ and}$$
  - for each left-limit transition $P \to q$, one adds the transition
  $$(p_2, P \cup \{q\}, q)\,.$$

  If there are several states such as p_2, only one (chosen arbitrarily) needs to be considered.
- If there exists $p_0 \in P$ such that $\mathcal{A}$ admits a successor transition from p_0 to p_1, and $\mathcal{A}_{i-1}$ admits an open $\mathbb{R}$-path from p_1 to p_0 labelled by P,
  - for each right-limit transition $q \to P$, one adds the transition
  $$(q, P \cup \{q\}, p_0)\,, \text{ and}$$
  - for each left-limit transition $P \to q$, one adds the transition
  $$(p_1, P \cup \{q\}, q)\,.$$

  If there are several states such as p_0, only one (chosen arbitrarily) needs to be considered.

Lemma 2. *If a limit set P can be visited by an infinite repetition, there exists another limit set P' such that $0 < |P'| < |P|$ and P' can be visited by a shuffle.*

Proof. Let us assume that there is no such smaller limit set P'. The fact that P can be visited by an infinite repetition implies the existence of an open $\mathbb{R}$-path in $\mathcal{A}_{|P|-1}$. By hypothesis, this open $\mathbb{R}$-path must itself result from a smaller limit set which can be visited by an infinite repetition. A contradiction appears when the only possible candidate is a limit set of size one. Indeed, the automaton $\mathcal{A}_0$ has an empty transition relation, thus it cannot provide an open $\mathbb{R}$-path to $\mathcal{A}_1$ for an infinite repetition. □

Lemma 3. *To every transition added by the original infinite repetition rule corresponds an open $\mathbb{R}$-path in the automaton built by the new infinite repetition rule.*

Proof. The proof is given for right-limit transitions, since left-limit transitions are symmetric. As for the shuffle rule, every transition added by new rule was also added by the original one. Moreover, it is required to show that the criterion determining if an infinite repetition is enabled is equivalent to the original one. We do this by showing the two following statements:

1. If there exists a state pair $p_k, p_\ell \in P$ such that there are a successor transition in $\mathcal{A}$ from p_k to p_ℓ and an open $\mathbb{R}$-path from p_ℓ to p_k, labelled by P, every state $p_0 \in P$ is reachable from itself by a successor transition of $\mathcal{A}$ followed, or preceded, by an open $\mathbb{R}$-path labelled by P.
2. For every right-limit transition $q \to P$, and state pair $p_k, p_\ell \in P$ such that there is a successor transition in $\mathcal{A}$ from p_k to p_ℓ and an open $\mathbb{R}$-path from p_ℓ to p_k, labelled by P, there exists an open $\mathbb{R}$-path from q to p_k, labelled by $P \cup \{q\}$, resulting from the new infinite repetition rule.

The proof is made by induction of limit set cardinality.

Base case.
Proof. We first prove Statement 1. Let P be a limit set of cardinality i such that,

- there is no smaller limit set which can be visited by an infinite repetition, and
- P can be visited by an infinite repetition.

By Lemma 2, there exists at least one limit set strictly smaller than P that can be visited by a shuffle. From Corollary 1.1, every such limit set forms a strongly connected component in the sense of $\mathbb{R}$-paths. Therefore, every open $\mathbb{R}$-path in $\mathcal{A}_{i-1}$ consists in

- at least one open $\mathbb{R}$-path, labelled by a set forming a strongly connected component in the sense of $\mathbb{R}$-paths, and
- if there are more than two such sets, a successor transitions of $\mathcal{A}$ connecting them.

Consider now a state pair $p_k, p_\ell \in P$ such that there are a successor transition in $\mathcal{A}$ from p_k to p_ℓ and an open $\mathbb{R}$-path $\pi_{\ell-k}$ from p_ℓ to p_k, labelled by P. If $\pi_{\ell-k}$ is composed of a single transition in $\mathcal{A}_{i-1}$, there exists another open $\mathbb{R}$-path $\pi'_{\ell-k}$ labelled by P, with the same origin and destination, and which follows transitions whose origins and destinations together cover exactly P. This is a consequence of Corollary 1.1. Therefore, we can decompose $\pi_{\ell-k}$ as follows:

- a successor transition in $\mathcal{A}$ from p_0 to p_1, $p_0, p_1 \in P$,
- $\pi_{\ell-0}$ is an open $\mathbb{R}$-path starting in p_ℓ and ending in p_0, and
- π_{1-k} is an open $\mathbb{R}$-path starting in p_1 and ending in p_k,

where the label union of $\pi_{\ell-0}$ and π_{1-k} is equal to P. We can now build an open $\mathbb{R}$-path π_{1-0}, by connecting π_{1-k} to $\pi_{\ell-0}$ through the successor transition from p_k to p_ℓ. From Corollary 1.1, this reasoning can be done to any state which belongs to P, and it will play the role of p_0 or p_1.

Now, we prove Statement 2. We now know that if one state can be reached from itself by following an open $\mathbb{R}$-path, preceded or followed by a successor transition of $\mathcal{A}$ while visiting a limit set P, so can be any other state that belongs to P. This implies that if at least one transition is added by the original rule in place of a right-limit transition $q \to P$, at least one is also added by the new rule. Now, we have to show that for such a right-limit transition, reachability is preserved. The transition $(q, P \cup \{q\}, p_1)$ is added by the new rule if p_1 can be reached from itself by following a successor transition of $\mathcal{A}$ to $p_2 \in P$ and then an open $\mathbb{R}$-path π_{2-k} labelled by P. We consider two states p_k and p_ℓ that are visited by π_{2-k}, and such that a successor transition of $\mathcal{A}$ from p_k to p_ℓ is used in the path. Clearly, the original rule would add the transition $(q, P \cup \{q\}, p_k)$. We now build an open $\mathbb{R}$-path from q to p_k, labelled by $P \cup \{q\}$. It is composed of the following transitions:

- the added transition $(q, P \cup \{q\}, p_1)$,
- the successor transition of $\mathcal{A}$ from p_1 to p_2, and
- the open $\mathbb{R}$-path π_{2-k}.

The transition $(q, P \cup \{q\}, p_0)$ is also added by the new infinite repetition rule if there is a successor transition in $\mathcal{A}$ from p_0 to p_1 and an open $\mathbb{R}$-path π_{1-0} from p_1 to p_0, labelled by P. This allows to consider states with a previously incoming successor transition used in a cycle and which now have an outgoing one. This is illustrated by Figure 18. Blue edges are open $\mathbb{R}$-paths present before applying the rule, red ones are those added by the rule and successor transitions are drawn in black. The transition from q to p_1, connects it to the outer cycle and the transition from q to p_0 connects it to the inner cycle.

We now show that at most two transitions are required to entirely recover reachability properties of the original rule. Let us assume that the infinite repetition rule added the transitions $(q, P \cup \{q\}, p_0)$ and $(q, P \cup \{q\}, p_1)$. Assume also that there exist a successor transition in $\mathcal{A}$ from p_k to p_ℓ and an open $\mathbb{R}$-path $\pi_{k-\ell}$ labelled by P, which neither contains a successor transition leaving state p_0 nor state p_1. From Corollary 1.1, there exists an open $\mathbb{R}$-path $\pi'_{k-\ell}$ labelled by P,

- whose origin is p_k,
- whose destination is p_ℓ, and
- which contains a successor transition of $\mathcal{A}$, whose origin is p_0 or p_1.

Therefore, two transitions in place of one are enough to preserve reachability.

Induction step.

Proof sketch. The proof of the induction step is essentially the same, consisting in breaking open $\mathbb{R}$-paths around states belonging to the cycle. The only difference is that it now relies on the fact that the new infinite repetition rule preserves strong connection in the sense of $\mathbb{R}$-paths, which is a consequence of Statement 2.

□

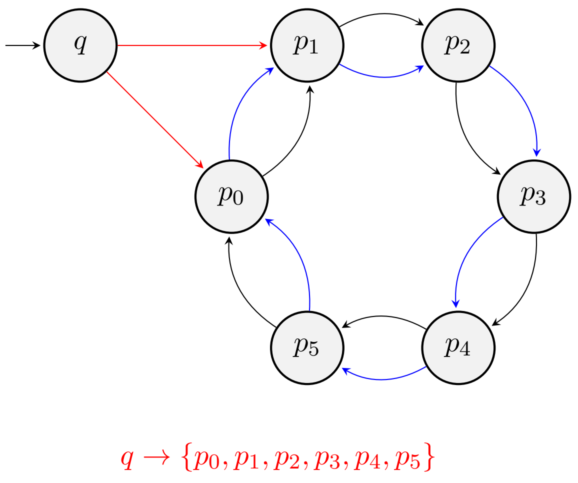


Figure 18: Illustration of the simplified infinite repetition rule.

4.2.4 Complexity of the new construction

We showed in Section 4.2.1 that the original procedure replaces each limit transition by $\mathcal{O}(|Q|)$ successor transitions. Now, we show that the new procedure which we propose adds $\mathcal{O}(1)$ successor transitions.

Theorem 4. *Given an automaton on linear orderings $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, the application of the new shuffle and infinite repetition rules produces an automaton sequence $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M$, such that:*

1. *The relation transition of each resulting automaton has a size $\mathcal{O}(|\Delta|)$*
2. *There exists an open $\mathbb{R}$-path in the automaton $\mathcal{A}_M$, from an initial state to a final one iff $\mathcal{A}$ accepts at least one word of length $\mathbb{R}$.*

Proof. We first prove Point 1 by counting how many transitions added by each rule, from a single limit transition.

- The shuffle rule converts each limit transition into 0 or 1 successor transition.
- The infinite repetition rule converts each limit transition into 0, 1, or 2 successor transition.

Thus, every limit transition of $\mathcal{A}$ is translated into at most 3 successor transitions.

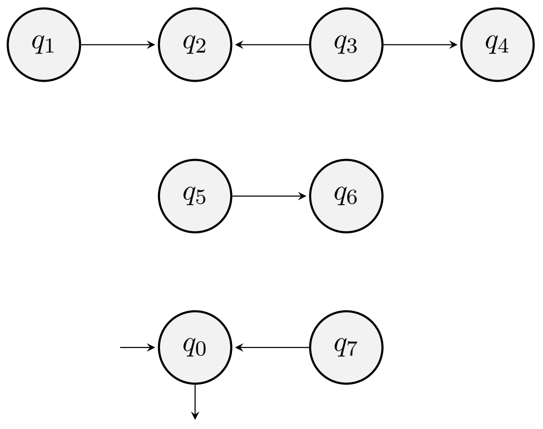
Point 2 is a consequence of Lemmas 1 and 3.

□

4.2.5 Illustration

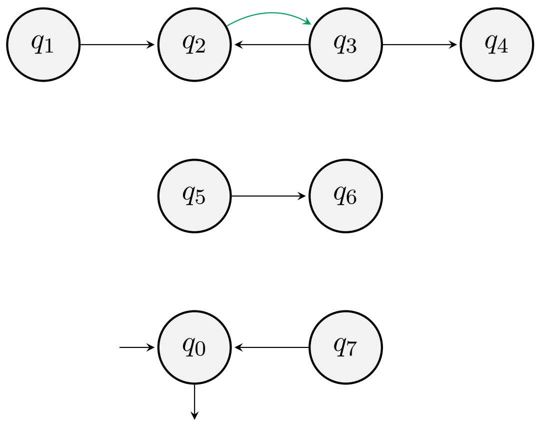
To conclude this chapter, we illustrate the two simplified rules by a means of an example.

Example 24. *We study the automaton in Figure 19a. In order to improve readability, original successor transitions of this automaton are displayed in black, without any label as these are not relevant for reachability. First, one studies the limit set $\{q_2, q_3\}$. It consists in a shuffle, and we add a transition from q_2 to q_3 (as usual, labels are omitted for the sake of readability). This results in the automaton $\mathcal{A}_2$ shown in Figure 19b. We chose to keep original successor transitions (in black) in annotated automata as it allows to see cycles and $\mathbb{R}$-paths more easily. Then, we analyse the limit set $\{q_1, q_2, q_3, q_4\}$. Although q_4 is reachable from q_1 by a closed $\mathbb{R}$-path visiting exactly states q_1, q_2, q_3 and q_4, the absence of a successor transition in $\mathcal{A}$ between q_1 and q_4 compromises the existence of a shuffle. Moreover, there is no cycle in $\mathcal{A}_2$ involving this limit set and therefore the infinite repetition cannot be invoked either. One then considers the limit set $\{q_1, q_2, q_3, q_4, q_5, q_6\}$. In this case, there are limit transitions, a successor transition and a closed $\mathbb{R}$-path which fulfil the requirements of the shuffle rule. The successor transitions of $\mathcal{A}_6$ are built around the successor transition joining q_5 and q_6 in $\mathcal{A}$. This is depicted in Figure 19c. Finally, we study the last limit set, which cannot be visited by a shuffle since there does not exist any right-limit transition involving it. Yet, an infinite repetition can be followed. First, state q_1 is arbitrarily picked, as it belongs to the limit set. There exist a successor transition from q_1 to q_2 and an open $\mathbb{R}$-path from q_2 to q_1 which visits every state of the limit set. Thus, we add a transition from q_2 to q_0. As an open $\mathbb{R}$-path from q_0 to itself now exists, one concludes that this automaton accepts at least one word of length $\mathbb{R}$.*



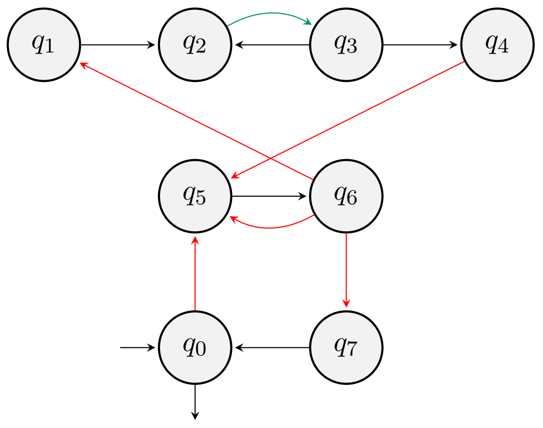
$$q_2 \to \{q_2, q_3\} \to q_3$$
$$q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1$$
$$q_0, q_4, q_6 \to \{q_1, q_2, q_3, q_4, q_5, q_6\} \to q_1, q_5, q_7$$
$$\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \to q_0$$

(a) Input automaton $\mathcal{A}$.



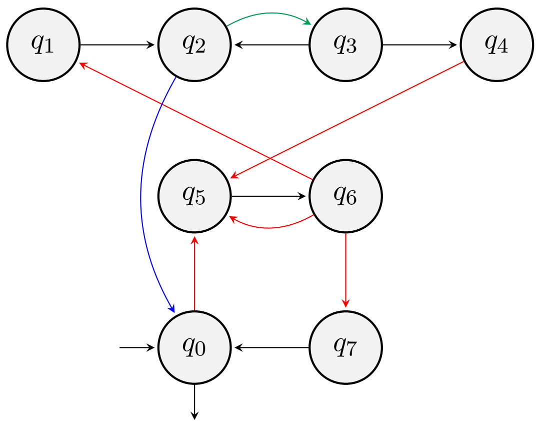
$$q_2 \to \{q_2, q_3\} \to q_3$$
$$q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1$$
$$q_0, q_4, q_6 \to \{q_1, q_2, q_3, q_4, q_5, q_6\} \to q_1, q_5, q_7$$
$$\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \to q_0$$

(b) Automaton $\mathcal{A}_2$.



$$q_2 \to \{q_2, q_3\} \to q_3$$
$$q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1$$
$$q_0, q_4, q_6 \to \{q_1, q_2, q_3, q_4, q_5, q_6\} \to q_1, q_5, q_7$$
$$\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \to q_0$$

(c) Automaton $\mathcal{A}_6$.



$$q_2 \to \{q_2, q_3\} \to q_3$$
$$q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1$$
$$q_0, q_4, q_6 \to \{q_1, q_2, q_3, q_4, q_5, q_6\} \to q_1, q_5, q_7$$
$$\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \to q_0$$

(d) Automaton $\mathcal{A}_8$.

Figure 19: Illustration of the simplified shuffle and infinite repetition rules.

Chapter 5

Implementation issues

In this chapter, we present how developments of Chapter 4 can be implemented. First, we explain how the input automaton and the resulting sequence can be represented as a single automaton. Then, we introduce a more concise labelling technique. After that, we provide algorithms for the shuffle and the infinite repetition rules, in the form of pseudocode. Finally, we present a practical implementation of the whole non-emptiness.

5.1 Representing all automata as a single one

In this section, we take one more step towards a practical implementation of the decision procedure for non-emptiness. Namely, we show how a single automaton $\tilde{\mathcal{A}}$ can replace the input automaton $\mathcal{A}$ and the sequence $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M$, where M is the cardinality of the greatest limit set. This is possible since all of these automata share the same set of states and the only difference between them is their respective transitions. Conceptually, our technique consists in turning the automaton $\mathcal{A}$ into $\tilde{\mathcal{A}}$ by replacing the label of each successor transition by the symbol $(0, \emptyset)$. Then, every transition of label P formerly added to the automaton $\mathcal{A}_i$, $i \in \{1, 2, \ldots, M\}$ is added to $\tilde{\mathcal{A}}$ with the label (i, P). We denote by $\tilde{\mathcal{A}_i}$ the automaton $\tilde{\mathcal{A}}$ in which limit sets of cardinality up to i have been processed.

The advantage of this approach is to significantly reduce the space complexity of the procedure, since $|Q|$ states are required in place of up to $|Q|^2$. Moreover, distinguishing to which original automaton a transition of $\tilde{\mathcal{A}}$ belongs is immediate, as it just amounts to checking the first element of its label. If this element is 0, the transition belongs to $\mathcal{A}$. In the developments of this chapter, we define this transition to be a type s transition (s standing for successor). Otherwise, a label of cardinality i would have belonged to each automaton of index greater than i, in this case we define the type of this transition as ℓ (standing for limit).

5.2 A simpler labelling strategy

Chapter 4 (and implicitly Section 5.1) rely on the same labelling technique as in the original paper [2]. Its main property is that each transition added by the shuffle or the infinite repetition rule is labelled by the content of its underlying limit set, plus its origin and destination. The main reason for adding the origin and the destination to the label is to be able to identify which states are visited by a run associated to each $\mathbb{R}$-path, only by considering the label union of this path. This makes proofs clearer and more concise. Yet, in the scope of a practical implementation, a more efficient technique consists in labelling transitions with limit set content only and to change the criterion for a set of states to be visited. A state q is visited by some $\mathbb{R}$-path if there exists some transition in the path for which

- q is the origin,

- q is the destination, or
- q is a label member.

This allows to reduce the alphabet size of the automaton $\tilde{\mathcal{A}}$. To each limit set now corresponds a single symbol of the new alphabet, while formerly, a single limit set could lead to $|Q|$ symbols. Consider the automaton in Figure 20, the limit transitions associated to the set $\{p\}$ form a shuffle structure. For each state $q_i, i \in \{0, 1, \dots, n\}$, the alphabet symbol $\{q_i, p\}$ is generated. By contrast, the technique that we propose generates one symbol, namely $\{p\}$.

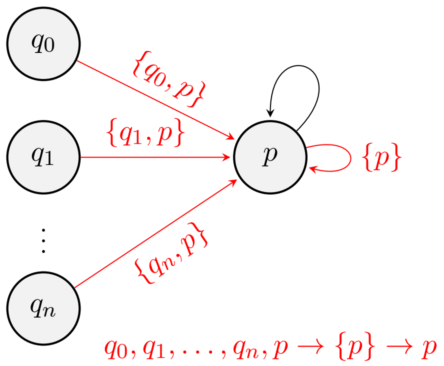


Figure 20: Original labelling technique.

Moreover, conceptual labels (i, P) become redundant since i now is the cardinality of P. Therefore, the alphabet of $\tilde{\mathcal{A}}$ consists in the set of limit sets in $\mathcal{A}$, plus the empty set. Formally, the alphabet $\tilde{\Sigma}$ of $\tilde{\mathcal{A}}$ is defined as

$$\{P \mid (P, q) \in \Delta, q \in Q\} \cup \{P \mid (q, P) \in \Delta, q \in Q\} \cup \{\emptyset\}.$$

5.3 Implementing the shuffle rule

5.3.1 Reminder

In Chapter 3 we saw that a limit set P can be visited by a run reading a word of length $\mathbb{R}$ when there exist

- right-limit transitions $q_0, q_1, \dots, q_n \to P$,
- left-limit transitions $P \to p_0, p_1, \dots, p_m$, and
- paths visiting P from states $p_0, p_1, \dots, p_m$ to states $q_0, q_1, \dots, q_n$, such that at least one of them is a single successor transition.

5.3.2 Procedure

Let $q_0, q_1, \dots, q_n, \dots, q_N \to P \to p_0, p_1, \dots, p_m, \dots, p_M$ be a set of limit transitions, where $\{q_0, q_1, \dots, q_n, p_0, p_1, \dots, p_n\} \subseteq P$ and $\{q_{n+1}, \dots, q_N, p_{m+1}, \dots, p_M\} \cap P = \emptyset$. The implementation that we propose in this work amounts to checking the following properties in the automaton $\tilde{\mathcal{A}}$.

1. States $q_0, q_1, \dots, q_n$ are reachable from states $p_0, p_1, \dots, p_m$ by means of paths visiting states of P only and indexed by closed intervals of $\mathbb{R}$, at least one of which consists in a single transition of successor type (w.l.o.g. from p_0 to q_0).
2. States $p_0, p_1, \dots, p_m$ are co-reachable from states $q_0, q_1, \dots, q_n$ by means of paths visiting states of P only and indexed by closed intervals of $\mathbb{R}$.

3. Each state of P is visited by some path from states $p_0, p_1, \ldots, p_m$ to states $q_0, q_1, \ldots, q_n$ and some path in the other direction.

If all of these properties are satisfied, successor transitions are then added so as to connect every state $q_0, q_1, \ldots, q_N$ to state p_0 and state q_0 to every state $p_0, p_1, \ldots, p_M$.

A pseudocode of this rule is given in Algorithm 1. It consists in performing two breadth-first searches, one from states $p_0, p_1, \ldots, p_m$ to states $q_0, q_1, \ldots, q_n$, and the second in the other direction. Each of these searches follows a sequence of the transitions that respects the following statements.

1. The first and the last transition has the type s.
2. The sequence strictly alternates between transitions of type s and ℓ.
3. The origin and the destination of each transition belong to P.
4. Each type ℓ transition is labelled by a proper subset of P.

Statements 1 and 2 ensure that only closed-$\mathbb{R}$-paths are followed and Statements 3 and 4 ensure that only states which belong to P are visited, and if type ℓ transitions are followed, they result from limit sets smaller than P.

The motivation for a breadth-first search is the following: a necessary condition for the shuffle rule is the existence of a type s transition, that we assumed w.l.o.g. to connect p_0 to q_0. If there exists no such transition, one can immediately conclude that this limit set cannot be visited by a shuffle. Hence, the breadth-first search from the set of states $\{q_0, q_1, \ldots, q_n\}$. By contrast, if there does exist such a transition, one remembers it, in order to use it for adding new transitions if the limit set can be visited by a shuffle.

5.3.3 Complexity

Now, we show that the worst-case time complexity of Algorithm 1 is $\mathcal{O}(|\Delta| \cdot |Q|)$. Indeed, each state of the automaton is explored at most twice (once through incoming transitions of type s and once through incoming transitions of type ℓ), this takes a time scaling as $|Q|$. To make sure that a pair (state, transition type) is not explored twice, a hash set is used, this allows to test membership in constant time and avoid a quadratic time complexity.

At most every transition is analysed, and one tests whether its label is a proper subset of the limit set. As before, testing the membership of a state in a set can be done in constant time by using a hash set. Since at most every state of the automaton belongs to the label of each transition, analysing transitions can take a time which grows linearly with $|Q| \cdot |\Delta|$.

Algorithm 1: Shuffle rule

```
shuffle_rule($q_0, q_1, \ldots, q_N \to P \to p_0, p_1, \ldots, p_M$):
    origins ← $\{q_0, q_1, \ldots, q_N\} \cap P$
    destinations ← $\{q_0, q_1, \ldots, q_N\} \cap P$
    **if** exploration($P$, destinations, forward, transition) **and**
    exploration($P$, origins, backward, $\emptyset$) **then**
        add_transitions(transition)
    **end**
**end**
exploration (limit_set, origins, destinations, direction, transition):
    q ← Queue()
    q.push(origins $\times\{s\}$)
    visited ← Set()
    explored ← Set()
    transition ← $\emptyset$
    **while** $q \neq \emptyset$ **do**
        state, type ← q.pop()
        explore($P$, state, type, origins, destinations, direction, transition)
    **end**
    **return** visited = limit_set
**end**
```

```
explore (limit_set, state, type, origins, destinations, direction, transition):
    if (state, type) $\in$ explored then
        return
    end
    explored.add((state, type))
    next_states $\leftarrow$ get_next_states(state, type, direction)
    is_visited $\leftarrow$ state $\in$ limit_set
    for next $\in$ next_states do
        if next $\in$ limit_set then
            if type $= s$ then
                q.push((next, $\ell$))
                is_visited $\leftarrow \top$
                if direction = forward and transition = $\emptyset$ and
                state $\in$ origins and next $\in$ destinations then
                    transition $\leftarrow$ (state, next)
                end
            end
            if type $= \ell$ and get_label(state, next) $\subsetneq$ limit_set then
                q.push((next, $s$))
                is_visited $\leftarrow \top$
            end
        end
    end
    if is_visited then
        visited.add(state)
    end
end
```

5.4 Implementing the infinite repetition rule

5.4.1 Reminder

Chapter 4 showed how a limit set P, of cardinality i, can be visited by an infinite repetition, and how to compute transitions to be added in this case. One arbitrarily chooses a state $p_1 \in P$ and checks whether

1. there exists a state p_2 reachable from p_1 by a successor transition of $\mathcal{A}$, such that there exists an open $\mathbb{R}$-path from p_2 to p_1 visiting exactly P in $\mathcal{A}_{i-1}$, or
2. there exists a state p_0 from which p_1 is reachable by a successor transition of $\mathcal{A}$, such that there exists an open $\mathbb{R}$-path from p_1 to p_0 visiting exactly P in $\mathcal{A}_{i-1}$.

- If Case 1 holds then, one converts every right-limit transition $q \to P$ into a successor transition from q to p_1 and every left-limit transition $Ptoq$ into a successor transition from p_2 to q'.
- If Case 2 holds then, similar transitions are added, connecting q to p_0 and p_1 to q'.

5.4.2 Procedure

This operation is implemented as follows. Given the limit set P of cardinality $i > 1$, one arbitrarily picks a state $p_1 \in P$, and then checks whether p_1 can be reached from itself,

1. by a transition of type s followed by an open $\mathbb{R}$-path visiting exactly P, or
2. by an open $\mathbb{R}$-path visiting exactly P, followed by a transition of type s.

This results in Algorithm 2, based on two depth-first searches inspired by Tarjan's strongly connected component algorithm [19]. The main idea behind Tarjan's algorithm is to explore a graph through a depth-first search and to maintain a LIFO stack that stores states already explored and annotations about them. Each vertex is associated with

- an index equal to the number of vertices explored before it, and
- the least index of a vertex which can be reached from it.

Then, if a vertex v cannot be reached from a vertex strictly below it on the stack, one knows that v and every vertex above it on the stack form a strongly connected component. This component is popped from the stack and the exploration of the graph continues.

The motivation for a Tarjan-inspired solution comes from the fact that we want to check whether the set of states P is a strongly connected component, in the sense of $\mathbb{R}$-paths. It is also required to know if the component leaves p_1 by following a transition of type s or ℓ.

Our implementation consists in virtually duplicating every state p of P into the tuples (p, s) and (p, ℓ), where the second component, s or ℓ, denotes the transition type used to leave state p. This follows the same strategy as in Algorithm 1, for the shuffle rule. Then, we compute the strongly connected components by starting the search from (p_1, s). If such a component contains (p_1, s) and the remaining states of P, regardless of their associated transition type, we deduce that Case 1 holds and add appropriate transitions. After that, we perform a second computation of strongly connected components, starting the search from (p_1, ℓ). Similarly, if there exists a strongly connected component containing (p_1, ℓ) and the remaining states of P, then we deduce that Case 2 holds and add appropriate transitions.

We use two data structures:

- A stack containing tuples of the form (p, t), where $t \in \{s, \ell\}$.
- A dictionary having as keys tuples present in the queue and as payloads the two corresponding indices of Tarjan's algorithm, namely `id` and `min_id`. This allows to check in constant time whether a tuple is on the stack as well as to update its indices.

Note that the return value of recursive calls of the function `explore_scc` is ignored. This is because we are only interested in a strongly connected component involving the state and the transition type from which the search was initiated.

Algorithm 2: Infinite repetition rule

```
infinite_repetition_rule($q_0, q_1, \dots, q_n \to P \to p_0, p_1, \dots, p_m$):
    choose state $\in P$
    for $type \in \{s, \ell\}$ do
        s ← Stack()
        explored ← Dictionary()
        if explore_scc(state, type, $P$, 0, transition, s, explored) then
            add_transitions(transition)
        end
    end
end
explore_scc(state, type, limit_set, $i$, transition, s, explored):
    explored.add((state, type), id = $i$, min_id = $i$)
    s.push((state, type))
    $i \leftarrow i + 1$
    next_states ← get_next_states(state, type, forward)
    for next $\in$ next_states do
        if next $\notin$ limit_set or (type = $\ell$ and $\neg$ (get_label(state, next) $\subsetneq$ limit_set )) then
            continue
        end
        if (next, inverse_type(type)) $\notin$ explored.keys then
            explore_scc(next, inverse_type(type), $i$, transition, s, explored, origin)
            explored[(state, type)].min_id ← min( explored[(state, type)].min_id,
              explored[(next, inverse_type(type))].min_id)
        else
            explored[(state, type)].min_id ← min(explored[(state, type)].min_id,
              explored[(next, inverse_type(type))].id)
        end
    end
    **if** explored[(state, type)][0] = explored[(state, type)][1] **then**
        destination ← $\emptyset$
        origin_index ← $\emptyset$
        states ← Set()
        **do**
            tuple ← s.pop()
            states.add(tuple[0])
            **if** tuple[1] = $\ell$ **and** explored[tuple].id ∈ $\{1, 2\}$ **and** destination = $\emptyset$ **then**
                destination ← tuple[0]
                origin_index ← explored[tuple].id $-1$
            **end**
            **if** explored[tuple].id = origin_index **then**
                transition ← (tuple[0], destination)
            **end**
        **while** tuple ≠ (state, type);
        **if** limit_set = states **then**
            **return** $\top$
        **end**
    **end**
    **return** $\bot$
**end**
```

5.4.3 Complexity

Algorithm 2 has the same complexity as Algorithm 1, namely $\mathcal{O}(|\Delta| \cdot |Q|)$, for similar reasons:

- Each state is visited at most twice and states already visited during the procedure are sorted in a hash table, together with the type of transition that allowed to reach them and annotations.
- Each transition is followed at most once and a proper subset test is to be made for those of type ℓ.

5.5 Implementing the whole test

In this section, we explain how the whole non-emptiness test can be implemented. First, we show how to preprocess a given automaton on linear orderings $\mathcal{A}$. Then, we iteratively call the shuffle rule and the infinite repetition rule and add necessary transitions. Eventually, we search for an open $\mathbb{R}$-path from an initial state to a final one.

5.5.1 Preprocessing the input automaton

As a first step, one orders limit sets by increasing cardinality. Yet, this ordering is only partial since several limit sets may have the same cardinality. Therefore, we add two tie-breaking rules for each limit set pair P, P':

1. if $P = P'$, these sets are merged into a single one,
2. if $P \neq P'$, they are ordered lexicographically.

Merging limit sets that exactly have the same content allows to test the existence of a limit transition associated to a given limit set content in constant time with respect to the number of limit sets in the the automaton.

Then, we replace the label of each successor transition by the symbol $\emptyset$, according to Sections 5.1 and 5.2. Finally, we compute the backward successor transition relation, which is required by the backward depth-first search. We also relate right-limit transitions to their respective limit set. These operations are done by exploring each transition of each state and maintaining data structures which store backward transitions.

5.5.2 Applying the rules and adding transitions

For the core of the test, limit sets are processed one after another according to the order explained in Section 5.5.1. One inputs every such limit set P and all the associated limit transitions to the shuffle and the infinite repetition rules. In Example 25, we show that it is mandatory to apply both rules to each limit set, regardless of the outcome of the other one.

Example 25. *We study the automaton in Figure 21a. Each limit set of cardinality* 2 *can be visited by a shuffle and not by an infinite repetition. This yields the automaton $\mathcal{A}_2$, in Figure 21b. Then, the limit set* $\{1, 2, 3, 4\}$ *is studied.*

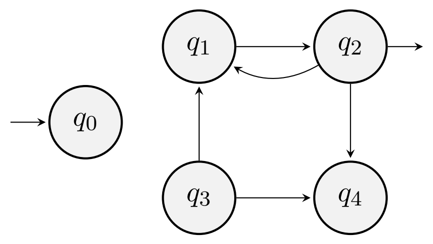
- *It can be visited by a shuffle reading a word of length $\mathbb{R}$, by taking the two successor transitions from q_1 to q_2 and form q_3 to q_4, plus limit transitions associated to the limit set. This gives the automaton in Figure 21c. At this stage, there is not any open $\mathbb{R}$-path from the initial state q_0 to the final state q_2.*
- *It can also be visited by an infinite repetition reading a word of length $\mathbb{R}$, using successor transitions from q_2 to q_1 and from q_3 to q_4, plus red ones, resulting from limit sets of size* 2*. This yields the automaton in Figure 21d.*

One concludes that this automaton accepts at least one word of length $\mathbb{R}$, since there is an open $\mathbb{R}$-path from the initial state q_0 to the final state q_2.

5.5.3 Searching for an open $\mathbb{R}$-path

The question of whether an automaton on linear orderings $\mathcal{A}$ accepts one word of length $\mathbb{R}$ is answered by checking the existence of an open $\mathbb{R}$-path from an initial state to a final one, in the associated automaton $\tilde{\mathcal{A}}$.

The solution that we propose consists in performing a depth-first search from the set of initial states to the set of final ones, by following open $\mathbb{R}$-paths. This means that initial states are left by following transitions of type ℓ. Then one strictly alternates between transitions of type ℓ and s, until a last transition of type ℓ is followed, to reach a final state. As soon as a final state is reached by an open $\mathbb{R}$-path, the procedure stops and answers **yes**. If every open $\mathbb{R}$-path from every initial state has been explored and does not lead to a final state, the procedure answers **no**. A recursive solution is given by Algorithm 3. Note that at line 15, when testing whether an accepting state has just been reached by an open $\mathbb{R}$-path, we test if it is final and if the type of transition allowed to leave it is s, which is equivalent to saying it has just been reached by a transition of type ℓ.

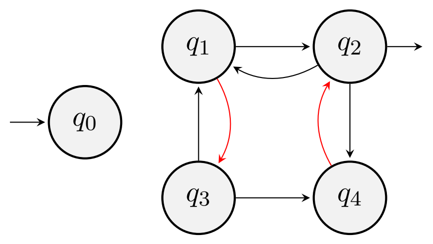


$q_1 \to \{q_1, q_3\} \to q_3$

$q_4 \to \{q_2, q_4\} \to q_2$

$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3$

(a) Input automaton $\mathcal{A}$.

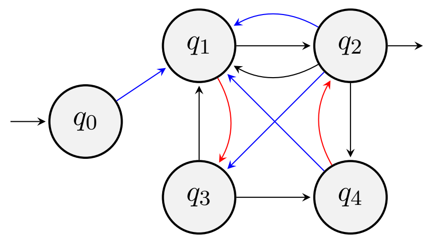


$q_1 \to \{q_1, q_3\} \to q_3$

$q_4 \to \{q_2, q_4\} \to q_2$

$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3$

(b) Automaton $\mathcal{A}_2$.

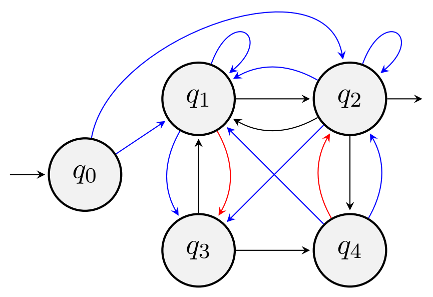


$q_1 \to \{q_1, q_3\} \to q_3$

$q_4 \to \{q_2, q_4\} \to q_2$

$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3$

(c) Automaton $\mathcal{A}_4$, after the shuffle rule.



$q_1 \to \{q_1, q_3\} \to q_3$

$q_4 \to \{q_2, q_4\} \to q_2$

$q_0, q_2, q_4 \to \{q_1, q_2, q_3, q_4\} \to q_1, q_3$

(d) Automaton $\mathcal{A}_4$, after both rules.

Figure 21: Automaton for which the two rules are complementary.

Algorithm 3: Final search

```
final_search($\mathcal{A}$):
    explored ← Set()
    for $q \in I$ do
        if explore(q, $\ell$) then
            return $\top$
        end
    end
    return $\bot$
end
explore(state, type):
    if (state,type) ∈ explored then
        return $\bot$
    end
    explored.add((state, type))
    if state ∈ $F$ and type = $s$ then
        return $\top$
    end
    next_states ← get_next_states(state, type, forward)
    for next ∈ next_states do
        return explore(next, inverse_type(type))
    end
    return $\bot$
end
```

Note that according to the application, it may or may not be relevant to perform this search before having studied all limit sets. In the next section, we show that this does not change the worst-case time complexity.

5.5.4 Overall complexity

Now, we show that the time complexity of the complete procedure is $\mathcal{O}\left(|\Delta|^2 \cdot |Q|\right)$, by listing the respective time complexities of each part of the test.

- Computing the backward transition relation is done in $\mathcal{O}\left(|Q| + |\Delta|\right)$ time, since every state and every successor transition needs to be considered exactly once;
- Sorting limit sets is done in $\mathcal{O}\left(|\Delta| \cdot |Q| \log\left(|Q|\right) + |\Delta| \cdot \log\left(|\Delta|\right) \cdot |Q|\right)$, indeed, there are at most $|\Delta|$ limit sets, each containing at most $|Q|$ states which are sorted lexicographically, then, $|\Delta|$ limit sets are sorted and each comparison between two sets requires to enumerate their whole content in the worst case;

- Each of the two rules is called at most $|\Delta|$ times and each of them has the time complexity $\mathcal{O}(|\Delta| \cdot |Q|)$;
- The final search has the time complexity $\mathcal{O}(|Q| + |\Delta|)$.

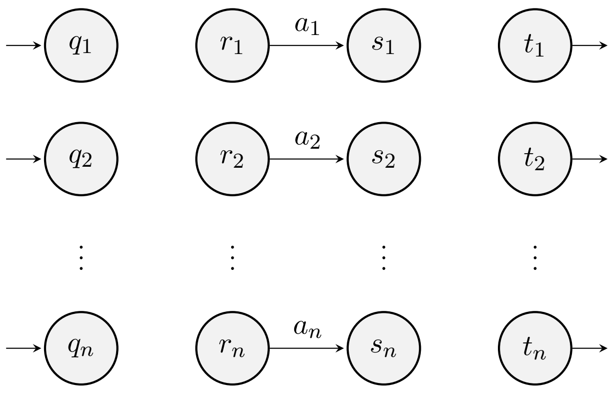
Since the “final” search has a time complexity less than either rule, performing it once in the end of after processing each limit set does not change the asymptotic complexity of the complete test.

5.5.5 Practical implementation and experimental results

We now give some metrics about the computer program implementing the whole test. It is entirely written in the C language, as a part of the LASH toolset [14]. It is composed of about 4000 lines of code, of which 1000 are shared with Tom Clara, defining the implementation of automata on linear orderings as a data structure, as well as some basic data structures such as FIFO queues. In the remaining part of this section, we give some experimentally measured runtimes. Results were obtained on a laptop equipped with 8 GB of RAM and an AMD Ryzen 5 3500U CPU working at 3.7 GHz.

Studying a single shuffle

To evaluate the performances our program, we start by considering several instances of a variable-sized automaton and measure the time taken to complete the test upon it. This variable-sized automaton, $\mathcal{A}_n$, accepts the language $\mathscr{L}(sh(a_1, a_2, \ldots, a_n))$ and its structure is shown in Figure 22. It has a single limit set of size $2\,n$, associated with $2\,n$ right-limit transitions and $2\,n$ left-limit transitions. The theoretical relation between the time taken to analyse the automaton and its size is linear. This is due to the fact that it has a single limit set and thus, there is not any subset test to be performed and the rules are used only once. Figure 23 contains practical measurements, indeed having a linear behaviour. Note that it takes less than 1.5 seconds to process an automaton having 400,000 states.



$$q_1, q_2, \ldots, q_n, s_1, s_2, \ldots, s_n \to \{r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_n\} \to r_1, r_2, \ldots, r_n, t_1, t_2, \ldots, t_n$$

Figure 22: Automaton accepting the shuffle of n symbols.

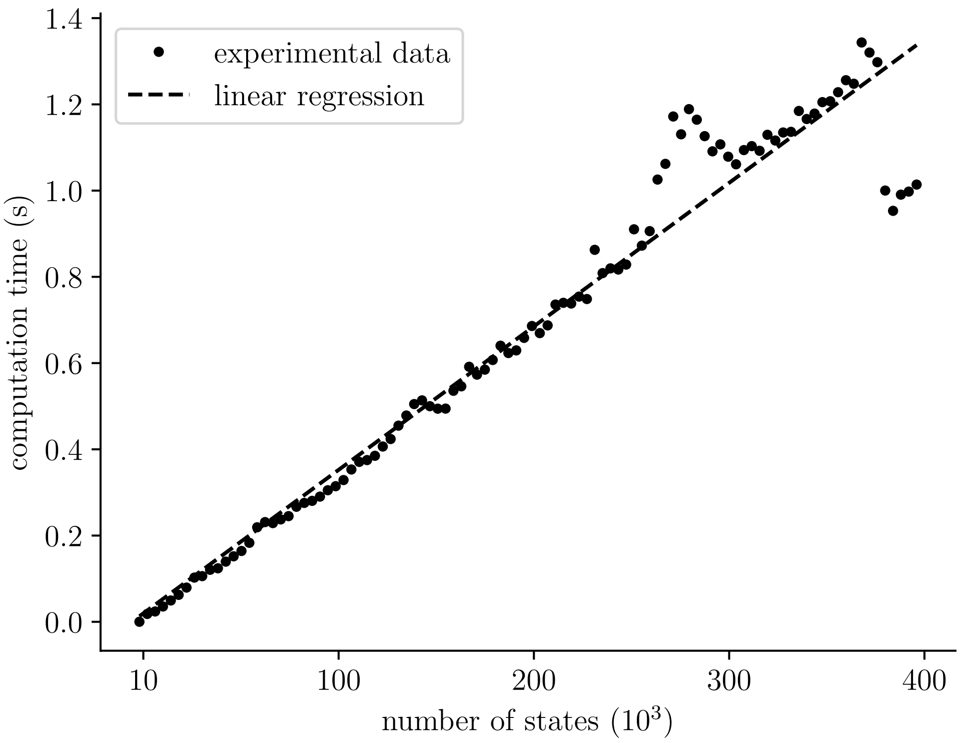


Figure 23: Duration of the analysis of automata accepting the shuffle of up to 100,000 symbols.

Studying nested shuffles

We now measure the time taken to process an automaton having m nested limit sets, each one being associated to a shuffle of n languages. Figure 24 shows an instance of this automaton for $m = 2$. Each member of this automaton family accepts words of length $\mathbb{R}$, composed of a Cantor shuffle of depth $m - 1$. We recursively define such a shuffle as follows:

- A Cantor shuffle of depth 0 is denoted by the rational expression

$$sh\left(a_1, a_2, \ldots, a_n\right),$$

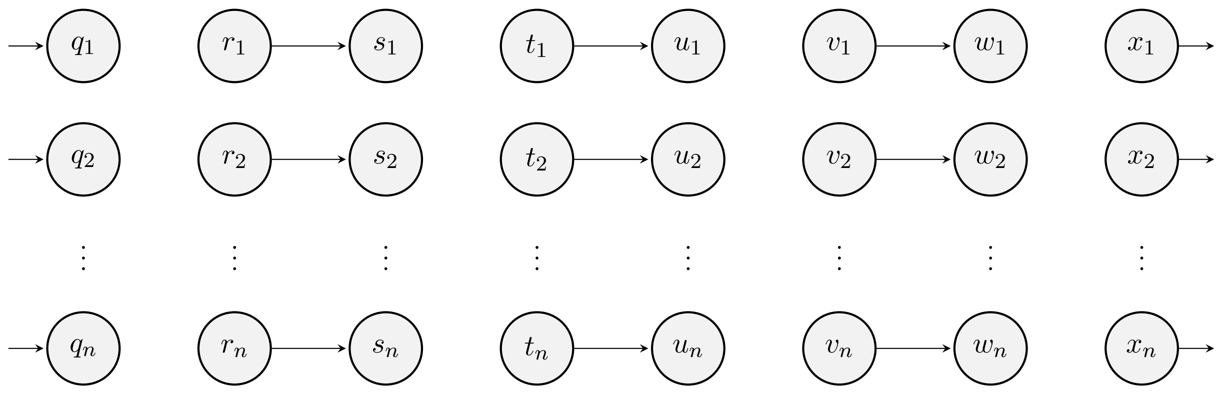
  for some $n > 0$ and where each $a_1, a_2, \ldots, a_n$ is a letter of the alphabet.

- A Cantor shuffle of depth $i > 0$ is the language denoted by the rational expression

$$sh\left(a_1 \cdot X \cdot a_2, a_3, a_4, \ldots, a_n\right),$$

  where X is a rational expression denoting a Cantor shuffle of depth $i - 1$ and each $a_1, a_2, \ldots, a_n$ is a letter of the alphabet.

The test which we performed was done on instances of automata accepting a Cantor shuffle of depth up to 1024, and for which the parameter n equals 100. The number of limit sets in these automata grows linearly with their number of states. The complexity of the test on this example thus is $\mathcal{O}\left(|Q|^3\right)$. Figure 25 compares recorded runtimes with a cubic function of the automaton size, on a log-log scale. Runtimes grow with a slope of 1.2 for instances of the automaton greater than 10,000 states, which is significantly less than the worst-case value of 3. The practical cost of the procedure on this automaton is thus much better than the theoretical worst-case complexity.



$$s_1, s_2, \ldots, s_n, u_1, u_2, \ldots, u_n \to \{r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_n\} \to r_1, r_2, \ldots, r_n, v_1, v_2, \ldots, v_n$$

$$s_1, s_2, \ldots, s_n, w_1, w_2, \ldots, w_n, q_1, q_2, \ldots, q_n \to \{r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_n, t_1, t_2, \ldots, t_n,$$
$$u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\} \to r_1, r_2, \ldots, r_n, t_1, t_2, \ldots, t_n, x_1, x_2, \ldots, x_n$$

Figure 24: Automaton accepting a Cantor shuffle of depth 1.

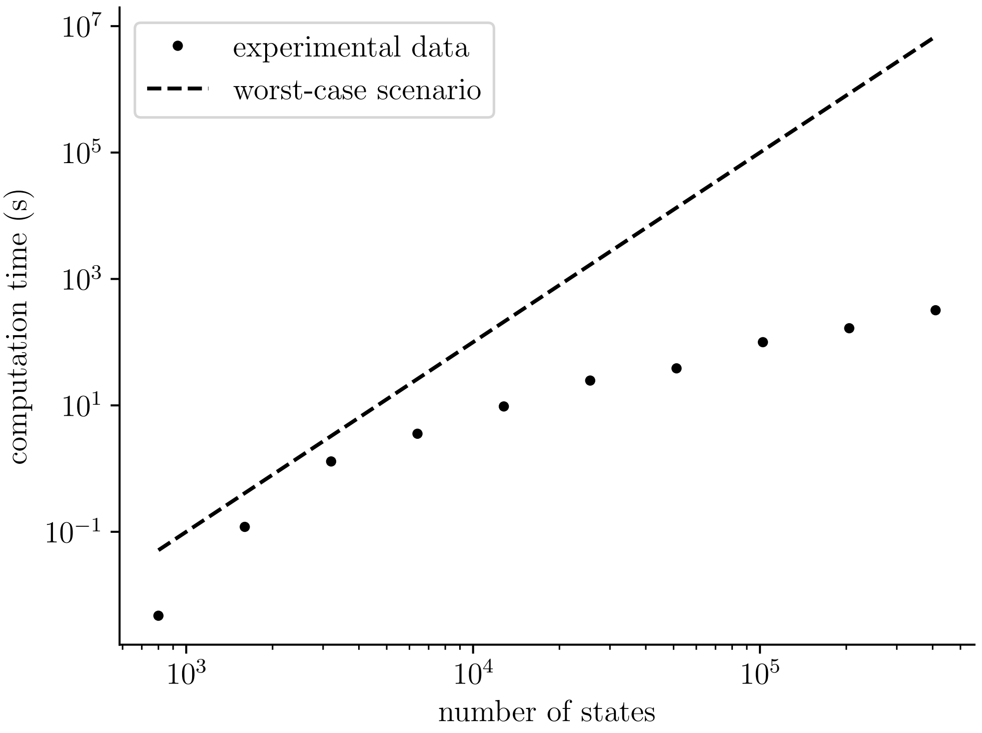


Figure 25: Duration of the analysis of automata accepting Cantor shuffles of depth up to 1024.

Studying a an automaton accepting models of a formula

To conclude this experimental section, we test our program on a automaton accepting the models of a formula belonging to the first-order theory of order over the real numbers. We refer the reader to Appendix B for explanations about the decision procedure. We consider instances of the following formula, for several values of the parameter n.

$$\begin{aligned}\varphi_n : (\forall x, y : x < y \Rightarrow (\exists z, w : x < z < y & \wedge P(z) \wedge x < w < y \wedge \neg P(w))) \wedge \\ & x_1 < x_2 < \cdots < x_n \wedge P(x_1) \wedge \neg P(x_2) \wedge \cdots \wedge P(x_n)\end{aligned}$$

Intuitively, φ_n states that

1. variables are partitioned into two dense subsets, one for which P is true and another one for which P is false, and
2. P is true for every other variable $x_1, x_2, \ldots, x_n$, chosen in increasing order, and false for the rest of them.

When the domain is chosen to be the real numbers, this formula is satisfiable. For instance, P can denote the set of rational numbers and the variables $x_1, x_2, \ldots, x_n$ can be increasingly sorted real numbers, where every other one is rational and the remaining ones are irrational.

Although a general solution for the universal quantification is not yet available, we can compute by hand an automaton accepting models of the sub-formula expressing Statement 1. This automaton needs to accept the shuffle of two symbols. Other automata consist in standard atoms. The intersection of these automata is then computed by the program developed by Tom Clara [12]. Figure 26 provides an illustration of the result for φ_2. In each label, the first row is associated to P and the two remaining ones to x_1 and x_2, respectively. Then, the non-emptiness test is performed on this resulting automaton and its result obviously is positive. Recorded runtimes of the two programs are presented in Figure 27. On this example, we observe that the time dedicated to the non-emptiness test is negligible with respect to the intersection.

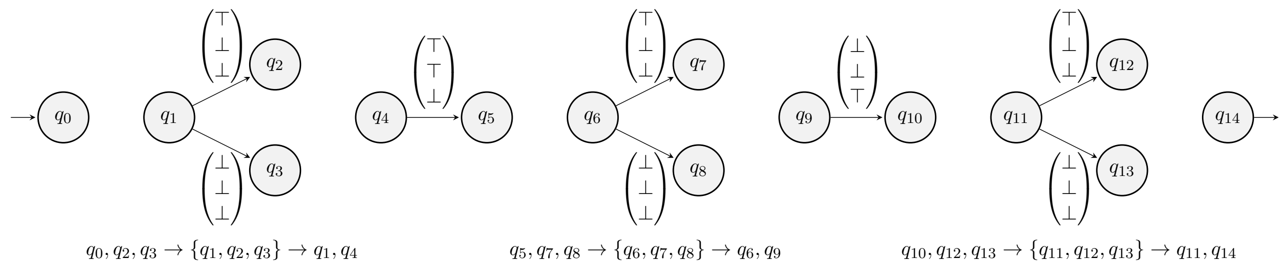


Figure 26: Automaton accepting models of φ_2.

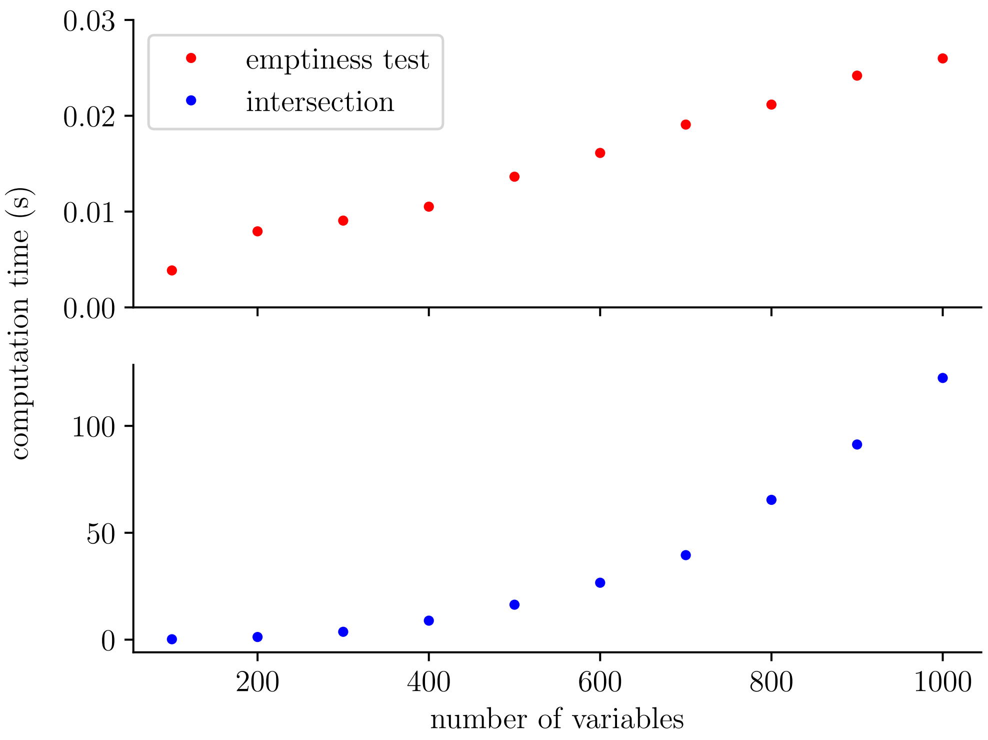


Figure 27: Duration of the computation and analysis of automata accepting instances of φ_n.

Chapter 6

Conclusion and perspectives

To conclude this work, we restate the context in which it took place. Then we summarise its contributions and finally suggest potential improvements and new problems to address.

6.1 Context

This master thesis is a step towards a practical implementation of a decision procedure for the monadic first-order theory of order with uninterpreted predicates on real and rational numbers. This procedure consists in translating a formula into an automaton on linear orderings, that represents the class of its models and then testing whether this automaton accepts at least one word indexed by the domain of interest. The objective was to design and implement a practical generic non-emptiness test deciding whether a given automaton on linear orderings accepts at least one word of length $\mathbb{R}$, based on results obtained by the research group in 2024 [2].

6.2 Contributions

Chapter 2 extends the theory of automata on linear orderings by allowing transitions labelled by the empty word in their transition relation, without affecting their expressive power. We demonstrated their usefulness in theoretical developments by correcting a (minor) error in the original proof of equivalence between automata and rational expressions [1]. It led to an article, co-authored by Bernard Boigelot and Tom Clara [3].

In Chapter 4, we simplified an existing procedure deciding whether an automaton on linear orderings accepts at least one word of length $\mathbb{R}$ [2]. We reduced the number of transitions resulting from the set of limit transitions associated to each limit set. While the original procedure has a quadratic worst-case cost in the size of the input automaton, the one that we propose has a linear complexity.

Finally, in Chapter 5, we presented our practical implementation of the test, as a part of the LASH toolset [14]. Experimental results show that our program can handle automata with more than 100,000 states in less than a few minutes. These experiments, made on non-trivial automata, also suggest that our solution scales well with the size of the input automaton.

6.3 Perspectives

Our implementation does not fully use the property that a limit transition $q \to P$ or $P \to q'$ cannot be enabled by following limit transitions of the form $q' \to P'$ or $P' \to p'$, where $P' \nsubseteq P$.

A first possible improvement would consist in computing limit set inclusion before performing the shuffle and the infinite repetition rules. This operation would have a worst-case asymptotic time complexity equal to the one of the current whole procedure, namely $\mathcal{O}\left(|\Delta|^2 \cdot |Q|\right)$. Yet, the core of the test (that is, calling the two rules and adding transitions for every limit set) would only require $\mathcal{O}\left(|\Delta|^2\right)$ time, since label inclusion could then be checked in constant time. In practice, since many successor transitions can have the same label, this modification would reduce the overall computation time of the procedure.

A second strategy would be to use information about which states are visited when following a transition labelled by a given limit set. This would require to carefully design backtracking mechanisms, to avoid considering a set of states visited before making sure that the transition actually belongs to a path having the desired properties. Again, in practice this could speed up the test but would not lower its asymptotic worst-case complexity.

The fact that the pre-computation of limit set inclusion is the bottleneck of the whole procedure in terms of complexity motivates a more efficient representation of limit sets in automata on linear orderings. In particular, if the data structure which represents limit sets takes into account the hierarchy formed by their relative inclusion, the problem of computing it becomes trivial and the worst-case asymptotic complexity is lowered. The idea of explicitly representing such a hierarchy between limit sets has been exploited by Tom Clara [12], as an internal tool for the intersection. Since, in the scope of the decision procedure, the operation of intersection appears before the non-emptiness, a strategy could consist in using this already computed hierarchical structure to sort limit sets and check their inclusion.

Finally, the formalism of epsilon automata on linear orderings seems to be a promising way of obtaining an automaton that accepts the models of a universally quantified formula. This motivates an extension of the non-emptiness test to epsilon automata on linear orderings.

Appendix A

Implementation of automata on linear orderings

In this appendix (common with Tom Clara), we explain and motivate our implementation of automata on linear orderings as a data structure. This implementation is done as a part of the LASH toolset [14]. This toolset is a library written in the C language able to manipulate finite-state automata, on finite or infinite words. Our contribution consists in extending these automata to automata on linear orderings.

A.1 Original automata

First, we describe the original implementation of automata in LASH. An automaton is represented as a data structure composed of

- an array of *states* Q,
- miscellaneous information, and
- a pointer towards a future extension.

Each state consists of a data structure containing information about initial and final status and an array of *transitions.* A transition consists of:

- Its destination state, represented as the corresponding index in the state array.
- Its label, consisting in a sequence of alphabet symbols. Each alphabet symbol is represented by an array of bytes, whose length is constant across the automaton.

A LASH automaton is thus an annotated adjacency list.

A.2 Automata on linear orderings

An automaton on linear orderings consists of a LASH automaton whose extension stores information about limit transitions. The implementation of our extension consists of a data structure composed of

- an array of *limit sets* P, and
- an array of *right-limit transitions* R.

A limit set is represented by a hash set (provided as a utility by the original toolset) and an array of right-limit transitions. The hash set contains the indices of the states composing it. Right-limit transitions are encoded as follows. An array whose length is the number of states in the automaton contains pointers to limit set index arrays. Each index refers to the limit set involved in the right-limit transition of the automaton.

The reason why right-limit transitions are not represented in the same fashion as left-limit transitions is that many operations require to know which successor and limit transitions can be followed from a given state. Symmetrically, many operations require to know to which states left-limit transitions can be followed. This justifies why left-limit transitions are stored near limit sets.

Remark

Limit sets are stored in the array P in the order in which they are created by the user (this paradigm is borrowed from how states are added in the core of the toolset). This implies in particular that several limit sets can have the same content.

Appendix B

A decision procedure for the monadic first-order theory of order, over $\mathbb{R}$ or $\mathbb{Q}$

In this appendix, we outline the main steps of the automaton-based decision procedure currently being developed in the research group [2]. It is directly inspired by the procedure for deciding S1S [8].

B.1 Encoding assignments

We start by explaining the mechanism used to represent the assignment of a single variable. Let x be a variable belonging to the linear ordering J, formed by the domain of interest. We denote the fact that $x = j$, for some $j \in J$ by the word $w : J \to \{\bot, \top\}$ defined as

$$w : \begin{cases} j \mapsto \top \\ k \mapsto \bot, \quad \forall k \in J \setminus \{j\} \end{cases}$$

This simply means that a word associated to the variable x answers the question "is x equal to this number?", for any number of the domain.

The same mechanism applies to unary predicates. A word w denotes the unary predicate $P(\cdot)$ as follows, $w : J \to \{\bot, \top\}$

$$w : \begin{cases} j \mapsto \top \text{ iff } P(j) = \top, \quad \forall j \in J \\ k \mapsto \bot \text{ iff } P(k) = \bot, \quad \forall k \in J \end{cases}$$

Similarly, an assignment of a formula having n variables and m predicates is a function $w : J \to \{\top, \bot\}^{n+m}$ that answers the question "are the variables equal to j and are predicates true in j?".

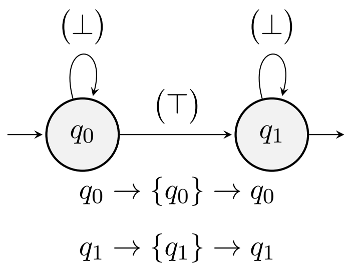
B.2 Representing models of a formula

The class of models of a formula is the language of the encodings of the assignments which satisfy the formula. The idea is to represent this language by an automaton built as follows. One first preprocesses the formula so as to push negations inwards near its atomic sub-formulas, because automata on linear orderings are not closed under complementation [17]. The atoms of a formula are

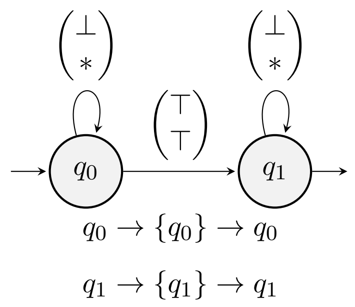
1. x,
2. $P(x)$,
3. $\neg P(x)$,
4. $x < y$,

5. $\neg(x < y)$,

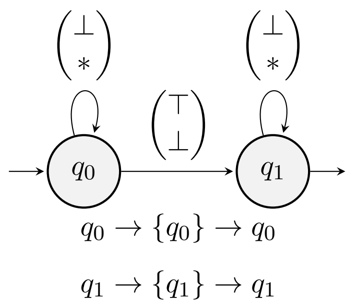
and their negations. Figure 28 presents automata respectively accepting models of these atoms, where the first line of a label denotes the value associated to the variable x and the second one, if any, denotes the predicate P or the variable y. The symbol $*$ means that the truth value of the parameter does not matter.



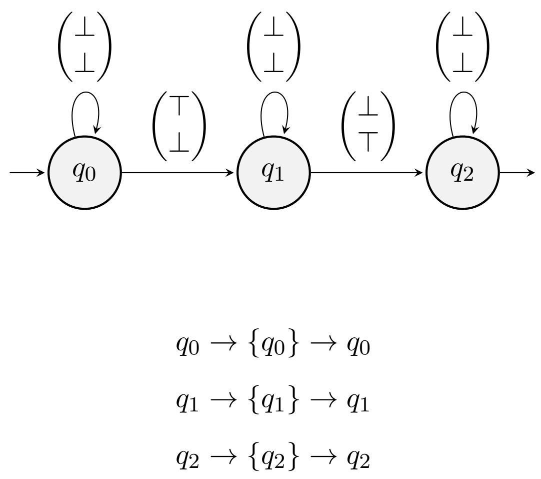
(a) x



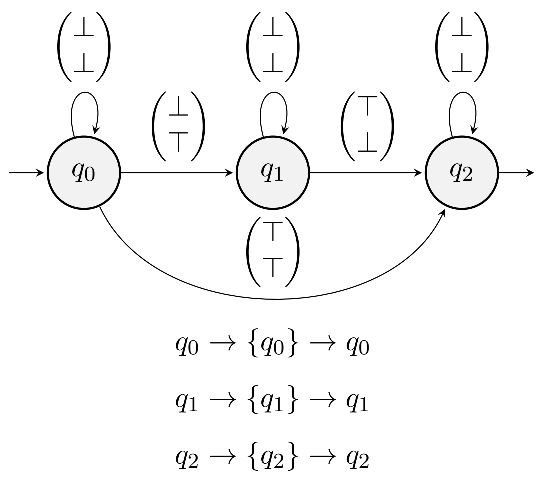
(b) $P(x)$



(c) $\neg P(x)$


$\binom{\bot}{\bot}$
$\binom{\top}{\bot}$
$\binom{\bot}{\top}$
q_0
q_1
q_2
$q_0 \to \{q_0\} \to q_0$
$q_1 \to \{q_1\} \to q_1$
$q_2 \to \{q_2\} \to q_2$

(d) $x < y$



(e) $\neg(x < y)$

Figure 28: Atomic automata.

We explain the intuition behind these automata. For instance, the automaton in Figure 28a denotes a single first-order variable. This means that there must be a single element of the linear ordering that is mapped to $\top$, and that every other element should be mapped to $\bot$. This automaton therefore consists in two sub-automata able to read any word on the alphabet $\{\bot\}$ (remember Example 15), connected by a single successor transition reading a unique occurrence of the letter $\top$. Similarly, the automaton in Figure 28d is composed of three sub-automata, each able to read words of any length on the alphabet $\{(\bot, \bot)\}$. Two successor transitions connect them. A first one reads the letter $(\top, \bot)$, to asses the fact that x comes before y, a second one assesses the existence of y.

The next step consists in combining the atomic automata according to Boolean connective of the formula and quantifiers. The operation of union and intersection of automata on linear orderings is addressed by Tom Clara [12]. As for automata on finite words, theses operations can be done by generating an automaton which simulates the joint behaviour of connected automata. The operation of existential quantification can be done by removing labels associated to existentially quantified variables. A last operation is universal quantification. The absence of complementation in automata on linear orderings does not allow to circumvent the universal quantification by double complementation and existential quantification. Universal quantification on automata on linear orderings still is under research in the group and results have already been obtained for automata on infinite words [5].

B.3 Testing the non-emptiness on the domain $\mathbb{R}$ or $\mathbb{Q}$

Up to now, the automaton representing the set of models of a formula does not make any assumption on the domain to which variables belong. The role of this step is to decide whether the automaton accepts at least one word indexed by $\mathbb{R}$ or $\mathbb{Q}$. In the positive case, the formula is satisfiable and otherwise it is not.

B.3.1 Remark

Automata in Figure 28 are fully independent on the domain of interest. Yet, a practical implementation can choose more specific automata, already taking into account some properties of the domain, so as to simplify operations made on them afterwards.

List of Figures

Bibliography

[1] Bès, A., Carton, O.: A Kleene theorem for languages of words indexed by linear orderings. International Journal of Foundations of Computer Science **17**(03), 519–541 (2006)

[2] Boigelot, B., Fontaine, P., Vergain, B.: Non-emptiness test for automata over words indexed by the reals and rationals. In: Proc. 28th International Conference on Implementation and Application of Automata (CIAA). Lecture Notes in Computer Science, vol. 15015, pp. 94–108. Springer (2024)

[3] Boigelot, B., Braipson, T., Clara, T.: Epsilon automata on linear orderings, submitted for publication

[4] Boigelot, B., Fontaine, P., Vergain, B.: Deciding reachability in automata on words indexed by the reals and rationals, submitted for publication

[5] Boigelot, B., Fontaine, P., Vergain, B.: Universal first-order quantification over automata. In: Proc. 27th International Conference on Implementation and Application of Automata (CIAA). Lecture Notes in Computer Science, vol. 14151, pp. 91–102. Springer (2023)

[6] Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bulletin of the Belgian Mathematical Society **1**(2), 191–238 (1994)

[7] Bruyère, V., Carton, O.: Automata on linear orderings. Journal of Computer and System Sciences **73**(1), 1–24 (2007)

[8] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. International Congress on Logic, Methodology and Philosophy of Science. pp. 1–12. Stanford University Press (1962)

[9] Cantor, G.: Über unendliche, lineare Punktmannichfaltigkeiten. Mathematische Annalen **20**, 113–121 (1882)

[10] Cantor, G.: Beiträge zur Begründung der transfiniten Mengenlehre. Mathematische Annalen **46**, 481–512 (1895)

[11] Carton, O.: Accessibility in automata on scattered linear orderings. In: Mathematical Foundations of Computer Science 2002. pp. 155–164. Springer Berlin Heidelberg (2002)

[12] Clara, T.: Efficient representation and manipulation of automata on linear orderings. Master's thesis, University of Liège (2024-2025)

[13] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 3rd Edition. Pearson international edition, Addison-Wesley (2007)

[14] The Liège Automata-based Symbolic Handler (LASH), available at : `https://people.montefiore.uliege.be/boigelot/research/lash/`

[15] Muller, D.E.: Infinite sequences and finite machines . In: Proc. 4th Annual Symposium on Switching Circuit Theory and Logical Design. pp. 3–16. IEEE Computer Society (1963)

[16] Perrin, D., Pin, J.: Infinite words - automata, semigroups, logic and games, Pure and applied mathematics series, vol. 141. Elsevier Morgan Kaufmann (2004)

[17] Rispal, C., Carton, O.: Complementation of rational sets on countable scattered linear orderings. International Journal of Foundations of Computer Science **16**(04), 767–786 (2005)

[18] Rosenstein, J.G.: Linear orderings. Academic press (1982)

[19] Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)

Declaration on the use of automatic tools for writing the manuscript

This page contains declarations about the use of automatic tools used to write this document. It is a requirement of the master in electrical engineering, details can be found here.

- I hereby certify that I have not used any generative intelligence tool in the writing of text, graphics, images, or data reproduced in this manuscript.
- I used the British English spell checker provided by Overleaf on the English parts of the document and the French spell checker provided by Overleaf on the French part of the document.