Eva Maia

On the Descriptional Complexity of Some Operations and Simulations of Regular Models

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

2015

Eva Maia

On the Descriptional Complexity of Some Operations and Simulations of Regular Models

Tese submetida à Faculdade de Ciências da
Universidade do Porto para obtenção do grau de Doutor
em Ciência de Computadores

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2015

Aos meus pais.

Ao Hélder.

Agradecimentos

Em primeiro lugar quero agradecer aos meus orientadores, Nelma Moreira e Rogério Reis, por acreditarem e confiarem no meu trabalho. Agradeço pelo apoio incansável, pela capacidade de ensinar, pela constante disponibilidade, incentivo e paciência. Por tudo, manifesto o meu profundo reconhecimento.

Aos meus colegas, em geral, por todos os momentos partilhados. Em especial, agradeço à Ivone toda a amizade, compreensão, diálogo e todos os momentos de loucura que teve que suportar. Sem ela não teria conseguido chegar aqui. Agradeço ainda à Alexandra e à Isabel por todo o apoio administrativo e emocional, por me ouvirem, me aturarem e estarem sempre dispostas a ajudar.

Agradeço à minha família, em especial aos meus pais e ao Hélder, que sempre me apoiaram e incentivaram em todas as minhas decisões. Aos meus pais agradeço todo o esforço e sacrifício para me proporcionarem uma boa educação. Ao Hélder todo o apoio, compreensão e carinho em todos os momentos, bons e maus, da nossa vida.

Considerando o suporte financeiro, agradeço à Fundação para a Ciência e Tecnologia pela bolsa de doutoramento [SFRH/BD/78392/2011], e ao Centro de Matemática da Universidade do Porto (UID/MAT/00144/2013), que é financiado pela FCT (Portugal) com os fundos estruturais nacionais (MEC) e europeus através de programas FEDER, sob o acordo de parceria PT2020, por financiar todas as despesas inerentes às minhas deslocações às várias conferências.

Abstract

Descriptional complexity studies the complexity measures of languages and their operations. These studies are motivated by the need to have good estimates of the amount of resources required to manipulate a given language. In general, having succinct objects will improve our software, which may consequently become smaller and more efficient.

The descriptional complexity of regular languages has recently been extensively investigated. Usually, the authors consider worst-case analysis, but this is not enough to a complete description of the objects and algorithms. Normally, the worst-case complexity does not reflect the real life algorithm performance, and this stimulates the study of the average-case complexity of these algorithms.

We study several properties of regular languages, improving or developing new simulation methods, and identifying which methods have better practical performance. We start to analyse the descriptional complexity of several operations over regular languages, considering incomplete deterministic finite automata. Then, we present some simulation methods of regular expressions by finite automata, and study their complexity. In both cases, we do not only focus on the worst-case analysis, but we also study some aspects of its average-case complexity.

Resumo

A complexidade descritiva estuda as medidas de complexidade das linguagens e das suas operações. Estes estudos devem-se à necessidade de ter boas estimativas da quantidade de recursos necessária para manipular uma dada linguagem. Em geral, ter objectos sucintos melhora o nosso software, que se torna menor e mais eficiente.

A complexidade descritiva das linguagens regulares tem sido muito estudada nos últimos tempos. Normalmente, os autores consideram a análise no pior caso, mas isto não é suficiente para uma completa descrição dos objectos e dos algoritmos. Geralmente, a complexidade no pior caso não reflecte o desempenho real dos algoritmos, o que estimula o estudo da complexidade no caso médio.

Neste trabalho, estudámos várias propriedades das linguagens regulares, melhorando ou desenvolvendo novos métodos de simulação, e identificando quais os métodos com melhor desempenho. Começámos por analisar a complexidade descritiva de várias operações nas linguagens regulares, considerando autómatos finitos determinísticos incompletos. Depois, apresentámos alguns métodos de simulação de expressões regulares em autómatos finitos, e estudámos a sua complexidade. Em ambos os casos, não nos focamos apenas na análise no pior caso, estudamos também alguns aspectos da complexidade no caso médio.

Résumé

La complexité des descriptions des langages formels étudie les mesures de la complexité des langages et de leurs opérations. Ces études sont motivées par la nécessité de avoir bonnes estimations de la quantité de ressources nécessaires pour manipuler un langage donnée. En général, ayant objets succincts permettront d'améliorer notre logiciel, qui peut par conséquent devenir plus petit et plus efficace.

La complexité des langages rationnels a récemment été largement étudiée. Généralement, les auteurs envisage la analyse dans le pire des cas, mais cela ne suffit pas à une description complète des objets et des algorithmes. Normalement, la complexité dans le pire des cas, ne reflète pas le comportement pratique des algorithmes, et ça fait l'importance de l'étude de la complexité en moyenne.

Nous étudions plusieurs propriétés de langages rationnels, l'amélioration ou le développement de nouvelles méthodes, et l'identification des méthodes qui ont un meilleur comportement pratique. Nous commençons pour analyser la complexité de plusieurs opérations sur les langages rationnels, considérant automates finis déterministes non complets. En suite, nous présentons certaines méthodes de simulation de expressions rationnelles par automates finis, et nos étudions leur complexité. Dans les deux cas, nous ne nous concentrons pas uniquement sur l'analyse dans le pire des cas, mais nous étudions aussi certains aspects de sa complexité en moyenne.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Regular languages and finite automata are one of the oldest topics in formal language theory: its formal study has been done for more than 60 years [Kle56]. Many have believed that everything of interest about regular languages is already known, however a lot of new and interesting results have been coming out recently. This is due to the application of regular languages and finite automata in areas such as software engineering, programming languages, parallel programming, network security, formal verification, natural language and speech processing.

In recent years, a number of software systems that manipulate automata, regular expressions, grammars, and related structures have been developed. Examples of such systems are AGL [Kam], AMoRE [Emi], FAdo [MR], Grail+ [oPEI], JFLAP [RFL], MONA [DoCS], Unitex [lV], Vaucanson [SL] and GAP [Gro].

The increasing number of practical applications and implementations of regular languages motivates the study of two kinds of complexity issues. On the one hand it is important to study the time and space needed for the execution of the processes. On the other hand, the succinctness of the model representations (descriptional complexity) is crucial, because having smaller objects permit us to improve the efficiency and the reliability of the software.

The studies on descriptional complexity can be divided into two different approaches. The representational complexity, which studies the complexity of simulations between models by comparing the sizes of different representations of the same formal languages; and the operational complexity, which studies the complexity of operations on languages. Authors typically present the worst-case complexity analysis, but that does not provide enough information on the practical behaviour. Despite its evident practical importance, the average-case complexity is not widely studied.

In this work we use the two above mentioned approaches for the study of descriptional complexity. First, we study the descriptional complexity of several operations on incomplete deterministic finite automata. Then, we present some simulation methods of regular expressions by finite automata and study their complexity. In both cases, we do not limit ourselves to the worst case analysis, studying some aspects of the average-case complexity.

All the source code developed was integrated on the FAdo project, and it is freely available from http://fado.dcc.fc.up.pt/.

1.1 Structure of this Dissertation

Chapter 2 presents some basic notions and definitions of language theory. We also introduce deterministic (DFA) and nondeterministic finite automata (NFA). The notion of regular expressions (REs) is exposed as well as their relation with finite automata. In particular, we consider a new method to convert REs to NFAs: the Previous Automaton.

In Chapter 3 we introduce the descriptional complexity of formal languages. First of all, we review the state and transition complexities of some individual regularity preserving language operations on regular languages, considering the worst-case analysis. Then, we refer a few results known on the average-case state complexity. We also review some analytic combinatorics methods which are useful to study the asymptotic

average size of models.

In Chapter 4 we study the state and transition complexity of some operations on regular languages based on non necessarily complete DFAs. This work was started by Gao et al. [GSY11], considering the worst-case analysis. We extend the analysis to the concatenation, the Kleene star and the reversal operations [MMR13b]. For these operations tight upper bounds were found. We also found a tight upper bound for the transition complexity of the union, which refutes the conjecture presented by Gao et al.. Then, we extend this line of research by considering the class of finite languages, finding tight upper bounds for all basic operations that preserve regularity [MMR13a]. We correct the upper bound for the state complexity of concatenation presented by Câmpeanu et al. [CCSY01], and show that if the *right* operand automaton is larger than the *left* one, the upper bound is only reached using an alphabet of variable size, contrary to what was stated by the same authors. We also performed some experimental tests in order to understand how significant are the wort-case results.

Chapter 5 presents a study of the partial derivative automaton, $\mathcal{A}_{\mathrm{PD}}$, and several of its properties. For regular expressions without Kleene star we characterise this automaton and we prove that it is isomorphic to the bisimilarity of the position automaton, under certain conditions [MMR14]. It is also shown that, in general, a partial derivative automaton A cannot be converted to a regular expression that is linear in the size of A. Still in this chapter, we present the right derivatives, with which we construct the right derivative automaton, and show its relation with Brzozowski's automaton. Using the notion of right-partial derivatives, we define the right-partial derivative automaton $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$, and we characterise its relation with $\mathcal{A}_{\mathrm{PD}}$ and position automaton, $\mathcal{A}_{\mathrm{Pos}}$. We also present a new construction method of the $\mathcal{A}_{\mathrm{Pre}}$ automaton, introduced by Yamamoto [Yam14], and show that it also is a quotient of the $\mathcal{A}_{\mathrm{Pos}}$ automaton. Considering the framework of analytic combinatorics we study the average size of $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Pre}}$ automata [MMR15b].

We finally conclude with some final remarks and possible future work on Chapter 6.

Chapter 2

Preliminaries

In this chapter we present some basic notions and definitions of language theory that we will use throughout this thesis. For more details, we refer the reader to the standard literature [HU79, Yu97, Sha08, Sak09]. We also define a new automaton, the Previous automaton, that does not appear in the literature.

2.1 Formal Languages

In the context of formal languages, an *alphabet* is a finite non-empty set of symbols, or letters, e.g. $\{a, b, c\}$ or $\{1, 2\}$. In this work we denote any alphabet by Σ. A finite sequence of symbols from an alphabet Σ is called a *word*. For example, with $\Sigma = \{a, b, c\}$, a or aba are words over Σ. The *length* of a word w, denoted by $|w|$, is the number of symbols or letters in w. For instance $|aba| = 3$. To represent the empty word, i.e., a word without any symbol or letter, we use the symbol ε. Naturally $|\varepsilon| = 0$.

The set of all words over an alphabet Σ is denoted by $\Sigma^\star$. Note that this is an infinite set with words of finite length.

The concatenation of two words $w = w_1 \cdots w_k$ and $w' = w'_1 \cdots w'_{k'}$, both with alphabet

Σ, denoted by $w \cdot w'$ or ww', is the word $w_1 \cdots w_k w'_1 \cdots w'_{k'}$. The empty string is the identity for the concatenation operation: $w\varepsilon = \varepsilon w = w$. The concatenation is associative: $(w_1 w_2)w_3 = w_1(w_2 w_3)$. Thus, the set $\Sigma^\star$ with word concatenation is a monoid.

We denote w^n as the word obtained by concatenating n copies of w:

$$\begin{aligned} w^0 &= \varepsilon, \\ w^{n+1} &= w^n w. \end{aligned}$$

For instance, $(ab)^2 = abab$ and $(ab)^0 = \varepsilon$.

Given a word $w = w_1 w_2$ we say that w_1 is a *prefix* and w_2 is a *suffix* of w. For instance, considering the word $w = abbbac$, ab is a prefix and c is a suffix of w. Note that ε is a prefix and suffix of every word, and every word is a prefix or a suffix of itself. A prefix w_1 of w is a *proper prefix* if $w_1 \neq \varepsilon$ and $w_1 \neq w$. If $w_2 \neq \varepsilon$ and $w_2 \neq w$ then w_2 is a *proper suffix* of w.

The *reversal of a word* $w = \sigma_1 \sigma_2 \cdots \sigma_n$ is the word written backwards, i.e. $w^R = \sigma_n \cdots \sigma_2 \sigma_1$. It is inductively defined by:

$$\begin{aligned} \varepsilon^R &= \varepsilon, \\ (\sigma w)^R &= w^R \sigma, \end{aligned}$$

for $\sigma \in \Sigma$, and $w \in \Sigma^\star$.

A *language* $\mathcal{L}$ over an alphabet Σ is a set of words over Σ, i.e. a set $\mathcal{L} \subseteq \Sigma^\star$. Its cardinality is denoted by $|\mathcal{L}|$. The *empty language*, $\emptyset$, is the language without words. The set of all words over Σ, $\Sigma^\star$, is called the *universal language*.

Beyond the usual operations on sets, as the union, intersection and complement, two operations that are specific to languages are considered: the concatenation and Kleene closure operations. Given two languages $\mathcal{L}_1 \subseteq \Sigma^\star$ and $\mathcal{L}_2 \subseteq \Sigma^\star$, its *concatenation*

$\mathcal{L}_1 \cdot \mathcal{L}_2$ or $\mathcal{L}_1\mathcal{L}_2$ is defined by:

$$\mathcal{L}_1\mathcal{L}_2 = \{w_1 w_2 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2\}.$$

Concatenation is associative as an operation on languages, i.e., for all languages $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ we have that $\mathcal{L}_1(\mathcal{L}_2\mathcal{L}_3) = (\mathcal{L}_1\mathcal{L}_2)\mathcal{L}_3$. As $\{\varepsilon\}\mathcal{L}_1 = \mathcal{L}_1\{\varepsilon\} = \mathcal{L}_1$ the set of all languages over some alphabet Σ, $2^{\Sigma^\star}$, is a monoid with respect to concatenation.

We can define the *power* $\mathcal{L}^n$ of a language $\mathcal{L} \subseteq \Sigma^\star$ inductively by:

$$\begin{aligned} \mathcal{L}^0 &= \varepsilon, \\ \mathcal{L}^{n+1} &= \mathcal{L}\mathcal{L}^n, \end{aligned}$$

for a non-negative integer n.

The *star* (or *Kleene closure*) of a language $\mathcal{L}$, denoted by $\mathcal{L}^\star$, is the set of all finite powers of $\mathcal{L}$:

$$\begin{aligned} \mathcal{L}^\star &= \mathcal{L}^0 \cup \mathcal{L}^1 \cup \mathcal{L}^2 \cup \cdots \\ &= \bigcup_{i=0}^{\infty} \mathcal{L}^i. \end{aligned}$$

Similarly we define $\mathcal{L}^+ = \bigcup_{i=1}^{\infty} \mathcal{L}^i$. For any language $\mathcal{L}$ the following results hold:

$$\begin{aligned} \mathcal{L}^\star\mathcal{L}^\star &= \mathcal{L}^\star, \\ (\mathcal{L}^\star)^\star &= \mathcal{L}^\star, \\ \mathcal{L}^\star &= \{\varepsilon\} \cup \mathcal{L}\mathcal{L}^\star = \{\varepsilon\} \cup \mathcal{L}^\star\mathcal{L}, \\ \emptyset^\star &= \varepsilon. \end{aligned}$$

The *reversal of a language* $\mathcal{L}$, denoted by $\mathcal{L}^R$, is the set of words whose reversal is on $\mathcal{L}$, $\mathcal{L}^R = \{w^R \mid w \in \mathcal{L}\}$.

Given a language $\mathcal{L} \subseteq \Sigma^\star$ and a word $w \in \Sigma^\star$, the *left-quotient* of $\mathcal{L}$ w.r.t. w is the language $w^{-1}\mathcal{L} = \{x \mid wx \in \mathcal{L}\}$.

We will consider the class of *regular languages* which can be built from $\emptyset$, $\{\varepsilon\}$ and $\{\sigma\}$ for every $\sigma \in \Sigma$, using union, concatenation and star operations.

2.2 Finite Automata

Finite automata are the main model to represent regular languages. It is one of the simplest and most fundamental computing models with applications, for example in pattern matching, in lexical analysis, in discrete event systems and in XML processing. Automata can be recognisers, i.e., they are used to recognise words of a language: the word is "processed" and, after finishing the recognising process, the automaton "decides" if the word belongs to the language or not.

In this section we will define two types of finite automata: deterministic and nondeterministic, both capable of recognising the same class of languages. We will also describe a method to convert a nondeterministic into a deterministic automaton.

2.2.1 Deterministic Finite Automata

A *deterministic finite automaton* (DFA) is a five-tuple $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ where

- Q is the finite set of states;
- Σ is the alphabet;
- δ is the transition function, $\delta : Q \times \Sigma \to Q$;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states.

The size of a DFA $A = \langle Q, \Sigma, \delta, q_0, F\rangle$, denoted by $|A|$, is its number of states, $|A| = |Q|$.

Two DFAs $A_1 = \langle Q_1, \Sigma, \delta_1, q_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, q_2, F_2 \rangle$ are isomorphic, represented by $A_1 \simeq A_2$, if there exists a bijection $f : Q_1 \to Q_2$ such that,

$$\begin{aligned} f(q_1) &= q_2, \\ f(\delta_1(q, a)) &= \delta_2(f(q), a),\ \forall q \in Q_1,\ a \in \Sigma, \\ q \in F_1 &\Leftrightarrow f(q) \in F_2,\ \forall q \in Q_1. \end{aligned}$$

A DFA can be represented by a *transition diagram*, which is a digraph with labelled arcs and nodes where:

- each node represents a state;
- a transition $\delta(p, a) = q$ is represented by an arc from the node p to the node q labelled by a;
- the initial state is signalled by an unlabelled incoming arrow with no starting node;
- final states are represented by a double circle or having an outgoing arc with no destination node.

Let us consider the DFA $E = \langle \{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_0, q_2\} \rangle$, where the transition function is defined as follows:

$$\begin{aligned} \delta(q_0, a) &= q_1, \\ \delta(q_1, a) &= q_1, \\ \delta(q_1, b) &= q_2, \\ \delta(q_2, b) &= q_0. \end{aligned}$$

The transition diagram of DFA E is represented in Figure 2.1.

A DFA is *complete* if the transition function δ is total, otherwise it is called an *incomplete* DFA. Any incomplete DFA can be completed by adding a state, called *sink state* or *dead state*, for which all missing transitions go. Figure 2.2 represents the complete version of the DFA E previously defined, where q_3 is the sink state.

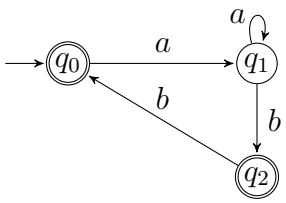


Figure 2.1: Transition diagram of an incomplete DFA.

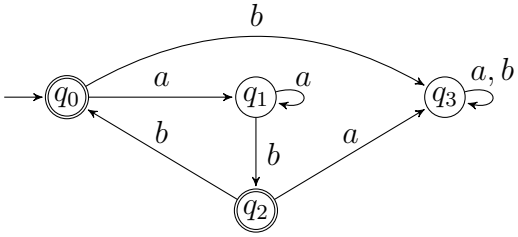


Figure 2.2: Transition diagram of a complete DFA.

For $q \in Q$ and $\sigma \in \Sigma$, if $\delta(q, \sigma)$ is defined we write $\delta(q, \sigma) \downarrow$, and $\delta(q, \sigma) \uparrow$, otherwise, and, when defining a DFA, an assignment $\delta(q, \sigma) = \uparrow$ means that the transition is undefined.

A transition labeled by $\sigma \in \Sigma$ is called a *σ-transition* and the number of σ-transitions of a DFA A is denoted by $t_\sigma(A)$. If $t_\sigma(A) = |Q|$ we say that A is *σ-complete*, and *σ-incomplete*, otherwise.

In order to process, not only symbols, but also words we need to extend the transition function δ to $Q \times \Sigma^\star \to Q$, such that

$$\begin{aligned} \delta(q, \varepsilon) &= q, \\ \delta(q, \sigma w) &= \delta(\delta(q, \sigma), w), \end{aligned}$$

where $\sigma \in \Sigma$ and $w \in \Sigma^\star$. We say that a word w is *recognised* from q if $\delta(q, w) \in F$. The words w such that $\delta(q_0, w) \in F$ are the ones *accepted* or *recognised* by the DFA.

Given a state $q \in Q$, the *right language* of q is $\mathcal{L}(A, q) = \{w \in \Sigma^\star \mid \delta(q, w) \in F\}$, and the *left language* is $\overleftarrow{\mathcal{L}}(A, q) = \{w \in \Sigma^\star \mid \delta(q_0, w) = q\}$. The *language* accepted by a DFA A is $\mathcal{L}(A) = \mathcal{L}(A, q_0)$. Two DFAs are *equivalent* if they accept the same

language.

For any states $q, q' \in Q$, if there exists a word $w \in \Sigma^\star$ such that $\delta(q, w) = q'$, then q' is a *successor* of q, and q is a *predecessor* of q'. An *accessible* state q is a state that is reachable from the initial state q_0 by some sequence of transitions, i.e., $\exists w \in \Sigma^\star\ \delta(q_0, w) = q$. A state is *useful* if it reaches a final state. Note that the dead state is an accessible state but it is not useful. If all states of a DFA are accessible then it is said to be *initially connected* (ICDFA). In this work, unless explicitly stated otherwise, all DFAs are initially connected. If all states of an ICDFA are useful it is said to be *trim*.

A DFA is *minimal* if there is no equivalent DFA with fewer states. The minimal DFA of a language has also the minimal number of transitions.

The Myhill-Nerode theorem has, among other consequences, the implication that minimal DFAs are unique up to isomorphism. Recall that an equivalence relation R on strings is right invariant if and only if for all strings u, v, and w, we have that uRv implies $uwRvw$.

Theorem 2.1 (Myhill-Nerode Theorem)**.** *Let $L \subseteq \Sigma^\star$. The following statements are equivalent:*

(a) L is regular;

(b) L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index;

(c) Let $\equiv_L$ be an equivalence relation on $\Sigma^\star$ such that $w_1 \equiv_L w_2 \Leftrightarrow \forall w_3 \in \Sigma^\star(w_1 w_3 \in L \Leftrightarrow w_2 w_3 \in L)$. The relation $\equiv_L$ is of finite index.

Proof. The proof presented follows the one in [HU79], which shows that $(a) \Rightarrow (b) \Rightarrow (c) \Rightarrow (a)$.

(a)⇒(b). Since L is a regular language, there exits a DFA $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ that recognises L. Let $\equiv_A$ be the equivalence relation on $\Sigma^\star$ such that $w_1 \equiv_A w_2$ if and

only if $\delta(q_0, w_1) = \delta(q_0, w_2)$. It is obvious that $\equiv_A$ is right invariant, since for any w_3, if $\delta(q_0, w_1) = \delta(q_0, w_2)$ then $\delta(q_0, w_1 w_3) = \delta(q_0, w_2 w_3)$. The index of $\equiv_A$ is finite, since the index is at most $|Q|$. Furthermore, L is the union of those equivalence classes that include a word w such that $\delta(q_0, w) \in F$, i.e., the equivalence classes corresponding to final states.

(b)⇒(c). We show that any equivalence relation $\equiv$ satisfying (b) is a refinement of $\equiv_L$, i.e., every equivalence class of $\equiv$ is entirely contained in some equivalence class of $\equiv_L$. Thus, the index of $\equiv_L$ cannot be greater than the index of $\equiv$ and so it is finite. Since $\equiv$ is right invariant, we have that for every pair $w_1, w_2 \in \Sigma^\star$ such that $w_1 \equiv w_2$, it must hold that $\forall w_3\ w_1 w_3 \equiv w_2 w_3$. Moreover, we have that L is the union of some equivalence classes of $\equiv$, so if $w_1 \equiv w_2$, then $w_1 \in L \Leftrightarrow w_2 \in L$. Combining these implications gives us

$$w_1 \equiv w_2 \Rightarrow \forall w_3\ w_1 w_3 \equiv w_2 w_3 \Rightarrow \forall w_3\ (w_1 w_3 \in L \Leftrightarrow w_2 w_3 \in L) \Rightarrow w_1 \equiv_L w_2.$$

Thus $\equiv$ is a refinement of $\equiv_L$.

(c)⇒(a). We must first show that $\equiv_L$ is a right invariant relation. Suppose $w_1 \equiv_L w_2$, and let w be in $\Sigma^\star$. We must prove that $w_1 w \equiv_L w_2 w$, i.e., for any w_3, $w_1 w w_3 \in L$ exactly when $w_2 w w_3 \in L$. But, since $w_1 \equiv_L w_2$, we know, by definition of $\equiv_L$, that for any w_4, $w_1 w_4 \in L$ exactly when $w_2 w_4 \in L$. Letting $w_4 = w w_3$ we conclude that $\equiv_L$ is right invariant.

To prove that if $\equiv_L$ is of finite index, then L is regular, it suffices to construct, for an arbitrary $\equiv_L$, a DFA A which recognises L. The idea that underlies the construction is to use the equivalence classes of $\equiv_L$ as states in A. First, we choose $x_1, \cdots, x_k$ as representatives for the k equivalence classes of $\equiv_L$, and then assemble the DFA $A_L = \langle Q_L, \Sigma, \delta_L, q_0, F_L \rangle$, where

- $Q_L = \{[x_1], \cdots, [x_k]\}$,
- $\delta_L([x], a) = [xa]$,

- $q_0 = [\varepsilon]$, and
- $F_L = \{[x] | x \in L\}$.

The definition of the transition function is well-defined, since $\equiv_L$ is right invariant. Had we chosen y instead of x from the equivalence class $[x]$, we would have obtained $\delta([x], a) = [ya]$. But $x \equiv_L y$, so $xz \in L$ exactly when $yz \in L$. In particular, if $z = az'$, $xaz' \in L$ exactly when $yaz' \in L$, so $xa \equiv_L ya$ and $[xa] = [ya]$. The finite automaton A accepts L, since $\delta(q_0, x) = [x]$, and thus $x \in L(A)$ if and only if $[x] \in F$. □

Theorem 2.2. *The minimal DFA accepting L is unique up to an isomorphism and is given by the DFA A_L defined in the proof of Theorem 2.1.*

Proof. The proof is a transcription of the proof of Lemma 3.10 in [HU79]. In the proof of Theorem 2.1 we saw that any DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ accepting L defines an equivalence relation that is a refinement of $\equiv_L$. Thus the number of states of M is not smaller than the number of states of A_L of Theorem 2.1. If both DFAs have the same number of states, then each of the states of M can be identified with one state of A. That is, let q be a state of M. There must be some $w \in \Sigma^\star$, such that $\delta(q_0, w) = q$. Identify q with the state $\delta_L(q_L, w)$ of A_L. If $\delta(q_0, w) = \delta(q_0, w') = q$, then, by the proof of the Theorem 2.1, w and w' are in the same equivalence class of $\equiv_L$. Thus, $\delta_L(q_L, w) = \delta_L(q_L, w') = q' \in Q_L$. □

Following the Myhill-Nerode theorem, we can say that two states q_1 and q_2, such that $\delta(q_0, w_1) = q_1$ and $\delta(q_0, w_2) = q_2$, are *equivalent* or *indistinguishable*, $q_1 \sim q_2$, if and only if $w_1 \equiv_L w_2$ or, in other words, if for all $w \in \Sigma^\star$ $(\delta(q_1, w) \in F) = (\delta(q_2, w) \in F)$. If there exists a word $w \in \Sigma^\star$ such that $(\delta(q_1, w) \in F) \neq (\delta(q_2, w) \in F)$ we say that q_1 is *distinguishable* from q_2 ($q_1 \not\sim q_2$). Formally, we define the relation $\sim$ on the states of Q by $\forall q_1, q_2 \in Q\ q_1 \sim q_2 \Leftrightarrow \forall w \in \Sigma^\star(\delta(q_1, w) \in F) \Leftrightarrow (\delta(q_2, w) \in F)$. This relation is obviously an equivalence relation. An equivalence relation R is *right invariant* on Q if and only if: $R \subseteq (Q - F)^2 \cup F^2$ and $\forall p, q \in Q, \sigma \in \Sigma, \text{if } pRq, \text{ then } \delta(p, \sigma)\ R\ \delta(q, \sigma)$.

Given a right invariant relation R, the *quotient automaton* ${}^{A}\!/_{R}$ can be constructed by

$$ {}^{A}\!/_{R} = \langle {}^{Q}\!/_{R}, \Sigma, {}^{\delta}\!/_{R}, [q_0], {}^{F}\!/_{R} \rangle, $$

where

$$ \begin{aligned} {}^{S}\!/_{R} &= \{[q] \mid q \in S\}, \text{ with } S \subseteq Q; \\ {}^{\delta}\!/_{R} &= \{([p], \sigma, [q]) \mid (p, \sigma, q) \in \delta\}. \end{aligned} $$

Note that each state of ${}^{A}\!/_{R}$ corresponds to an equivalence class of R. It is easy to see that $\mathcal{L}({}^{A}\!/_{R}) = \mathcal{L}(A)$.

Therefore, in any DFA $A = (Q, \Sigma, \delta, q_0, F)$ the equivalent states can be merged without change the language accepted by A. The resulting automaton is the DFA ${}^{A}\!/_{\sim}$, which can not be collapsed further. Thus, it is not difficult to conclude that:

Theorem 2.3. *Let A be a DFA. The DFA ${}^{A}\!/_{\sim}$ is the minimal DFA equivalent to A.*

Proof. The proof is a transcription of the proof of Lemma 3.11 in [HU79]. Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$. We must show that ${}^{A}\!/_{\sim}$ has no more states than $\equiv_L$ has equivalence classes. Suppose it had; then there are two accessible states $q, p \in Q$ such that $[q] \neq [p]$, yet there are w_1, w_2 such that $\delta(q_0, w_1) = q$ and $\delta(q_0, w_2) = p$, and $w_1 \equiv_L w_2$. We claim that $p \sim q$, for if not, then some $w \in \Sigma^\star$ distinguishes p from q. But then $w_1 w \equiv_L w_2 w$ is false, for we may let $z = \varepsilon$ and observe that exactly one of $w_1 wz$ and $w_2 wz$ is in L. But since $\equiv_L$ is right invariant, $w_1 w \equiv_L w_2 w$ is true. Hence q and p do not exist, and ${}^{A}\!/_{\sim}$ has no more states than the index of $\equiv_L$. Thus ${}^{A}\!/_{\sim}$ is the minimal DFA equivalent to A. □

A process to minimize DFAs consists in collapsing all the equivalent states. Thus, to prove that a DFA is minimal it is enough to show that for each state q of that DFA there is a word which is recognised only from q. This word distinguishes q from any other state. Using this approach we can check that the DFA in Figure 2.2 is minimal.

We can conclude that a regular language can be univocally identified, up to automata isomorphism, by the minimal DFA that accepts it.

2.2.2 Nondeterministic Finite Automata

Nondeterministic finite automata (NFA) are a generalisation of DFAs where, for a given state and an input symbol, the number of possible transitions can be greater than one. So, an NFA can be thought of as a DFA that can be in many states at once, i.e., an NFA can try any number of options in parallel. This parallelism is important because it allows for an increased generative power or higher efficiency, such as faster processing time or less (dynamic) space consumption. However, deterministic and nondeterministic automata both accept the class of regular languages, and are thus equal in generative power. In fact, any language that can be described by some NFA can also be described by a DFA.

Formally, an NFA is also a five-tuple $A = \langle Q, \Sigma, \delta, S, F \rangle$, where Q, Σ and F are defined in the same way as for DFAs, $S \subseteq Q$ is the set of initial states and the transition function is defined by $\delta : Q \times \Sigma \to 2^Q$. Sometimes, we only want to consider NFAs with a single initial state. In that case, the NFA can be denoted by a five-tuple $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where $q_0 \in Q$ is the initial state. In this work, when $S = \{q_0\}$, we use $S = q_0$. Normally, if for some $q \in Q$ and $\sigma \in \Sigma$, $\delta(q, \sigma) = \emptyset$, we omit this in the definition of δ.

As it happens for DFAs, we need to extend the transition function to words:

$$\begin{aligned} \delta : Q \times \Sigma^\star &\to 2^Q \\ \delta(q, \varepsilon) &= \{q\}, \\ \delta(q, \sigma w) &= \bigcup_{q' \in \delta(q,\sigma)} \delta(q', w), \end{aligned}$$

where $\sigma \in \Sigma$ and $w \in \Sigma^\star$.

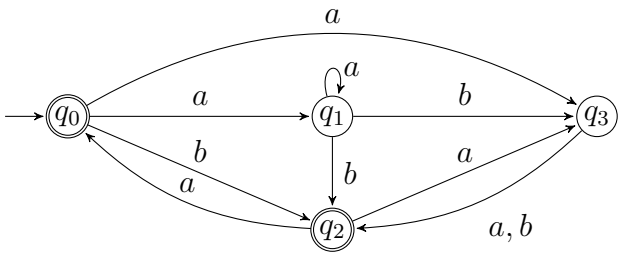


Figure 2.3: Transition diagram of an NFA.

It is also useful to extend the transition function to sets of states:

$$\delta : 2^Q \times \Sigma \to 2^Q$$
$$\delta(P, a) = \bigcup_{p \in P} \delta(p, a).$$

The size of an NFA A, $|A|$, is its number of states plus its number of transitions. The *reversal of an automaton* A is the automaton A^R, where the initial and final states are interchanged and all transitions are reversed.

Given a state $q \in Q$, the *right language* of q is $\mathcal{L}(A, q) = \{w \in \Sigma^\star \mid \delta(q, w) \cap F \neq \emptyset\}$, and the *left language* is $\overleftarrow{\mathcal{L}}(A, q) = \{w \in \Sigma^\star \mid q \in \delta(q_0, w)\}$. The language accepted by an NFA A is $\mathcal{L}(A) = \bigcup_{q \in S} \mathcal{L}(A, q)$. Two NFAs are *equivalent* if they accept the same language. If two NFAs A and B are isomorphic, we write $A \simeq B$. We can also represent NFAs using transition diagrams (Figure 2.3).

An ε-NFA, a special kind of NFA that can have transitions labelled by the empty word, is a five tuple $A_\varepsilon = \langle Q, \Sigma, \delta, q_0, F \rangle$ as defined above, but the domain of the transition function is now $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$. This extension permits that a transition can be taken without reading any input. The ε-NFA model does not add any expressive power to NFAs, but it can be useful to simplify the construction of some automata.

Contrary to what happens for DFAs, minimisation of NFAs is a hard problem (PSPACE-complete) and minimal NFAs are not unique up to isomorphism [MS72]. Nevertheless, there are several algorithms with a practical performance, that permit to reduce the size of the NFAs, even though there is no guarantee that the obtained NFA is the

smallest possible one.

Intuitively, two states are bisimilar if they can simulate each other, so the presence of bisimilar states in an NFA indicates redundancy. Thus, identifying bisimilar states permits a reduction of the NFA size.

Bisimulations are an attractive alternative for reducing the size of NFAs. A binary equivalence relation R on Q is a *bisimulation* if $\forall p, q \in Q$ and $\forall \sigma \in \Sigma$ if pRq then

- $p \in F$ if and only if $q \in F$;
- $\forall p' \in \delta(p, \sigma)\ \exists q' \in \delta(q, \sigma)$ such that $p'Rq'$.

The set of bisimulations on Q is closed under finite union. Note that the notion of bisimulation coincides to the notion of right-invariance for NFAs. R is a *left invariant* relation w.r.t. A if and only if it is a right invariant relation w.r.t. A^R.

The largest bisimulation, i.e., the union of all bisimulation relations on Q, is called *bisimilarity* ($\equiv_b$), and it can be computed in almost linear time using the Paige and Tarjan algorithm [PT87].

Given a right invariant relation R and an NFA $A = \langle Q, \Sigma, \delta, S, F \rangle$, the *quotient automaton* ${}^{A}\!/_{R}$ can be constructed by

$$ {}^{A}\!/_{R} = \langle {}^{Q}\!/_{R}, \Sigma, {}^{\delta}\!/_{R}, {}^{S}\!/_{R}, {}^{F}\!/_{R} \rangle, $$

where ${}^{\delta}\!/_{R}([q], a) = [\delta(q, a)]$. It is easy to see that $\mathcal{L}({}^{A}\!/_{R}) = \mathcal{L}(A)$.

The quotient automaton ${}^{A}\!/_{\equiv_b}$ is the minimal automaton among all quotient automata ${}^{A}\!/_{R}$, where R is a bisimulation on Q, and it is unique up to isomorphism. By abuse of language, we will call ${}^{A}\!/_{\equiv_b}$ the *bisimilarity* of automaton A. If A is a DFA, ${}^{A}\!/_{\equiv_b}$ is the minimal DFA equivalent to A, although if A is an NFA there is no guarantee that ${}^{A}\!/_{\equiv_b}$ is the minimal NFA.

2.2.3 Conversion between Nondeterministic and Deterministic Finite Automata

In many situations it is easier and more succinct to construct an NFA than a DFA that represents a given language. But, as we already mentioned, the expressive power of both models is the same, and there exists an algorithm to convert an NFA into an equivalent DFA. This conversion, usually called *determinization* and denoted by D, uses the *subset construction* which, in the worst case, constructs all the subsets of the set of states of the NFA.

Given an NFA $N = \langle Q, \Sigma, \delta, S, F\rangle$ using the subset construction we construct a DFA $D(N) = \langle Q', \Sigma, \delta', q_0', F\rangle$, such that:

$$\begin{aligned} Q' &= 2^Q; \\ \delta'(q,a) &= \delta(q,a), \text{ for } q \in Q', a \in \Sigma; \\ q_0' &= S; \\ F &= \{p \in Q' \mid p \cap F \neq \emptyset\}. \end{aligned}$$

It is not difficult to conclude that the DFA resulting from this construction has 2^n states, where $n = |Q|$. Although this method can produce a huge and possibly not initially connected DFA, in some situations all states are connected. Figure 2.4 represents the DFA obtained by the conversion of the NFA of Figure 2.3. Note that only the useful states are represented. We can prove that:

Proposition 2.4. *For any NFA $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ and any right invariant equivalence relation $\equiv$ on Q, extended in the usual way to 2^Q, $D(A/\equiv) = D(A)/\equiv$.*

Proof. We know that $D(A/\equiv) = \langle 2^{Q/\equiv}, \Sigma, \delta_\equiv, \{[q_0]\}, \{p \mid p \cap F/\equiv \neq \emptyset\}\rangle$ and $D(A)/\equiv = \langle 2^Q/\equiv, \delta'/\equiv, [\{q_0\}], \{p \mid p \cap F \neq \emptyset\}/\equiv\rangle$. We also know that for any $q \in Q$ $\delta_\equiv([q], \sigma) = \delta(q,\sigma)/\equiv$. To prove that the equality holds we only need to prove that for any $S_i = \{r_1, \ldots, r_n\}$ and $S_i/\equiv = \{[q_1], \ldots, [q_m]\}$, $\delta'(S_i, \sigma)/\equiv = \delta_\equiv(S_i/\equiv, \sigma)$, because the other

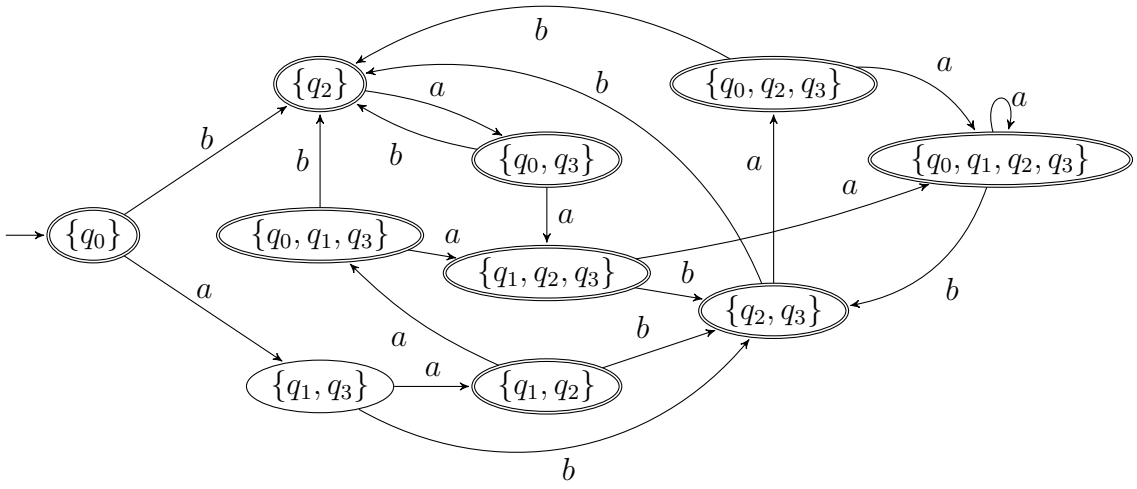


Figure 2.4: DFA obtained from the NFA represented in the Figure 2.3.

equalities are obvious. It is easy to see that,

$$\begin{aligned}
\delta'(S_i, \sigma)/_{\equiv} &= (\delta(r_1, \sigma) \cup \cdots \cup \delta(r_n, \sigma))/_{\equiv} \\
&= \delta(r_1, \sigma)/_{\equiv} \cup \cdots \cup \delta(r_n, \sigma)/_{\equiv} \\
&= \delta_{\equiv}([q_1], \sigma) \cup \cdots \cup \delta_{\equiv}([q_m], \sigma) \\
&\text{because if } r_i \equiv r_j \text{ then } \delta(r_i, \sigma) \equiv \delta(r_j, \sigma) \\
&= \delta_{\equiv}(S_i/_{\equiv}, \sigma).
\end{aligned}$$

□

2.3 Regular Expressions

Regular expressions (REs) are a more succinct and readable representation of regular languages. Let Σ be an alphabet such that $\emptyset, \varepsilon, (,), +, \cdot, \star$ do not belong to Σ. A regular expression over Σ is inductively defined by the following rules:

- the constants $\emptyset$ and ε are regular expressions;
- any symbol $a \in \Sigma$ is a regular expression;

- if α and β are regular expressions, the *disjunction* or *union* $(\alpha + \beta)$ is a regular expression;
- if α and β are regular expressions, the *concatenation* $(\alpha \cdot \beta)$ is a regular expression;
- if α is a regular expressions, the *Kleene closure* or *star* $\alpha^\star$ is also a regular expression.

Thus, given an alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of size k, we can say that the set RE of *regular expressions* α over Σ is defined by the following grammar:

$$\alpha := \emptyset \mid \varepsilon \mid \sigma \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^\star. \tag{2.1}$$

Usually we omit the non-necessary parenthesis and the concatenation operator, according to the following conventions:

- the star operator is of highest precedence;
- the concatenation operator comes next in precedence and is left-associative;
- the disjunction operator has the lowest precedence and is also left-associative.

The language recognised by a regular expression α, $\mathcal{L}(\alpha)$, is defined by the following rules, where α_1 and α_2 are arbitrary regular expressions:

- $\mathcal{L}(\emptyset) = \emptyset$;
- $\mathcal{L}(\varepsilon) = \varepsilon$;
- $\mathcal{L}(\sigma) = \{\sigma\}$, for $\sigma \in \Sigma$;
- $\mathcal{L}(\alpha_1 + \alpha_2) = \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$;
- $\mathcal{L}(\alpha_1\alpha_2) = \mathcal{L}(\alpha_1)\mathcal{L}(\alpha_2)$;
- $\mathcal{L}(\alpha_1^\star) = \mathcal{L}(\alpha_1)^\star$.

If two regular expressions α_1 and α_2 are syntactically equal, we write $\alpha_1 \equiv \alpha_2$. Two regular expressions α_1 and α_2 are equivalent, $\alpha_1 = \alpha_2$, if they accept the same language. The *length* of a regular expression α, denoted by $|\alpha|$, is the total number of symbols

in α including operators and excluding parentheses. The *alphabetic size* of α, $|\alpha|_\Sigma$, counts only the number of alphabetic symbols in α. The number of occurrences of ε in α is denoted by $|\alpha|_\varepsilon$. We represent by Σ_α the set of alphabetic symbols in α.

We define $\varepsilon : RE \to \{\emptyset, \varepsilon\}$ such that $\varepsilon(\alpha) = \varepsilon$ if $\varepsilon \in \mathcal{L}(\alpha)$ and $\varepsilon(\alpha) = \emptyset$, otherwise. We can inductively define $\varepsilon(\alpha)$ as follows:

$$\begin{aligned} \varepsilon(\sigma) = \varepsilon(\emptyset) &= \emptyset, \\ \varepsilon(\varepsilon) &= \varepsilon, \\ \varepsilon(\alpha^*) &= \varepsilon, \end{aligned} \qquad \begin{aligned} \varepsilon(\alpha_1 + \alpha_2) &= \begin{cases} \varepsilon & \text{if } (\varepsilon(\alpha_1) = \varepsilon) \text{ or } (\varepsilon(\alpha_2) = \varepsilon), \\ \emptyset & \text{otherwise,} \end{cases} \\ \varepsilon(\alpha_1\alpha_2) &= \begin{cases} \varepsilon & \text{if } (\varepsilon(\alpha_1) = \varepsilon) \text{ and } (\varepsilon(\alpha_2) = \varepsilon), \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

Given a set of regular expressions S, we define $\varepsilon(S) = \{\varepsilon(\alpha) \mid \alpha \in S\}$. The algebraic structure $(RE, +, \cdot, \emptyset, \varepsilon)$ constitutes an idempotent semiring, and with the Kleene star operator $\star$, a Kleene algebra. There are several well-known complete axiomatisations of Kleene algebras [Koz97, Sal66]. Following Kozen we can consider the axiomatic system below:

$$\alpha_1 + (\alpha_2 + \alpha_3) = (\alpha_1 + \alpha_2) + \alpha_3, \tag{2.2}$$

$$\alpha_1 + \alpha_2 = \alpha_2 + \alpha_1, \tag{2.3}$$

$$\alpha + \alpha = \alpha, \tag{2.4}$$

$$\alpha + \emptyset = \alpha, \tag{2.5}$$

$$\alpha\varepsilon = \varepsilon\alpha = \alpha, \tag{2.6}$$

$$\alpha\emptyset = \emptyset\alpha = \emptyset, \tag{2.7}$$

$$\alpha_1(\alpha_2\alpha_3) = (\alpha_1\alpha_2)\alpha_3, \tag{2.8}$$

$$\alpha_1(\alpha_2 + \alpha_3) = \alpha_1\alpha_2 + \alpha_1\alpha_3, \tag{2.9}$$

$$(\alpha_1 + \alpha_2)\alpha_3 = \alpha_1\alpha_3 + \alpha_2\alpha_3, \tag{2.10}$$

$$\varepsilon + \alpha\alpha^\star = \alpha^\star, \tag{2.11}$$

$$\varepsilon + \alpha^\star\alpha = \alpha^\star, \tag{2.12}$$

$$\alpha_1 + \alpha_2\alpha_3 \leq \alpha_3 \Rightarrow \alpha_2^\star\alpha_1 \leq \alpha_3, \tag{2.13}$$

$$\alpha_1 + \alpha_2\alpha_3 \leq \alpha_2 \Rightarrow \alpha_1\alpha_3^\star \leq \alpha_2. \tag{2.14}$$

Axioms (2.2) through (2.10) follow from the fact that the structure is an idempotent semiring. The remaining axioms refer the properties of the $\star$ operator. In (2.13) and (2.14) $\alpha \leq \beta$ means $\alpha + \beta = \beta$. It follows from the axioms that $\leq$ is a partial order, i.e., it is reflexive, transitive, and antisymmetric. From these axioms we can derive some typical theorems of Kleene algebra:

$$\alpha^\star\alpha^\star = \alpha^\star,$$

$$\alpha^{\star\star} = \alpha^\star,$$

$$(\alpha_1^\star\alpha_2)^\star\alpha_1^\star = (\alpha_1 + \alpha_2)^\star, \tag{2.15}$$

$$\alpha_1(\alpha_2\alpha_1)^\star = (\alpha_1\alpha_2)^\star\alpha_1, \tag{2.16}$$

$$\alpha^\star = (\alpha\alpha)^\star + \alpha(\alpha\alpha)^\star.$$

Equations (2.15) and (2.16) are very useful to simplify regular expressions.

We say that two regular expressions are *similar* if one can be transformed into the other using only the Axioms (2.2) through (2.7). Otherwise the regular expressions are called *dissimilar*.

We denote by ACI the set of axioms that includes the associativity (Axiom (2.2)), commutativity (Axiom (2.3)) and idempotence (Axiom (2.4)) of the disjunction operation.

Along this work we only consider regular expressions reduced by the Axioms (2.5), (2.6), (2.7), by the rule $\emptyset + \alpha = \alpha$ and without superfluous parentheses (we adopt the usual operator precedence conventions and omit outer parentheses).

The *reversal of a regular expression* $\alpha \in RE$, α^R can be inductively define by the following rules [HU79]:

$$\sigma^R = \sigma, \qquad\qquad (\alpha + \beta)^R = \alpha^R + \beta^R,$$

$$\emptyset^R = \emptyset, \qquad (\alpha\beta)^R = \beta^R\alpha^R, \qquad (2.17)$$
$$\varepsilon^R = \varepsilon, \qquad (\alpha^\star)^R = (\alpha^R)^\star.$$

Given a set of regular expressions S, we define $S^R = \{\alpha^r \mid \alpha \in S\}$.

A regular expression α is in *star normal form* (snf) [BK93] if for all subexpressions $\alpha_1^\star$ of α, $\varepsilon(\alpha_1) = \emptyset$.

2.3.1 Conversion to Finite Automata

Simulation (or conversion) methods of regular expressions into equivalent finite automata have been widely studied in the last decades. The resulting automata can be deterministic or nondeterministic. As the direct conversion to a DFA can be both time and space consuming, usually we do the conversion to an equivalent NFA and thereafter, if necessary, we convert it into an equivalent DFA using the subset construction.

The NFAs resulting from the simulation of an equivalent regular expression can have ε-transitions or not. The standard conversion with ε-transitions is the Thompson automaton ($\mathcal{A}_\mathsf{T}$) [Tho68] and the standard conversion without ε-transitions is the Glushkov (or position) automaton ($\mathcal{A}_{\text{Pos}}$) [Glu61]. In the following, we give a brief description of these two methods. We also introduce the previous automaton ($\mathcal{A}_{\text{Prev}}$), which is another conversion method without ε-transitions.

2.3.1.1 Thompson Automaton

The following algorithm due to Thompson converts any regular expression into an ε-NFA that recognises the same language. It proceeds by induction on the structure of the regular expression. The basis rules are:

- $\mathcal{A}_\mathsf{T}(\emptyset) = \langle\{q_0, f\}, \Sigma, \delta, q_0, \{f\}\rangle$, where δ is empty.

- $\mathcal{A}_{\mathsf{T}}(\varepsilon) = \langle \{q_0, f\}, \Sigma, \delta, q_0, \{f\} \rangle$, where $\delta(q_0, \varepsilon) = \{f\}$ is the only transition.
- $\mathcal{A}_{\mathsf{T}}(\sigma) = \langle \{q_0, f\}, \Sigma, \delta, q_0, \{f\} \rangle$, where $\delta(q_0, \sigma) = \{f\}$ is the only transition.

Let $\mathcal{A}_{\mathsf{T}}(\alpha_1) = \langle Q_1, \Sigma, \delta_1, q_1, F_1 \rangle$ and $\mathcal{A}_{\mathsf{T}}(\alpha_2) = \langle Q_2, \Sigma, \delta_2, q_2, F_2 \rangle$, such that $Q_1 \cap Q_2 = \emptyset$. Thus the inductive rules are:

- $\mathcal{A}_{\mathsf{T}}(\alpha_1 + \alpha_2) = \langle Q, \Sigma, \delta, q_0, \{f_0\} \rangle$ where q_0 and f_0 are new states, and

$$\begin{aligned} Q &= \{q_0, f_0\} \cup Q_1 \cup Q_2; \\ \delta(q_0, \varepsilon) &= \{q_1, q_2\}; \\ \delta(q, \varepsilon) &= \{f_0\}, \text{ for all } q \in F_1 \cup F_2; \\ \delta(q, \sigma) &= \begin{cases} \delta_1(q, \sigma) & \text{if } q \in Q_1 \\ \delta_2(q, \sigma) & \text{if } q \in Q_2 \end{cases}, \text{ for } \sigma \in \Sigma. \end{aligned}$$

- $\mathcal{A}_{\mathsf{T}}(\alpha_1 \alpha_2) = \langle Q, \Sigma, \delta, q_1, \{f_2\} \rangle$ where

$$\begin{aligned} Q &= Q_1 \cup Q_2; \\ \delta(q, \varepsilon) &= \{q_2\}, \text{ for all } q \in F_1; \\ \delta(q, \sigma) &= \begin{cases} \delta_1(q, \sigma) & \text{if } q \in Q_1 \\ \delta_2(q, \sigma) & \text{if } q \in Q_2 \end{cases}, \text{ for } \sigma \in \Sigma. \end{aligned}$$

- $\mathcal{A}_{\mathsf{T}}(\alpha_1^\star) = \langle Q, \Sigma, \delta, q_0, \{f_0\} \rangle$ where q_0 and f_0 are new states, and

$$\begin{aligned} Q &= \{q_0, f_0\} \cup Q_1; \\ \delta(q_0, \varepsilon) &= \{q_1, f_0\}; \\ \delta(q, \varepsilon) &= \{q_1, f_0\}, \text{ for all } q \in F_1; \\ \delta(q, \sigma) &= \delta_1(q, \sigma), \text{for all } q \in Q_1 \text{ and } \sigma \in \Sigma. \end{aligned}$$

By this construction, which is also depicted in Fig. 2.5, we observe that the Thompson

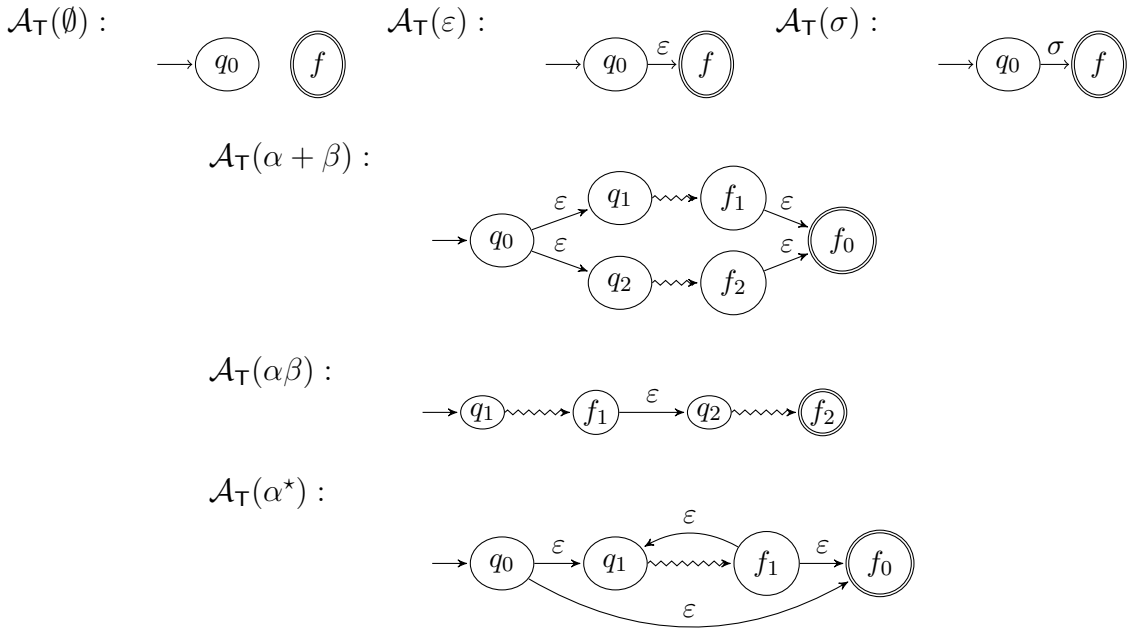


Figure 2.5: Inductive construction of $\mathcal{A}_\mathsf{T}$.

automaton of a given regular expression presents the following properties:

- there is exactly one final state;
- there are no arcs returning into the initial state;
- there are no arcs leaving the final state.

It can be computed in linear time.

2.3.1.2 Position Automaton

The position automaton, introduced by Glushkov [Glu61], permits us to convert a regular expression into an equivalent NFA without ε-transitions. The states in the position automaton ($\mathcal{A}_{\text{Pos}}$) correspond to the positions of letters in α plus an additional initial state. McNaughton & Yamada [MY60] also use the positions of a regular expression to define an automaton, however they computed directly a deterministic version of the position automaton.

We can modify the grammar (2.1) in order to define the set PO of *linear regular expressions* $\overline{\alpha}$ over Σ:

$$\overline{\alpha} := \emptyset \mid \varepsilon \mid (n, \sigma) \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^\star, \tag{2.18}$$

where n is an integer. In the following we restrict this set PO to a set of linear regular expressions $\overline{\alpha} \in \mathsf{PO}$, where for any (n, σ) occurring in $\overline{\alpha}$, $1 \leq n \leq |\alpha|_\Sigma$ and σ occurs in α in position n. We also use σ_n instead of use (n, σ), i.e., $\mathcal{L}(\overline{\alpha}) \in \overline{\Sigma}^\star$ where $\overline{\Sigma} = \{\sigma_i \mid \sigma \in \Sigma, 1 \leq i \leq |\alpha|_\Sigma\}$. For example, the marked version of the regular expression $\tau = (ab^\star + b)^\star a$ is $\overline{\tau} = (a_1 b_2^\star + b_3)^\star a_4$. The same notation is used to remove the markings, i.e., $\overline{\overline{\alpha}} = \alpha$. Let $\mathsf{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_\Sigma\}$, and $\mathsf{pos}_0(\alpha) = \mathsf{pos}(\alpha) \cup \{0\}$.

To define the $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ we consider the following sets:

$$\begin{aligned} \mathsf{First}(\alpha) &= \{i | \sigma_i w \in \mathcal{L}(\overline{\alpha})\}, \\ \mathsf{Last}(\alpha) &= \{i | w\sigma_i \in \mathcal{L}(\overline{\alpha}))\}, \\ \mathsf{Follow}(\alpha, i) &= \{j | u\sigma_i\sigma_j v \in \mathcal{L}(\overline{\alpha}))\}. \end{aligned}$$

It is necessary to extend $\mathsf{Follow}(\alpha, 0) = \mathsf{First}(\alpha)$ and define that $\mathsf{Last}_0(\alpha)$ is $\mathsf{Last}(\alpha)$ if $\varepsilon(\alpha) = \emptyset$, or $\mathsf{Last}(\alpha) \cup \{0\}$ otherwise. These sets can also be inductively defined in the structure of $\overline{\alpha}$ as follows:

$$\begin{aligned} \mathsf{First}(\varepsilon) = \mathsf{First}(\emptyset) &= \emptyset, & \mathsf{Last}(\varepsilon) = \mathsf{Last}(\emptyset) &= \emptyset, \\ \mathsf{First}(\sigma_i) &= \{i\}, & \mathsf{Last}(\sigma_i) &= \{i\}, \\ \mathsf{First}(\alpha_1\alpha_2) &= \mathsf{First}(\alpha_1) \cup \varepsilon(\alpha_1)\mathsf{First}(\alpha_2), & \mathsf{Last}(\alpha_1^\star) &= \mathsf{Last}(\alpha_1), \\ \mathsf{First}(\alpha_1 + \alpha_2) &= \mathsf{First}(\alpha_1) \cup \mathsf{First}(\alpha_2), & \mathsf{Last}(\alpha_1 + \alpha_2) &= \mathsf{Last}(\alpha_1) \cup \mathsf{Last}(\alpha_2), \\ \mathsf{First}(\alpha_1^\star) &= \mathsf{First}(\alpha_1), & \mathsf{Last}(\alpha_1\alpha_2) &= \mathsf{Last}(\alpha_2) \cup \varepsilon(\alpha_2)\mathsf{Last}(\alpha_1). \end{aligned} \tag{2.19}$$

$$\begin{aligned} \mathsf{Follow}(\varepsilon, i) = \mathsf{Follow}(\emptyset, i) &= \emptyset, \\ \mathsf{Follow}(\sigma_i, i) = \mathsf{Follow}(\sigma_i, j) &= \emptyset, \ j \neq i, \end{aligned}$$

$$\mathsf{Follow}(\alpha_1+\alpha_2,i)=\begin{cases}\mathsf{Follow}(\alpha_1,i) & \text{If } i\in\mathsf{pos}(\alpha_1),\\ \mathsf{Follow}(\alpha_2,i) & \text{If } i\in\mathsf{pos}(\alpha_2),\end{cases}$$

$$\mathsf{Follow}(\alpha_1\alpha_2,i)=\begin{cases}\mathsf{Follow}(\alpha_1,i) & \text{If } i\in\mathsf{pos}(\alpha_1)\setminus\mathsf{Last}(\alpha_1),\\ \mathsf{Follow}(\alpha_1,i)\cup\mathsf{First}(\alpha_2) & \text{If } i\in\mathsf{Last}(\alpha_1),\\ \mathsf{Follow}(\alpha_2,i) & \text{If } i\in\mathsf{pos}(\alpha_2),\end{cases} \tag{2.20}$$

$$\mathsf{Follow}(\alpha_1^\star,i)=\begin{cases}\mathsf{Follow}(\alpha_1,i) & \text{If } i\in\mathsf{pos}(\alpha_1)\setminus\mathsf{Last}(\alpha_1),\\ \mathsf{Follow}(\alpha_1,i)\cup\mathsf{First}(\alpha_1) & \text{If } i\in\mathsf{Last}(\alpha_1).\end{cases}$$

The following result relates the functions First and Last of a regular expression α and its reversal α^R.

Proposition 2.5. *For any regular expression* α, $\mathsf{First}(\alpha^R)=\mathsf{Last}(\alpha)$ *and* $\mathsf{Last}(\alpha^R)=\mathsf{First}(\alpha)$.

The *position automaton* for α is

$$\mathcal{A}_{\mathrm{Pos}}(\alpha)=\langle \mathsf{pos}_0(\alpha),\Sigma,\delta_{pos},0,\mathsf{Last}_0(\alpha)\rangle$$

where $\delta_{pos}(i,\sigma)=\{j|j\in\mathsf{Follow}(\alpha,i),\sigma=\overline{\sigma_j}\}$. Considering $\tau=(ab^\star+b)^\star a$ and $\overline{\tau}=(a_1b_2^\star+b_3)^\star a_4$, we can compute the sets:

$$\mathsf{First}(\tau)=\{1,3,4\},\qquad \mathsf{Last}(\tau)=\{4\},$$
$$\mathsf{Follow}(\tau,1)=\{1,2,3,4\},\qquad \mathsf{Follow}(\tau,2)=\{1,2,3,4\},$$
$$\mathsf{Follow}(\tau,3)=\{1,3,4\},\qquad \mathsf{Follow}(\tau,4)=\emptyset.$$

Then, we can construct the $\mathcal{A}_{\mathrm{Pos}}(\tau)$, which is represented in Figure 2.6.

From the construction of the $\mathcal{A}_{\mathrm{Pos}}$ we can infer the following properties:

- the initial state has no incoming transitions;

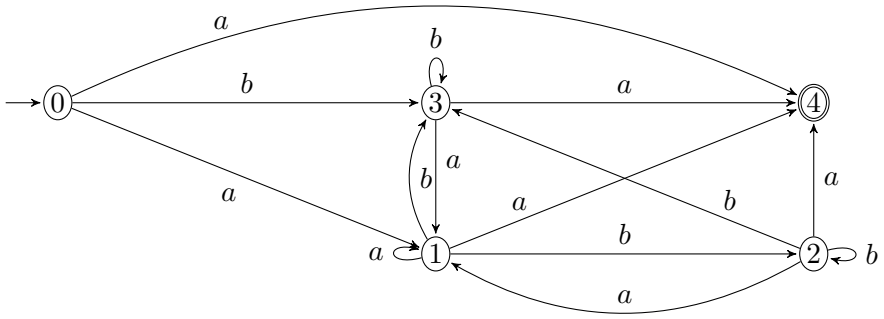


Figure 2.6: $\mathcal{A}_{\mathrm{Pos}}((ab^\star + b)^\star a)$.

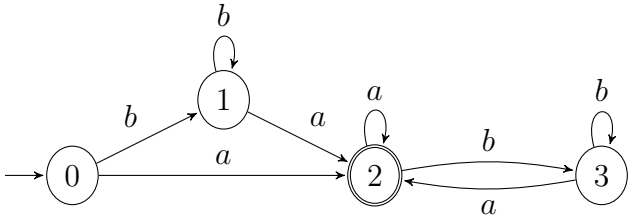


Figure 2.7: $\mathcal{A}_{MY}((ab^\star + b)^\star a)$.

- for a given state, all incoming transitions are labelled by the same symbol;
- given a regular expression α, the number of states of the resulting NFA is always $|\alpha|_\Sigma + 1$.

The position automaton can be computed in quadratic time. Brüggemann-Klein and Wood [BKW97] showed that Thompson automata can be transformed into position automata by eliminating the ε-transitions.

If we determinize the $\mathcal{A}_{\mathrm{Pos}}$ automaton, we obtain the McNaughton and Yamada DFA,

$$\mathcal{A}_{MY}(\alpha) = \mathrm{D}(\mathcal{A}_{\mathrm{Pos}}) = \langle 2^{\mathsf{pos}(\alpha)} \cup \{0\}, \Sigma, \delta_{MY}, 0, F_{MY} \rangle$$

where for $S \in 2^{\mathsf{pos}(\alpha)}$, $\delta_{MY}(S, \sigma) = \{j | j \in \mathsf{Follow}(\alpha, i), i \in S, \sigma = \overline{\sigma_j}\}$, $\delta_{MY}(0, \sigma) = \{j | j \in \mathsf{First}(\alpha), \sigma = \overline{\sigma_j}\}$, and $F_{MY} = \{S \in 2^{\mathsf{pos}(\alpha)} | S \cap \mathsf{Last}(\alpha) \neq \emptyset\} \cup \varepsilon(\alpha)\{0\}$. In Figure 2.7 is represented $\mathcal{A}_{MY}((ab^\star + b)^\star a)$.

2.3.1.3 Previous Automaton

Maintaining the idea of using the positions of the letters in a RE α, we introduce an automaton with an unique final state f, and a state for each position $i \in \mathsf{pos}(\alpha)$. To define the transition function, given a state $i \in \mathsf{pos}(\alpha)$ we compute the set of positions which precede σ_i, instead of the set of positions which follow σ_i, in $\mathcal{L}(\overline{\alpha})$ words. Thus, as we defined the set $\mathsf{Follow}(\alpha, j)$ to construct the $\mathcal{A}_{\text{Pos}}$ automaton, we define the set $\mathsf{Previous}(\alpha, j) = \{i \mid u\sigma_i\sigma_j v \in \mathcal{L}(\overline{\alpha})\}$ to construct this new automaton.

The set $\mathsf{Previous}(\alpha, j)$ can be inductively defined in the structure of α, as follows:

$$\mathsf{Previous}(\varepsilon, i) = \mathsf{Previous}(\emptyset, i) = \emptyset,$$

$$\mathsf{Previous}(\sigma_i, i) = \mathsf{Previous}(\sigma_i, j) = \emptyset,$$

$$\mathsf{Previous}(\alpha_1 + \alpha_2, i) = \begin{cases} \mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1), \\ \mathsf{Previous}(\alpha_2, i) & \text{If } i \in \mathsf{pos}(\alpha_2), \end{cases}$$

$$\mathsf{Previous}(\alpha_1\alpha_2, i) = \begin{cases} \mathsf{Previous}(\alpha_2, i) & \text{If } i \in \mathsf{pos}(\alpha_2) \setminus \mathsf{First}(\alpha_2), \\ \mathsf{Previous}(\alpha_2, i) \cup \mathsf{Last}(\alpha_1) & \text{If } i \in \mathsf{First}(\alpha_2), \\ \mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1), \end{cases}$$

$$\mathsf{Previous}(\alpha_1^\star, i) = \begin{cases} \mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1) \setminus \mathsf{First}(\alpha_1), \\ \mathsf{Previous}(\alpha_1, i) \cup \mathsf{Last}(\alpha_1) & \text{If } i \in \mathsf{First}(\alpha_1). \end{cases}$$

The *previous automaton* for α is

$$\mathcal{A}_{\text{Prev}}(\alpha) = \langle Q_{\text{Prev}}, \Sigma, \delta_{\text{Prev}}, \mathsf{First}(\alpha) \cup \varepsilon(\alpha)\{f\}, \{f\}\rangle.$$

where $Q_{\text{Prev}} = \mathsf{pos}(\alpha) \cup \{f\}$, $\delta_{\text{Prev}} = \{(i, \overline{\sigma_i}, j) \mid i \in \mathsf{Previous}(\alpha, j), j \in \mathsf{pos}(\alpha)\} \cup \{(i, \overline{\sigma_i}, f) \mid i \in \mathsf{Last}(\alpha)\}$. Note that the $\mathcal{A}_{\text{Prev}}$ automaton can have many initial states but has only one final state. Whereas each state i in the $\mathcal{A}_{\text{Pos}}$ only has in-transitions by an $\sigma = \overline{\sigma_i}$, in the $\mathcal{A}_{\text{Prev}}$ each state i only has out-transitions by an $\sigma = \overline{\sigma_i}$.

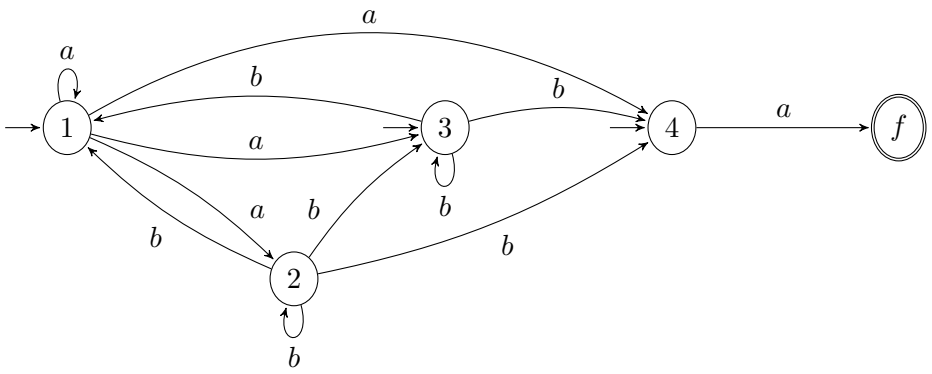


Figure 2.8: $\mathcal{A}_{\mathrm{Prev}}((ab^\star + b)^\star a)$.

Let $\tau = (ab^\star + b)^\star a$, then we can compute the sets:

$$\begin{aligned} \mathsf{First}(\tau) &= \{1,3,4\}, & \mathsf{Last}(\tau) &= \{4\}, \\ \mathsf{Previous}(\tau,1) &= \{1,2,3\}, & \mathsf{Previous}(\tau,2) &= \{1,2\}, \\ \mathsf{Previous}(\tau,3) &= \{1,2,3\}, & \mathsf{Previous}(\tau,4) &= \{1,2,3\}. \end{aligned}$$

The $\mathcal{A}_{\mathrm{Prev}}(\tau)$ is represented in Figure 2.8.

It is not difficult to see that:

$$\begin{aligned} &\forall j \in \mathsf{Previous}(\alpha, i),\ i \in \mathsf{Follow}(\alpha, j); \\ &\forall j \in \mathsf{Follow}(\alpha, i),\ i \in \mathsf{Previous}(\alpha, j). \end{aligned}$$

Moreover, we can relate these two sets considering a regular expression (α) and its reverse (α^R):

Proposition 2.6. *For any regular expression α and $i \in \mathsf{pos}(\alpha)$,*

$$\mathsf{Follow}(\alpha^R, i) = \mathsf{Previous}(\alpha, i).$$

Proof. Let us prove the result by induction on the structure of α. For $\alpha \equiv \varepsilon$, $\alpha \equiv \emptyset$ and $\alpha \equiv \sigma$ the result is obvious.

Let $\alpha \equiv \alpha_1 + \alpha_2$ then

$$\begin{aligned}\mathsf{Follow}((\alpha_1+\alpha_2)^R, i) = \mathsf{Follow}(\alpha_1^R + \alpha_2^R, i) &= \begin{cases}\mathsf{Follow}(\alpha_1^R, i) & \text{If } i \in \mathsf{pos}(\alpha_1^R)\\ \mathsf{Follow}(\alpha_2^R, i) & \text{If } i \in \mathsf{pos}(\alpha_2^R)\end{cases}\\ &= \begin{cases}\mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1)\\ \mathsf{Previous}(\alpha_2, i) & \text{If } i \in \mathsf{pos}(\alpha_2)\end{cases}\\ &= \mathsf{Previous}(\alpha_1+\alpha_2, i).\end{aligned}$$

If $\alpha \equiv \alpha_1\alpha_2$ then, as $(\alpha_1\alpha_2)^R \equiv \alpha_2^R\alpha_1^R$,

$$\begin{aligned}\mathsf{Follow}(\alpha_2^R\alpha_1^R, i) &= \begin{cases}\mathsf{Follow}(\alpha_2^R, i) & \text{If } i \in \mathsf{pos}(\alpha_2^R) \setminus \mathsf{Last}(\alpha_2^R)\\ \mathsf{Follow}(\alpha_2^R, i) \cup \mathsf{First}(\alpha_1^R) & \text{If } i \in \mathsf{Last}(\alpha_2^R)\\ \mathsf{Follow}(\alpha_1^R, i) & \text{If } i \in \mathsf{pos}(\alpha_1^R)\end{cases}\\ &= \begin{cases}\mathsf{Previous}(\alpha_2, i) & \text{If } i \in \mathsf{pos}(\alpha_2) \setminus \mathsf{First}(\alpha_2)\\ \mathsf{Previous}(\alpha_2, i) \cup \mathsf{Last}(\alpha_1) & \text{If } i \in \mathsf{First}(\alpha_2)\\ \mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1)\end{cases}\\ &= \mathsf{Previous}(\alpha_1\alpha_2, i).\end{aligned}$$

Finally if $\alpha \equiv \alpha_1^\star$ then

$$\begin{aligned}\mathsf{Follow}((\alpha_1^R)^\star, i) &= \begin{cases}\mathsf{Follow}(\alpha_1^R, i) & \text{If } i \in \mathsf{pos}(\alpha_1^R) \setminus \mathsf{Last}(\alpha_1^R)\\ \mathsf{Follow}(\alpha_1^R, i) \cup \mathsf{First}(\alpha_1^R) & \text{If } i \in \mathsf{Last}(\alpha_1^R)\end{cases}\\ &= \begin{cases}\mathsf{Previous}(\alpha_1, i) & \text{If } i \in \mathsf{pos}(\alpha_1) \setminus \mathsf{First}(\alpha_1)\\ \mathsf{Previous}(\alpha_1, i) \cup \mathsf{Last}(\alpha_1) & \text{If } i \in \mathsf{First}(\alpha_1)\end{cases}\\ &= \mathsf{Previous}(\alpha_1^\star, i).\end{aligned}$$

Thus the equality holds. □

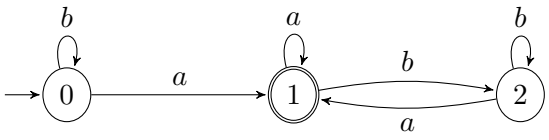


Figure 2.9: $\mathcal{A}_{dPrev}((ab^\star + b)^\star a)$.

Using this relation one can conclude that:

Proposition 2.7. *For any regular expression α, $\mathcal{A}_{\mathrm{Prev}}(\alpha) \simeq (\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$.*

Proof. The automaton $(\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$ is defined by

$$\langle \mathsf{pos}_0(\alpha^R), \Sigma, \delta^R_{pos}, \mathsf{Last}(\alpha^R) \cup \varepsilon(\alpha)\{0\}, \{0\}\rangle,$$

where $\delta^R_{pos} = \{(i, \overline{\sigma_i}, j) \mid i \in \mathsf{Follow}(\alpha^R, j)\} \cup \{(i, \overline{\sigma_i}, 0) \mid i \in \mathsf{First}(\alpha^R)\}$, and $\delta^R_{pos}(s, \sigma) = \{0\}$, if $s \in \mathsf{First}(\alpha^R)$. Let $\varphi(i) = i$ for $i \in \mathsf{pos}(\alpha)$ and $\varphi(0) = f$. It is obvious that φ is an isomorphism between $(\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$ and $\mathcal{A}_{\mathrm{Prev}}$, because $\mathsf{Last}(\alpha^R) = \mathsf{First}(\alpha)$, $\mathsf{First}(\alpha^R) = \mathsf{Last}(\alpha)$ and Proposition 2.6. □

If we determinize $\mathcal{A}_{\mathrm{Prev}}$, we obtain

$$\mathcal{A}_{dPrev}(\alpha) = \langle Q_{dPrev}, \Sigma, \delta_{dPrev}, \mathsf{First}(\alpha) \cup \varepsilon(\alpha)\{f\}, F_{dPrev}\rangle$$

where $Q_{dPrev} = 2^{\mathsf{pos}(\alpha)\cup\{f\}}$, $\delta_{dPrev}(P, \sigma) = \{j \mid i \in \mathsf{Previous}(\alpha, j), i \in P, j \in \mathsf{pos}(\alpha), \sigma = \overline{\sigma_i}\} \cup \{f\}$, if $\mathsf{Last}(\alpha) \cap P \neq \emptyset$ or $\delta_{dPrev}(P, \sigma) = \{j \mid i \in \mathsf{Previous}(\alpha, j), i \in P, j \in \mathsf{pos}(\alpha), \sigma = \overline{\sigma_i}\}$, otherwise; and $F_{dPrev} = \{S \in Q_{dPrev} \mid f \in S\}$.

2.3.2 Derivatives

The derivative of a regular expression α with respect to a symbol $\sigma \in \Sigma$ [Brz64] is a regular expression, denoted by $\sigma^{-1}\alpha$, and can be defined recursively on the structure of α as follows:

$$
\begin{array}{rcl}
\sigma^{-1}\emptyset & = & \sigma^{-1}\varepsilon = \emptyset, \\
\sigma^{-1}\sigma' & = & \begin{cases} \varepsilon & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases}
\end{array}
\qquad
\begin{array}{rcl}
\sigma^{-1}(\alpha_1 + \alpha_2) & = & \sigma^{-1}\alpha_1 + \sigma^{-1}\alpha_2, \\
\sigma^{-1}(\alpha^\star) & = & (\sigma^{-1}\alpha)\alpha^\star, \\
\sigma^{-1}(\alpha_1\alpha_2) & = & \begin{cases} (\sigma^{-1}\alpha_1)\alpha_2 & \text{if } \varepsilon(\alpha_1) \neq \varepsilon, \\ (\sigma^{-1}\alpha_1)\alpha_2 + \sigma^{-1}\alpha_2 & \text{otherwise.} \end{cases}
\end{array}
\tag{2.21}
$$

This notion can be naturally extended to words: $\varepsilon^{-1}\alpha = \alpha$, and $(\sigma w)^{-1}\alpha = w^{-1}(\sigma^{-1}\alpha)$, where $w \in \Sigma^\star$; or, more generally, $(ps)^{-1}\alpha = s^{-1}(p^{-1}\alpha)$ for every factorisation $w = ps,\ p, s \in \Sigma^\star$.

Brzozowski proved that for any $w \in \Sigma^\star$, $L(w^{-1}\alpha) = w^{-1}L(\alpha)$. Let $\mathcal{D}(\alpha)$ be the quotient of the set of all derivatives of a regular expression α w.r.t. a word, modulo the ACI-equivalence relation. Brzozowski also proved that the set $\mathcal{D}(\alpha)$ is finite. Using this result it is possible to define the *Brzozowski's automaton*:

$$A_{\mathcal{B}}(\alpha) = \langle \mathcal{D}(\alpha), \Sigma, \delta, [\alpha], F \rangle,$$

where $F = \{[d] \in \mathcal{D}(\alpha) \mid \varepsilon(d) = \varepsilon\}$, and $\delta([q], \sigma) = [\sigma^{-1}q]$, for all $[q] \in \mathcal{D}(\alpha), \sigma \in \Sigma$. From what has been said above and from the left-quotient definition it follows that this automaton recognises $L(\alpha)$.

2.3.2.1 c-Continuations

Berry & Sethi [BS86] characterised the Brzozowski's derivatives of a linear regular expression. Champarnaud & Ziadi [CZ02] extended their study introducing the notion of canonical derivative of a regular expression, in order to compute a canonical representative of the set of the ACI-similar derivatives of a linear regular expression computed by Berry and Sethi.

Given a regular expression α and a symbol σ, the *c-derivative* of α w.r.t. σ, denoted by $d_\sigma\alpha$, is defined by

$$
\begin{array}{rcl}
d_\sigma(\emptyset) & = & d_\sigma(\varepsilon) = \emptyset, \\
d_\sigma(\sigma') & = & \begin{cases} \varepsilon & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} \\
d_\sigma(\alpha^\star) & = & d_\sigma(\alpha)\alpha^\star,
\end{array}
\qquad
\begin{array}{rcl}
d_\sigma(\alpha+\beta) & = & \begin{cases} d_\sigma(\alpha), & \text{if } d_\sigma(\alpha) \neq \emptyset, \\ \varepsilon(\alpha)d_\sigma(\beta), & \text{otherwise,} \end{cases} \\
d_\sigma(\alpha\beta) & = & \begin{cases} d_\sigma(\alpha)\beta, & \text{if } d_\sigma(\alpha) \neq \emptyset, \\ d_\sigma(\beta), & \text{otherwise.} \end{cases}
\end{array}
\tag{2.22}
$$

The extension to a word follows the equations: $d_\varepsilon(\alpha) = \alpha$ and $d_{\sigma w}(\alpha) = d_w(d_\sigma(\alpha))$.

If α is a linear regular expression, for every symbol $\sigma \in \overline{\Sigma}$ and every word $w \in \overline{\Sigma}^\star$, $d_{w\sigma}(\alpha)$ is either $\emptyset$ or unique modulo ACI [BS86]. If $d_{w\sigma}(\alpha)$ is different from $\emptyset$, it is named *c-continuation* of α w.r.t. $\sigma \in \overline{\Sigma}$, denoted by $c_\sigma(\alpha)$, and it is defined as follows:

$$
\begin{array}{rcl}
c_\sigma(\sigma') & = & \begin{cases} \varepsilon & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} \\
c_\sigma(\alpha^\star) & = & c_\sigma(\alpha)\alpha^\star,
\end{array}
\qquad
\begin{array}{rcl}
c_\sigma(\alpha+\beta) & = & \begin{cases} c_\sigma(\alpha), & \text{if } c_\sigma(\alpha)\downarrow, \\ c_\sigma(\beta), & \text{otherwise,} \end{cases} \\
c_\sigma(\alpha\beta) & = & \begin{cases} c_\sigma(\alpha)\beta, & \text{if } c_\sigma(\alpha)\downarrow, \\ c_\sigma(\beta), & \text{otherwise,} \end{cases}
\end{array}
\tag{2.23}
$$

where $c_\sigma(\alpha)\downarrow$ means that $c_\sigma(\alpha)$ is defined. Let $c_0(\alpha) = d_\varepsilon(\alpha) = \alpha$. This means that we can associate to each position $i \in \mathsf{pos}_0(\alpha)$, a unique c-continuation. For example, given $\overline{\tau} = (a_1 b_2^\star + b_3)^\star a_4$ we have $c_{a_1}(\overline{\tau}) = b_2^\star\overline{\tau}$, $c_{b_2}(\overline{\tau}) = b_2^\star\overline{\tau}$, $c_{b_3}(\overline{\tau}) = \overline{\tau}$, and $c_{a_4}(\overline{\tau}) = \varepsilon$. The *c-continuation automaton* for α is

$$\mathcal{A}_c(\alpha) = \langle Q_c, \Sigma, \delta_c, q_0, F_c \rangle$$

where $Q_c = \{q_0\} \cup \{(i, c_{\sigma_i}(\overline{\alpha})) \mid i \in \mathsf{pos}(\alpha)\}$, $q_0 = (0, c_0(\overline{\alpha}))$, $F_c = \{(i, c_{\sigma_i}(\overline{\alpha})) \mid \varepsilon(c_{\sigma_i}(\overline{\alpha})) = \varepsilon\}$, $\delta_c = \{((i, c_{\sigma_i}(\overline{\alpha})), b, (j, c_{\sigma_j}(\overline{\alpha}))) \mid \overline{\sigma_j} = b \wedge d_{\sigma_j}(c_{\sigma_i}(\overline{\alpha})) \neq \emptyset\}$. The $\mathcal{A}_c(\tau)$ is represented in Figure 2.10.

Note that if we ignore the c-continuations in the label of each state, we obtain the position automaton.

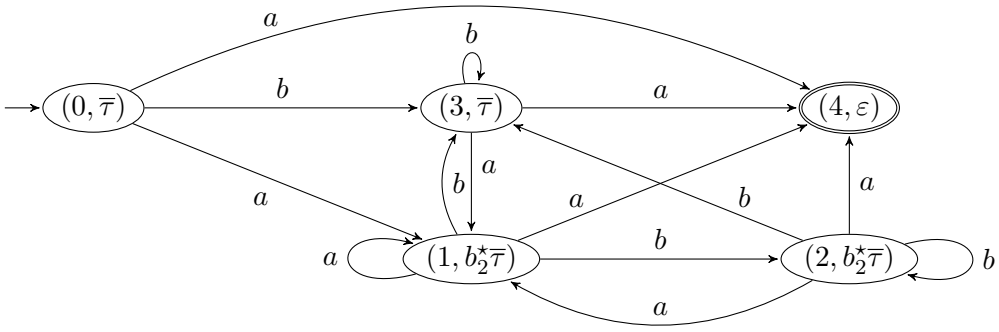


Figure 2.10: $\mathcal{A}_c((a_1b_2^\star + b_3)^\star a_4)$.

Proposition 2.8 (Champarnaud & Ziadi)**.** $\forall \alpha \in RE$, $\mathcal{A}_{\mathrm{Pos}}(\alpha) \simeq \mathcal{A}_c(\alpha)$.

The following proposition establishes a relation between the sets First, Follow and Last and the c-continuations.

Proposition 2.9 (Champarnaud & Ziadi)**.** *For all $\alpha \in RE$, the following equalities hold*

$$\mathsf{First}(\alpha) = \{\sigma \in \overline{\Sigma} | d_a(\overline{\alpha}) \neq \emptyset\},$$
$$\mathsf{Last}(\alpha) = \{\sigma \in \overline{\Sigma} | \varepsilon(c_\sigma(\overline{\alpha})) \neq \emptyset\},$$
$$\mathsf{Follow}(\alpha, i) = \{\sigma_j \in \overline{\Sigma} | d_{\sigma_j}(c_{\sigma_i}(\overline{\alpha})) \neq \emptyset\}.$$

The c-continuation automaton can be computed in quadratic time.

2.3.2.2 Partial Derivatives

Partial derivatives, presented by Antimirov [Ant96], are a generalisation to the non-deterministic case of the notion of derivative. For a RE α and a symbol $\sigma \in \Sigma$, the set of partial derivatives of α w.r.t. σ can be inductively defined as follows:

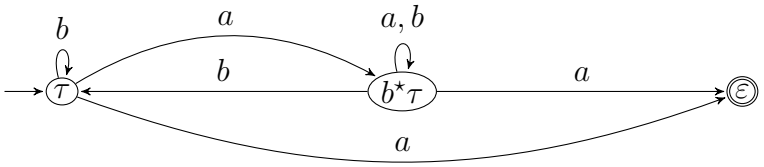


Figure 2.11: $\mathcal{A}_{\mathrm{PD}}((ab^\star + b)^\star a)$.

$$\begin{array}{rclcrcl} \partial_\sigma(\emptyset) & = & \partial_\sigma(\varepsilon) = \emptyset, & & \partial_\sigma(\alpha+\beta) & = & \partial_\sigma(\alpha) \cup \partial_\sigma(\beta), \\ \partial_\sigma(\sigma') & = & \begin{cases} \{\varepsilon\} & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} & & \partial_\sigma(\alpha\beta) & = & \partial_\sigma(\alpha)\beta \cup \varepsilon(\alpha)\partial_\sigma(\beta), \\ & & & & \partial_\sigma(\alpha^\star) & = & \partial_\sigma(\alpha)\alpha^\star, \end{array} \tag{2.24}$$

where for any $S \subseteq RE$, $\beta \in RE$, $S\emptyset = \emptyset S = \emptyset$, $S\varepsilon = \varepsilon S = S$, $S\beta = \{\alpha\beta | \alpha \in S\}$ and $\beta S = \{\beta\alpha | \alpha \in S\}$ if $\beta \neq \emptyset$, and $\beta \neq \varepsilon$. The definition of partial derivative can be extended to sets of regular expressions, words, and languages. Given $\alpha \in RE$ and $\sigma \in \Sigma$, $\partial_\sigma(S) = \bigcup_{\alpha\in S} \partial_\sigma(\alpha)$ for $S \subseteq RE$, $\partial_\varepsilon(\alpha) = \{\alpha\}$ and $\partial_{w\sigma}(\alpha) = \partial_\sigma(\partial_w(\alpha))$, for any $w \in \Sigma^\star, \sigma \in \Sigma$, and $\partial_L(\alpha) = \bigcup_{w\in L} \partial_w(\alpha)$ for $L \subseteq \Sigma^\star$. We know that $\bigcup_{\tau\in\partial_w(\alpha)} \mathcal{L}(\tau) = w^{-1}\mathcal{L}(\alpha)$ and also that $\sum \partial_w(\alpha) = w^{-1}\alpha$, where for $S = \{\alpha_1 \cdots \alpha_n\}$, $\sum S = \alpha_1 + \cdots + \alpha_n$. The set of all partial derivatives of α w.r.t. words is denoted by $\mathsf{PD}(\alpha) = \bigcup_{w\in\Sigma^\star} \partial_w(\alpha)$. The set $\mathsf{PD}(\alpha)$ is always finite [Ant96]. We also define the set $\mathsf{PD}^+(\alpha) = \bigcup_{w\in\Sigma^+} \partial_w(\alpha)$. Note that $\mathsf{PD}(\alpha) = \mathsf{PD}^+(\alpha) \cup \{\alpha\}$.

The partial derivative automaton of a regular expression was introduced independently by Mirkin [Mir66] and Antimirov [Ant96]. Champarnaud & Ziadi [CZ01] proved that the two formulations are equivalent. It is defined by

$$\mathcal{A}_{\mathrm{PD}}(\alpha) = \langle \mathsf{PD}(\alpha), \Sigma, \delta_{pd}, \alpha, F_{pd} \rangle,$$

where $\delta_{pd} = \{(\tau, \sigma, \tau') \mid \tau \in \mathsf{PD}(\alpha) \text{ and } \tau' \in \partial_\sigma(\tau)\}$ and $F_{pd} = \{\tau \in \mathsf{PD}(\alpha) \mid \varepsilon(\tau) = \varepsilon\}$. Considering $\tau = (ab^\star + b)^\star a$, Figure 2.11 shows $\mathcal{A}_{\mathrm{PD}}(\tau)$.

Given the c-continuation automaton $\mathcal{A}_c(\alpha)$, let $\equiv_c$ be the right invariant equivalence

relation on Q_c defined by $(i, c_{\sigma_i}(\overline{\alpha})) \equiv_c (j, c_{\sigma_j}(\overline{\alpha}))$ if $\overline{c_{\sigma_i}(\overline{\alpha})} \equiv \overline{c_{\sigma_j}(\overline{\alpha})}$. The fact that the $\mathcal{A}_{\mathrm{PD}}$ is isomorphic to the resulting quotient automaton, follows from the following proposition.

Proposition 2.10 (Champarnaud & Ziadi)**.** $\forall \alpha \in RE$, $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {\mathcal{A}_c(\alpha)}/{\equiv_c}$.

For our running example, we have $(0, c_\varepsilon) \equiv_c (3, c_{b_3})$ and $(1, c_{a_1}) \equiv_c (2, c_{b_2})$. In Figure 2.11, we can see the merged states, and that the corresponding REs are unmarked.

The partial derivative automaton can be computed in quadratic time.

2.3.2.3 Related Constructions

In [IY03a], Ilie & Yu proposed a new method to construct NFAs from regular expressions. First, the authors construct an NFA with ε-transitions – $A_f^\varepsilon(\alpha)$. Then they use an ε-elimination method to build the *follow automaton* – $A_f(\alpha)$. The authors also proved that the follow automaton is a quotient of the position automaton.

Proposition 2.11 (Ilie & Yu)**.** *For all* $\alpha \in RE$, $A_f(\alpha) \simeq {\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_f}$, *where* $i \equiv_f j$ *iff both* i, j *or none belong to* $\mathsf{Last}(\alpha)$ *and* $\mathsf{Follow}(\alpha, i) = \mathsf{Follow}(\alpha, j)$.

Recently, Garcia et al. also proposed a new method to construct NFAs from regular expressions [GLRA11]. The size of the resulting automaton is bounded above by the size of the smallest automata obtained by the follow and partial derivatives methods.

Let the equivalence $\equiv_\vee$ be the join of the relations $\equiv_c$ and $\equiv_f$, where the join relation between two equivalence relations E_1 and E_2 is the smallest equivalence relation that contains E_1 and E_2. The Garcia et al. automaton, $\mathcal{A}_u(\alpha)$, is a quotient of the position automaton by that relation – $\mathcal{A}_u(\alpha) \simeq {\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_\vee}$.

Both automata can be computed in quadratic time.

If we consider any regular expression α in snf, the size of $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is equal to the size of $\mathcal{A}_u(\alpha)$, and not greater than the size of $A_f(\alpha)$.

Chapter 3

Descriptional Complexity

Over the last two decades and motivated by the increasing number of new practical applications, the descriptional complexity of formal languages has become a major topic of research. Given a complexity measure and a model of computation, the descriptional complexity of a language w.r.t. that measure and model is the size of its smallest representation.

Given a formal language, it can be represented by several models, e.g. nondeterministic finite automata, deterministic finite automata, regular expressions, etc.. All these models are equally powerful, in the sense that they represent exactly the same language. In the same way, the proofs of the same mathematical theorem can differ greatly in length and complexity, but all of them have the same purpose. So, it is important to study computational models not only with respect to their expressive power, but also taking into account its size according to a specific measure. A typical example is the exponential trade-off between the number of states of a nondeterministic and a deterministic automaton for the same regular language.

The descriptional complexity of formal languages is concerned with questions like:

- How efficiently can a model describe a formal language w.r.t. other models?

- What is the cost of a conversion from one model to another? What are the upper and lower bounds of such costs and can they be attained?

The same questions can be posed when applying operations on models. It is important to know how the size varies when several such models are combined, since this has a direct influence on the amount of resources required by the applications. In general, having succinct objects will improve our software, which may then become smaller, more efficient and reliable.

Regular languages, despite their limited expressive power, have many applications in almost all Computer Science areas. In general, regular languages properties are decidable and the computational complexity of associated problems is known, which is also attractive for this class of languages, mainly, when compared with the undecidability world of context-free languages. However, many descriptional complexity aspects of regular languages are still open problems, and are directly related with a more refined analysis of the performance of a particular algorithm. So, it is essential that the structural properties of regular language representations are further researched.

The descriptional complexity aspects can be study in two different approaches: in the worst case [Yu01] and in the average case [Nic99]. Although its evident practical importance, there is still very few research on average-case complexity, contrary to what happens for the worst-case complexity for which a lot of results are known.

In Section 3.1, we review the state and transition complexities of individual regularity preserving language operations like Boolean operations, concatenation, star and reversal, considering the worst-case analysis.

In Section 3.2 we introduce a few results known on average-case descriptional complexity. We also present some analytic tools, which will be used in Section 5.5 to analyse the asymptotic average size of some conversions between regular expressions and NFAs. For a more extensive study on analytic combinatorics we refer the reader to Flajolet & Sedgewick [FS08].

3.1 Operational State and Transition Complexities on Regular Languages

Concerning the DFAs, there are many ways to measure their size: the number of states, the number of transitions or the sum of the number of states and transitions. In the case of a complete DFA the number of transitions is totally determined by the number of states and the alphabet size, i.e., the number of transitions is equal to the product of the alphabet size by the number of states. Therefore, the number of states is the key measure on the size of a complete DFA.

As we have already seen, a regular language is accepted by infinitely many different DFAs. The usual complexity measure is the number of states of the complete minimal DFA that accepts L, which is called *state complexity* of the regular language L, and it is denoted by $\mathrm{sc}(L)$ [Yu05, Yu06, BHK09, HK09a, YG11]. This is the most studied descriptional measure for regular languages. First results concerning the state complexity of regular languages and their operations date from the 1960's and 1970's [Mas70, Moo71, Lup66]. In 1994, the work [YZS94] on the state complexity of the languages resulting from basic operations (Boolean, concatenation, star and reversal), revived the interest of the community on this topic. The proliferous research gave origin to a few hundred of papers which were surveyed, for example, in [Yu97, Yu01, Yu05, HK09a, HK11].

In many applications where large alphabets need to be considered or, in general, when very sparse transition functions take place, partial transition functions are very convenient. Examples include lexical analysers, discrete event systems, or any application that uses dictionaries where compact automaton representations are essential [ORT09, DW11, CL06]. Thus, it makes sense to study complexity measures of regular languages based on non necessarily complete DFAs. The *incomplete state complexity* of a regular language L ($\mathrm{isc}(L)$) is the number of states of the minimal not necessarily complete DFA that accepts L. Note that $\mathrm{isc}(L)$ differs at most by 1 from $\mathrm{sc}(L)$ ($\mathrm{isc}(L) \in \{\mathrm{sc}(L) - 1, \mathrm{sc}(L)\}$).

Table 3.1: State complexity, nondeterministic state and transition operational complexity of basic regularity preserving operations on regular languages.

Operation	sc	nsc	ntc
$L_1 \cup L_2$	mn	$m+n+1$	$\mathrm{ntc}(L_1)+\mathrm{ntc}(L_2)+s(L_1)+s(L_2)$
$L_1 \cap L_2$	mn	mn	$\sum_{\sigma\in\Sigma} \mathrm{ntc}_\sigma(L_1)\,\mathrm{ntc}_\sigma(L_2)$
L^C	n	2^n	$\lvert\Sigma\rvert 2^{\mathrm{ntc}(L)+1}$
			$2^{\frac{\mathrm{ntc}(L)}{2}-2}-1$
L_1L_2	$m2^n - f_1 2^{n-1}$	$m+n$	$\mathrm{ntc}(L_1)+\mathrm{ntc}(L_2)+f_{in}(L_1)$
$L^\star$	$2^{m-1}+2^{m-l-1}$	$m+1$	$\mathrm{ntc}(L)+f_{in}(L)$
L^R	2^m	$m+1$	$\mathrm{ntc}(L)+f(L)$

Contrary to what happens for complete DFAs, in non necessarily complete DFAs the study of the number of transitions is relevant, because it is not determined by the number of states. The *incomplete transition complexity*, $\mathrm{itc}(L)$, of a regular language L is the minimal number of transitions over all non necessarily complete DFAs that accept L. Given a $\sigma \in \Sigma$, the *σ-transition complexity* of L, $\mathrm{itc}_\sigma(L)$, is the minimal number of σ-transitions of any DFA recognising L. In [GSY11, Lemma 2.1] it was proved that the minimal DFA accepting L has the minimal number of σ-transitions, for every $\sigma \in \Sigma$. From this it follows that $\mathrm{itc}(L) = \sum_{\sigma\in\Sigma} \mathrm{itc}_\sigma(L)$. The incomplete transition complexity has not been much studied. Recently, Gao et al. [GSY11] study this measure for the boolean operations for the first time. In this work (Section 4.1) we extend their analysis to the concatenation, the Kleene star and the reversal operations.

The *nondeterministic state complexity* of a regular language L, $\mathrm{nsc}(L)$, is the number of states of a minimal NFA that accepts L; and similarly the *nondeterministic transition complexity* of a regular language L, $\mathrm{ntc}(L)$, is the number of transitions of a minimal NFA that accepts L. We can refine this last measure using the *σ-nondeterministic transition complexity* of L, $\mathrm{ntc}_\sigma(L)$, which is the minimal number of σ-transitions of any transition-minimal NFA recognising L. Note that $\mathrm{ntc}(L) = \sum_{\sigma\in\Sigma} \mathrm{ntc}_\sigma(L)$. Both measures, $\mathrm{nsc}(L)$ and $\mathrm{ntc}(L)$, were thoroughly studied [DS07, Sal07, HK03, HK09b, HK09a].

The *complexity of an operation* on regular languages is the (worst-case) complexity of

a language resulting from the operation, considered as a function of the complexities of the operands. Following the formulation from Holzer & Kutrib [HK09b], given a binary operation $\diamond$ on languages that preserves regularity, the $\diamond$-*language operation state complexity problem* for DFAs (NFAs) is defined as follows:

- Given an n-state DFA (NFA) A_1 and an m-state DFA (NFA) A_2.
- How many states are sufficient and necessary, in the worst case, to accept the language $\mathcal{L}(A_1) \diamond \mathcal{L}(A_2)$ by a DFA (NFA).

This formulation can be generalised for other operation arities, complexity measures, automata and classes of languages.

Usually an *upper bound* is obtained by providing an algorithm which, given representations of the operands (e.g. DFAs), constructs a model (e.g. DFA) that accepts the language resulting from the referred operation. The number of states or transitions of the resulting representation (e.g. DFA) is an upper bound for the state or the transition complexity of the operation, respectively. To prove that an upper bound is *tight*, for each operand we give a family of languages (parametrised by the complexity measures), called *witnesses*, such that the complexity of the resulting language achieves that upper bound.

Consider L_1 and L_2 such that $\mathrm{sc}(L_1) = m$ ($\mathrm{nsc}(L_1) = m$) and $\mathrm{sc}(L_2) = n$ ($\mathrm{nsc}(L_2) = n$). Table 3.1 summarises the results for state complexity, nondeterministic state and nondeterministic transition complexity of basic regularity preserving operations on regular languages. The parameter $s(L)$ is the minimal number of transitions leaving the initial state of any transition-minimal NFA accepting L, $f_i(L_i)$ is the minimal number of final states of any transition-minimal NFA accepting L_i, and $f_{in}(L)$ is the number of transitions entering the final states of any transition-minimal NFA accepting L.

Yu et al. [YZS94] studied the state complexity of concatenation, star, reversal, union, and intersection. However, some of these results had already been presented by

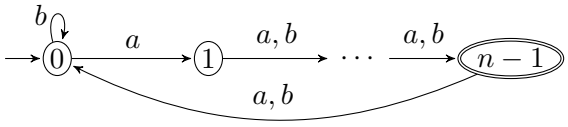


Figure 3.1: Witness DFA for the state complexity of the star for $m > 2$.

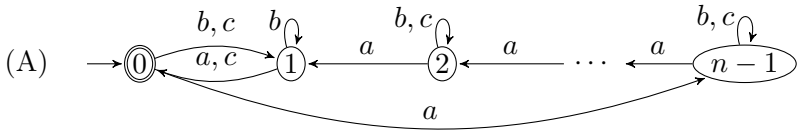


Figure 3.2: Witness DFA for the state complexity of the reversal.

Maslov [Mas70] and Rabin & Scott [RS59] earlier. The families of languages which witness the tightness for intersection are $\{x \in \{a,b\} \mid \#_a(x) = 0 \pmod m\}$ and $\{x \in \{a,b\} \mid \#_b(x) = 0 \pmod n\}$. Their complements are witnesses for union. For concatenation, the authors also present binary languages tight bound witnesses for $m \geq 1$, $n = 1$ and $m = 1$, $n \geq 2$, but ternary languages tight bound witnesses for $m > 1$, $n \geq 2$. Considering the star operation, the upper bound is achieved for the languages $\{w \in \{a,b\}^\star \mid \#_a(w) \text{ is odd}\}$, if $m = 2$; if $m > 2$ it is achieved for the family of binary languages accepted by the DFAs presented in Figure 3.1. The authors also proved the tightness of the bound of the reversal operation for a family of ternary languages (see Figure 3.2). A family of binary languages for which the upper bound for reversal is tight was given by Jirásková & Sěbej [Seb10, JS11]. Complementation for DFAs is trivial and it is obvious that the state complexity of the complement is the same one of the original language.

Concerning the unary languages, the state complexity for several operations is much lower than what is predicted by the results for the general case. The main state complexity results for this class of languages are presented in Table 3.2. For union and intersection operations, the state complexity coincides asymptotically with the one for general regular languages. Yu [Yu01] showed that the bound for these operations was tight if m and n are coprimes and the witness languages are $(a^m)^\star$ and $(a^n)^\star$. In [YZS94] is also shown the tightness of the upper bound for the concatenation,

Table 3.2: State complexity and nondeterministic state complexity of basic regularity preserving operations on unary regular languages. The symbol $\sim$ means that the complexities are asymptotically equal to the given values. The upper bounds of state complexity for union, intersection and concatenation are exact if m and n are coprimes.

Operation	sc	nsc
$L_1 \cup L_2$	$\sim mn$	$m+n+1$
$L_1 \cap L_2$	$\sim mn$	mn
L^C	m	$e^{\theta(\sqrt{n \ln n})}$
$L_1 L_2$	$\sim mn$	$[m+n-1, m+n]$ if $m, n > 1$
$L^\star$	$(m-1)^2+1$ if $m > 1$	$m+1$ if $m > 2$
L^R	m	m

again just if m and n are coprimes. The languages $(a^m)^\star a^{m-1}$ and $(a^n)^\star a^{n-1}$ are the witnesses of tightness. In the same paper the authors proved that the upper bound for the star operation is tight and the witnesses of tightness are $(aa)^\star$ if $m = 2$, and $(a^m)^\star a^{m-1}$ if $m > 2$. The state complexity of the reversal of a unary language L is trivially equal to the state complexity of L.

The state complexity of basic operations on NFAs was first studied by Holzer & Kutrib [HK03], and also by Ellul [Ell02]. For the union operation, the idea is to construct an NFA that starts with a new initial state and guesses which of the operands should be simulate. Considering the families $(a^m)^\star$ and $(b^n)^\star$ over a binary alphabet we observe that the upper bound $(m+n+1)$ is tight. For intersection, the operands have to be simulated in parallel, thus a product construction is needed. The languages $\{w \in \{a,b\}^\star \mid \#_a(w) \equiv 0 \pmod m\}$ and $\{w \in \{a,b\}^\star \mid \#_b(w) \equiv 0 \pmod n\}$, where m and n are the respective nondeterministic state complexity, witness the tightness of the bound for this operation. Since the complementation operation on DFAs neither increases nor decreases the number of states of the referred DFA, the upper bound for the nondeterministic state complexity of this operation on NFAs is obtained by determinization. Jirásková [Jir05] proved that this upper bound is tight even for binary languages. The same families considered for union operation, $(a^m)^\star$ and $(b^n)^\star$, permit that the upper bound for the concatenation is reached. For the star operation, the upper bound is achieved for the languages $\{w \in \{a,b\}^\star \mid \#_a(w) \equiv n-1 \pmod n\}$, for

any $n > 2$. The languages $a^k(a^{k+1})^\star(b^\star + c^\star)$ for $k \geq 1$, presented by Holzer & Kutrib, serve as example for the fact that the upper bound for reversal operation is reached. However, the referred bound is also tight for a family of binary languages [Jir05]. The comparison between the upper bounds of this operation for DFAs and NFAs shows how powerful the nondeterminism concept can be.

The nondeterministic state complexity of basic operations on unary regular languages was studied by Holzer & Kutrib [HK02], and also by Ellul [Ell02]. For union and intersection, the upper bound coincides with the one for general regular languages. The upper bound for the union operation is only achievable if m is not a divisor or multiple of n. The witness languages are the same of the deterministic case: $(a^m)^\star$ and $(a^n)^\star$. The same witnesses are used to prove the tightness of the upper bound for the nondeterministic state complexity of intersection, which only occurs if m and n are coprimes. Considering the concatenation, it is not known the tightness of the upper bound $m + n$. However, considering the languages $\{a^l \mid l = m - 1 \pmod m\}$ and $\{a^l \mid l = (n - 1) \pmod n\}$, the lower bound $m + n - 1$ is achieved. The same languages can be used to show the tightness of the bound for the star operation. Holzer & Kutrib also proved that the upper bound for the nondeterministic state complexity of the complement is tight.

Concerning the nondeterministic transition complexity, the results in Table 3.1 were provided by Domaratzki & Salomaa [DS07] and they used a refined number of transitions (ntc_σ) for a more precise computation of the operational transition complexity. For union, intersection and concatenation the families of languages which reach the upper bound for the nondeterministic transition complexity are the same families we presented for the nondeterministic state complexity. The languages $(a + b)^\star a(a + b^{m-3}a(a+b)^\star$ for $m \geq 3$ witness that the upper bound for complement is reached. This family was presented by Holzer & Kutrib to show that for any integer $n > 2$ there exists an n-state NFA A such that any NFA that accepts the complement of $\mathcal{L}(A)$ needs at least 2^{n-2} states. For the star operation, the upper bound is achieved for the languages $a^{k-1}b(a^kb)^\star$. Considering the reversal, the languages $(a^k)^\star((b^2)^+ + (c^2)^+)$

Table 3.3: State complexity and nondeterministic state complexity of basic regularity preserving operations on finite languages.

Operation	sc	nsc
$L_1 \cup L_2$	$mn - (m+n)$	$m+n-2$
$L_1 \cap L_2$	$mn - 3(m+n) + 12$	mn
L^C	m	$\theta(k^{\frac{m}{1+log(k)}})$
$L_1 L_2$	$\sum_{i=0}^{m-2} \min\left\{k^i, \sum_{j=0}^{f(A,i)} \binom{n-2}{j}\right\} + \sum_{j=0}^{f(A)} \binom{n-2}{j}$	$m+n-1$
$L^\star$	$2^{m-f(A)-2} + 2^{m-3}$	$m-1$
L^R	$\sum_{i=0}^{l-1} k^i + 2^{m-l-1}$	m

witness the tightness of this operation.

Finite languages, that are the languages accepted by acyclic finite automata, are an important subset of regular languages. Câmpeanu et al. [CCSY01] presented the first formal study of state complexity of operations on finite languages. They studied the operational state complexity of concatenation, star, and reversal. Yu [Yu01] presented upper bounds for the union and the intersection, but the tight upper bounds were given by Han & Salomaa [HS08] using growing size alphabets. In this work (Section 4.2) we study the state and transition complexity of basic regularity preserving operations, for incomplete DFAs representing finite languages. Nondeterministic state complexity of basic operations on finite languages were studied by Holzer & Kutrib [HK03].

Let L_1 and L_2 such that $\mathrm{sc}(L_1) = m$ ($\mathrm{nsc}(L_1) = m$) and $\mathrm{sc}(L_2) = n$ ($\mathrm{nsc}(L_2) = n$), and let A be the complete minimal DFA (NFA) such that $L_1 = L(A)$ and B be the complete minimal DFA (NFA) such that $L_2 = L(B)$. Table 3.3 presents some results on deterministic and nondeterministic state complexity of basic regularity preserving operations on finite languages, where $f(A)$ is the number of final states of DFA A, and $f(A, i)$ is the larger number of final states of any path from the initial state to the state i in DFA A.

Câmpeanu et al. gave tight upper bounds for the state complexity of concatenation, star and reversal operations. For concatenation the DFAs of the witness languages are

presented in Figure 3.3. The upper bound for the star operation is achieved for the

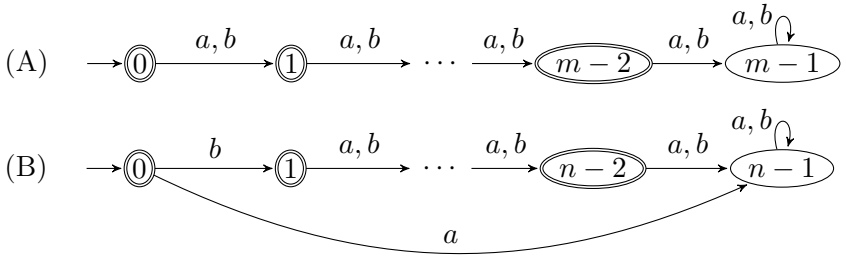


Figure 3.3: Witness DFAs for the state complexity of concatenation on finite languages.

family of languages accepted by the DFAs presented in Figure 3.4. Concerning the reversal operation the Figure 3.5 present binary languages tight bound witnesses.

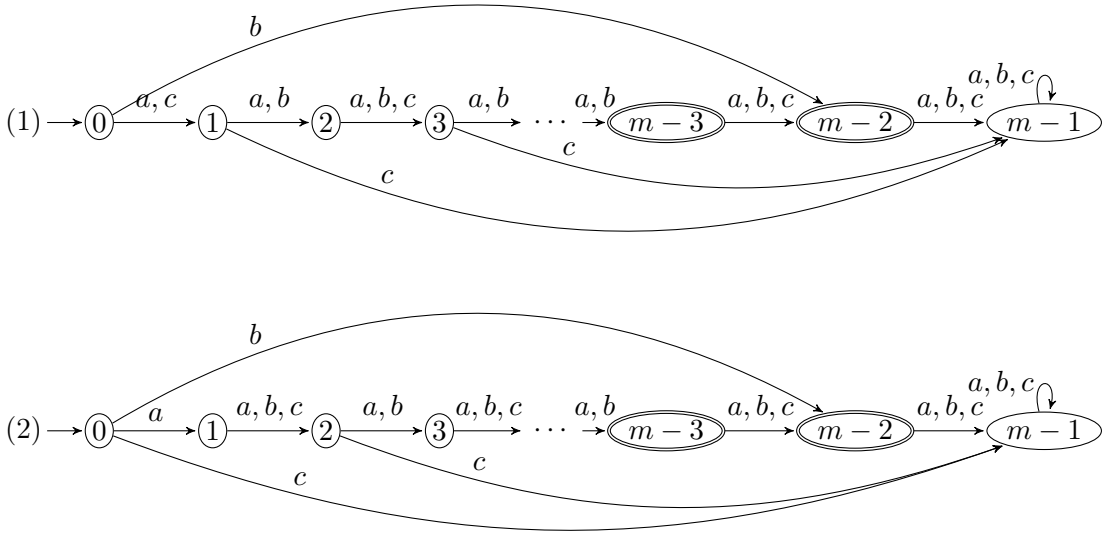


Figure 3.4: Witness DFA for the state complexity of star on finite languages, with m even (1) and odd (2).

Nondeterministic state complexity of basic operations on finite languages were studied by Holzer & Kutrib [HK03]. The authors show that the finite languages a^m and b^n are witnesses for the necessity of the number of states for the union in the worst case. For the intersection, the upper bound and the witness of tightness coincides with the general case. The tight bound for complement is reached for alphabets $\Sigma = \{a_1, \cdots, a_k\}$ of size $k \geq 2$, and the languages $\Sigma^j a_1 \Sigma^i y$, where $i \geq 0$, $0 \geq j \geq i$,

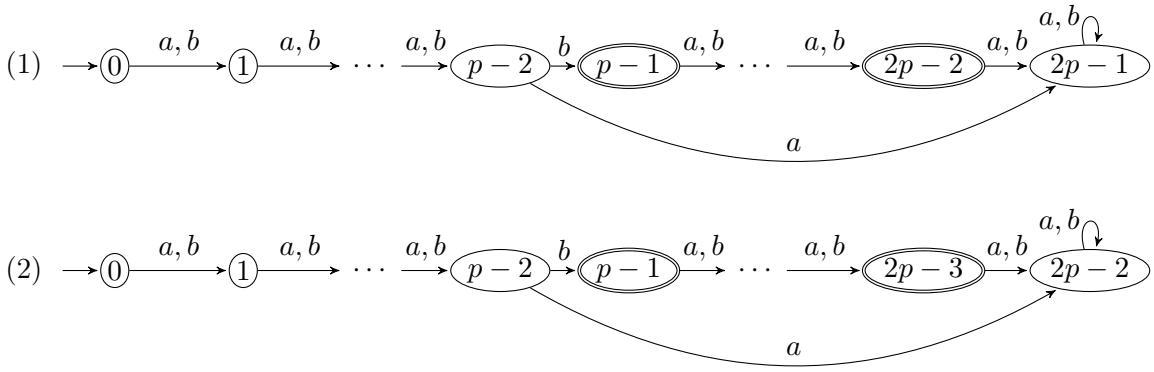


Figure 3.5: Witness DFA for the state complexity of reversal on finite languages, with $2p-1$ states (1) and with $2p-2$ (2).

Table 3.4: State complexity and nondeterministic state complexity of basic regularity preserving operations on finite unary languages.

Operation	sc	nsc
$L_1 \cup L_2$	$\max(m,n)$	$\max(m,n)$
$L_1 \cap L_2$	$\min(m,n)$	$\min(m,n)$
L^C	m	$m+1$
L_1L_2	$m+n-2$	$m+n-1$
$L^\star$	$m^2-7m+13$ for $m>4$	$m-1$
L^R	m	m

$y \in \Sigma \setminus \{a_1\}$, and $m > 2$. For concatenation, the witness languages can be the ones used for union. The languages a^m witness that the upper bound for the star operation is achieved. Witness languages for reversal operation are $(a+b)^{m-1}$.

The results in Table 3.3 show that the (nondeterministic) state complexity of operations on finite languages are, in general, lower than in the general case.

Table 3.4 summarises the state complexity and nondeterministic state complexity results of some basic operations on finite unary languages [CCSY01, Yu01, HK02]. State complexity of union, intersection, and concatenation on finite unary languages are linear, while they are quadratic for general unary languages. The tightness of the bounds are not difficult to prove, even considering the nondeterminism.

3.2 Average-case Descriptional Complexity

Usually, studies on descriptional complexity consider worst-case analysis, for which well established methods are known. However, study the worst-case complexity is not enough for a complete description of the objects and algorithms. A worst-case behaviour seldom occurs, and a worst-case upper bound can be of little use in practical applications. Normally, the worst-case complexity does not reflect the real life algorithm performance. So, for practical purposes, an estimate for the average case constitutes a much more useful information.

Average-case complexity turns out to be much harder to determine than worst-case complexity. Most known results on average-case complexity were obtained using generating functions and complex analysis. The analytic combinatorics framework provides a tool for asymptotic average-case analysis, by relating the enumeration of combinatorial objects to the algebraic and complex analytic properties of generating functions. Another approach used to study the average complexity is to perform statistically significant experiments, considering uniform random generators. However, with this approach only small ranges of object sizes can be considered. Usually, both in experimental and analytic results, a uniform distribution is considered.

Although its evident practical importance, there is still very few research on average-case complexity. Concerning average state complexity, Nicaud [Nic99] proved that the state complexity of union, intersection and concatenation on two unary languages L_1 and L_2 is asymptotically equivalent to mn, where $m = \mathrm{sc}(L_1)$ and $n = \mathrm{sc}(L_2)$. The average operational state complexity on finite languages is studied by Gruber & Holzer [GH07] and by Bassino et al. [BGN10]. Felice & Nicaud [FN13, FN14] study the average-case computational complexity of the Brzozowski minimisation algorithm which provide some characterisations of the state complexity of reversal.

Regarding the asymptotic average size of NFAs equivalent to a given regular expression, Nicaud [Nic09] proved that the average size of the Glushkov automata is linear on the size of the original regular expression, which is quadratic in the worst-case. Broda et

al. [BMMR11, BMMR12] proved that the size of partial derivative automaton is on average half of the size of the Glushkov automaton.

3.2.1 Generating Functions and Analytic methods

In this section we introduce some of the results on analytic combinatorics which will be used during this work. For a gentle introduction to the basic analytical tools of this theory, with some illustrative examples using regular expressions, one points the reader to [BMMR14].

The symbolic method is a general way to count families of combinatorial objects, since it permits to directly and almost automatically build the generating functions associated to combinatorial classes families.

A *combinatorial class* C is a set of objects on which a non-negative integer size function $|\cdot|$ is defined, and such that for each $n \geq 0$, the number of objects of size n in C, c_n, is finite. The sequence $c_0, c_1, c_2, \cdots$ is called the counting sequence of the class C.

The *generating function* $C(z)$ of a combinatorial class C is the formal series

$$\mathcal{G}(C) = C(z) = \sum_{c \in C} z^{|c|} = \sum_{n=0}^{\infty} c_n z^n.$$

We denote by $[z^n]C(z)$ the coefficient c_n of z^n in $C(z)$.

The *symbolic method* allows the construction of a combinatorial class C in terms of simpler ones, $B_1, \ldots, B_n$, such that the generating function of C ($C(z)$) is a function of the generating functions of B_i, for $1 \leq i \leq n$. For example, if A and B are two disjoint combinatorial classes, with generating functions $A(z)$ and $B(z)$, respectively, then $A \cup B$ is a combinatorial class whose generating function is $A(z) + B(z)$. Moreover, if we consider the combinatorial class $A \times B$ its generating function is given by $A(z)B(z)$. The Kleene closure is other usual admissible operation.

Following Flajolet, let C be a combinatorial class of generating function $C(z)$ and let

$f : C \to \mathbb{R}$ be a mapping from this class to $\mathbb{R}$. The *cost generating function* $F(z)$ of C associated to f is

$$F(z) = \sum_{c \in C} f(c) z^{|c|} = \sum_{n \geq 0} f_n z^n, \text{ with } f_n = \sum_{c \in C, |c| = n} f(c).$$

For a given n, the average value of f for the uniform distribution on the elements of size n of C is, obviously,

$$\mu_n(C, f) = \frac{[z^n]F(z)}{[z^n]C(z)}.$$

Once a generating function is known, we can compute asymptotic estimations of its coefficients, using the theory of complex analysis, seeing generating functions as analytic complex functions in $\mathbb{C}$. Studying the generating function around its dominant singularities we obtain the asymptotics of its coefficients.

Theorem 3.1. *The coefficients of the function* $f(z) = (1 - z)^{-\alpha}$ *where* $\alpha \in \mathbb{C} \setminus \mathbb{Z}_0^-$, *have the following asymptotic approximation:*

$$[z^n]f(z) = \frac{n^{\alpha - 1}}{\Gamma(\alpha)} + o(n^{\alpha - 1})$$

where Γ *is Euler's gamma function.*

For $R \geq 1$, $\xi \in \mathbb{C}$ and $0 \geq \phi \geq \pi/2$, the domain $\Delta(\xi, \phi, R)$ at $z = \xi$ is the open set

$$\Delta(\xi, \phi, R) = \{z \in \mathbb{C} \mid |z| < R, z \neq \xi \text{ and } |Arg(z - \xi)| > \phi\}$$

where $Arg(z)$ denotes the argument of $z \in \mathbb{C}$. A *domain* is a Δ-*domain* at ξ if it is of the form $\Delta(\xi, \phi, R)$ for some ξ, ϕ and R.

We will consider that the generating functions have always a unique dominant singularity and satisfy one of the two conditions of the following proposition.

Proposition 3.2. *Let* $f(z)$ *be a function that is analytic in some* Δ*-domain at* $\rho \in \mathbb{R}^+$.

1) If on the intersection of a neighbourhood of ρ and its Δ-domain,

$$f(z) = a - b\sqrt{1 - z/\rho} + o(\sqrt{1 - z/\rho}), \text{ with } a,\ b \in \mathbb{R}, b \neq 0$$

then $[z^n]f(z) \sim \frac{b}{2\sqrt{\pi}}\rho^{-n}n^{-3/2}$.

2) If on the intersection of a neighbourhood of ρ and its Δ-domain,

$$f(z) = \frac{a}{\sqrt{1 - z/\rho}} + o(\frac{1}{\sqrt{1 - z/\rho}}), \text{ with } a \in \mathbb{R},\ a \neq 0,$$

then $[z^n]f(z) \sim \frac{a}{\sqrt{\pi}}\rho^{-n}n^{-1/2}$.

The following lemma it is useful in some analytic computations.

Lemma 3.3. *If $f(z)$ is an entire function with $\lim_{z\to\rho} f(z) = a$ and $r \in \mathbb{R}$, then*

$$f(z)(1 - z/\rho)^r = a(1 - z/\rho)^r + o((1 - z/\rho)^r).$$

In the following section we present a simple example that illustrates the use of the symbolic method to compute the generating function corresponding to the regular expressions given by a particular grammar. We also estimate the number of letters in regular expressions of a given size.

3.2.1.1 From a Grammar to a Generating Function

Let R_k be the set of regular expressions defined by the following grammar:

$$\alpha := \varepsilon \mid \sigma_1 \mid \cdots \mid \sigma_k \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^\star.$$

Consider that the size of a regular expression is its number of symbols (letters and operators) not counting parentheses, as we already referred. Equipped with this size function, R_k is a combinatorial class.

Using the recursive definition of R_k given by the grammar, we will compute the associated generating function $R_k(z) = \sum_{n\geq 0} r_n z^n$. Note that the values of r_n are the number of regular expressions α of size n. The regular expression α can be either a letter σ_i or one of the forms $\alpha + \alpha$, $\alpha \cdot \alpha$ or $\alpha^\star$. Since these are disjoint cases, they have to be counted separately using the symbolic method already presented:

$$\begin{aligned} R_k(z) &= (k+1)z + \mathcal{G}(R_k \times \{+\} \times R_k) + \mathcal{G}(R_k \times \{\cdot\} \times R_k) + \mathcal{G}(R_k \times \{\star\}) \\ &= (k+1)z + zR_k(z)^2 + zR_k(z)^2 + zR_k(z) \\ &= (k+1)z + 2zR_k(z)^2 + zR_k(z) \end{aligned}$$

Solving this equation for $R_k(z)$, we obtain two possible solutions:

$$R_k(z) = \frac{1 - z \pm \sqrt{\Delta_k(z)}}{4z}, \text{ where } \Delta_k(z) = 1 - 2z - (7+8k)z^2.$$

As $R_k(0) = r_0 = 0$, one must have $\lim_{z\to 0} R_k(z) = 0$, which is satisfied only by

$$R_k(z) = \frac{1 - z - \sqrt{\Delta_k(z)}}{4z}.$$

The zeros of $\Delta_k(z)$ are

$$\rho_k = \frac{1}{1 + 2\sqrt{2+2k}} \text{ and } \overline{\rho_k} = \frac{1}{1 - 2\sqrt{2+2k}}.$$

The coefficients of the series $\tilde{R}_k(z) = 4zR_k(z) + z = 1 - \sqrt{\Delta_k(z)}$, have the same asymptotical behaviour of the ones of $R_k(z)$.

We know that

$$\begin{aligned} \Delta_k(z) &= (7+8k)(z - \rho_k)(z\overline{\rho}_k) \\ &= (7+8k)(z - \overline{\rho_k})\rho_k(1 - z/\rho_k) \end{aligned}$$

and

$$(7+8k)(\rho_k - \overline{\rho}_k) = 4\sqrt{2+2k}\rho_k.$$

Thus by Lemma 3.3,

$$\begin{aligned}\sqrt{\Delta_k(z)} &= \sqrt{4\sqrt{2+2k}\rho_k}\sqrt{(1-z/\rho_k)} + o(\sqrt{(1-z/\rho_k)}) \\ &= 2\sqrt[4]{2+2k}\sqrt{\rho_k}\sqrt{(1-z/\rho_k)} + o(\sqrt{(1-z/\rho_k)}).\end{aligned}$$

Therefore,

$$\tilde{R}_k(z) = 4zR_k(z) + z = 1 - 2\sqrt[4]{2+2k}\sqrt{\rho_k}\sqrt{(1-z/\rho_k)} + o(\sqrt{(1-z/\rho_k)}$$

By Proposition 3.2, one obtains

$$\begin{aligned}[z^n](4zR_k(z)+z) &\sim \frac{\sqrt[4]{2+2k}\sqrt{\rho_k}}{\sqrt{\pi}}\rho_k^{-n}n^{-3/2}, \\ [z^n]R_k(z) &\sim \frac{\sqrt[4]{2+2k}\sqrt{\rho_k}}{4\sqrt{\pi}}\rho_k^{-(n+1)}(n+1)^{-3/2}\end{aligned}$$

where $[z^n]R_k(z)$ is the number of regular expressions α with size n.

Nicaud showed that the cost generating function for the number of letters in a regular expressions α is

$$L_k(z) = \frac{kz}{\sqrt{\Delta_k(z)}},$$

and satisfies

$$[z^n]L_k(z) \sim \frac{k\rho_k}{\sqrt{\pi(2-2\rho_k)}}\rho_k^{-n}n^{-1/2}.$$

From this we can deduce that for a given n, the average number of letters in a regular expression of size n is given by

$$\frac{[z^n]L_k(z)}{[z^n]R_k(z)} \sim \frac{4k\rho_k^2}{1-\rho_k}n.$$

It is easy to see that

$$\lim_{k\to\infty} \frac{4k\rho_k^2}{1-\rho_k} \nearrow \frac{1}{2},$$

which means that, for large alphabets, the average number of letters in a regular expression grows to about half of its size.

Chapter 4

Operational Complexity on Incomplete DFAs

The descriptional complexity of regular languages has been extensively investigated in the last years, as we already saw in the previous chapter. The complexity measure usually studied for DFAs is the state complexity. However for NFAs and incomplete DFAs the transition complexity is generally considered a more interesting measure.

In this chapter we study the incomplete operational transition complexity of several operations on regular and finite languages. To be comprehensive we also analyse the state complexity of the resulting languages. In general, transition complexity bounds depend not only on the complexities of the operands but also on other refined measures, as the number of undefined transitions or the number of transitions that leave the initial state. For both families of languages we performed some experimental tests in order to have an idea of the average-case complexity of those operations. This study was presented by Maia et al. [MMR15a], and it expands the contributions in two extended abstracts from the same authors [MMR13b, MMR13a].

In Section 4.1, we study the state and transition complexity for the union, concatenation, Kleene star and reversal operations on regular languages. For all these operations tight upper bounds are given. The tight upper bound presented for the

Table 4.1: Incomplete transition complexity for regular and finite languages, where m and n are the (incomplete) state complexities of the operands, $f_1(m,n) = (m-1)(n-1)+1$ and $f_2(m,n) = (m-2)(n-2)+1$. The column $|\Sigma|$ indicates the minimal alphabet size for which the upper bound is reached.

Operation	Regular	$\|\Sigma\|$	Finite	$\|\Sigma\|$
$L_1 \cup L_2$	$2n(m+1)$	2	$3(mn-n-m)+2$	$f_1(m,n)$
$L_1 \cap L_2$	nm	1	$(m-2)(n-2)(2+\sum_{i=1}^{\min(m,n)-3}(m-2-i)(n-2-i))+2$	$f_2(m,n)$
L^C	$m+2$	1	$m+1$	1
L_1L_2	$2^{n-1}(6m+3)-5$, if $m,n \geq 2$	3	$2^n(m-n+3)-8$, if $m+1 \geq n$	2
			See Theorem 4.18(4.7)	$n-1$
$L^\star$	$3.2^{m-1}-2$, if $m \geq 2$	2	$9 \cdot 2^{m-3} - 2^{m/2} - 2$, if m is odd	3
			$9 \cdot 2^{m-3} - 2^{(m-2)/2} - 2$, if m is even	
L^R	$2(2^m-1)$	2	$2^{p+2}-7$, if $m = 2p$	2
			$3 \cdot 2^p - 8$, if $m = 2p-1$	

transition complexity of the union operation refutes the conjecture presented by Gao et al. [GSY11]. We also present the same study for unary regular languages. In Subsection 4.1.6 we analyse some experimental results. In the Section 4.2 we continue the line of research of Section 4.1 considering finite languages. For the concatenation, we correct the upper bound for the state complexity of complete DFAs [CCSY01], and show that if the *right* operand is larger than the *left* one, the upper bound is only reached using an alphabet of variable size. We also present some experimental results for finite languages. The algorithms and the witness language families used, although new, are based on the ones of Yu et al. [YZS94]; several proofs required new techniques.

Table 4.1 presents a summary and a comparison of the obtained results for transition complexity on general and finite languages. Note that the values in the table are obtained using languages for which the upper bounds are reached.

To express the transition complexity of a language operation, we also use the following measures and refined numbers of transitions. Let $A = \langle Q, \Sigma, \delta, 0, F\rangle$ be a DFA, with $Q = [0, n[$, $\sigma \in \Sigma$, and $i \in Q$, we define

- $f(A) = |F|$;
- $t_\sigma(A, i) = \begin{cases} 1, & \text{if there exists a } \sigma\text{-transition leaving } i; \\ 0, & \text{otherwise;} \end{cases}$
- $\overline{t}_\sigma(A, i)$ is the complement of $t_\sigma(A, i)$;
- $s_\sigma(A) = t_\sigma(A, 0)$;
- $t_\sigma(A) = \sum_{i\in Q} t_\sigma(A, i)$;
- $u_\sigma(A) = |Q| - t_\sigma(A)$; and
- $\tilde{u}_\sigma(A)$ is the number of non-final states without σ-transitions.

Whenever there is no ambiguity we omit A from the above definitions. All the above measures can be defined, for a regular language L, considering the measure values for its minimal DFA. Thus we can use following notation, $f(L)$, $s_\sigma(L)$, $t_\sigma(L)$, $u_\sigma(L)$, and $\tilde{u}_\sigma(L)$, respectively.

4.1 Regular Languages

Gao et al. [GSY11] were the first to study the transition complexity of Boolean operations on regular languages based on incomplete DFAs. For the intersection and the complement, tight bounds were presented, but for the union operation the upper and lower bounds differ by a factor of two. Nevertheless, they conjectured a tight upper bound for this operation.

In this section, we continue this study by extending the analysis to the concatenation, the Kleene star and the reversal operations. For these operations tight upper bounds

Table 4.2: State complexity of basic regularity preserving operations on regular languages.

Operation	sc	isc	nsc
$L_1 \cup L_2$	mn	$\mathbf{mn + m + n}$	$m + n + 1$
$L_1 \cap L_2$	mn	mn	mn
L^C	n	$n + 1$	2^n
$L_1 L_2$	$m2^n - f_1 2^{n-1}$	$\mathbf{(m + 1)2^n - f_1 2^{n-1} - 1}$	$m + n$
$L^\star$	$2^{m-1} + 2^{m-l-1}$	$\mathbf{2^{m-1} + 2^{m-l-1}}$	$m + 1$
L^R	2^m	$\mathbf{2^m - 1}$	$m + 1$

are given. We also give a tight upper bound for the transition complexity of the union, which refutes the conjecture presented by Gao et al., as we already mentioned. We also prove that the upper bounds are maximal when $f(L)$ is minimal. This study is also done for unary regular languages.

In Tables 4.2 and 4.3 we summarise the results of this section (in bold) as well as some known results for other descriptional complexity measures: state complexity, and nondeterministic transition complexity, already referred in Section 3.1.

At the end of the section, we present some experimental results in order to analyse the descriptional complexity measures when the referred operations are performed with uniformly random generated DFAs as operands. These experiments allow the reader to make an approximate prediction of the average-case complexity of the operations.

4.1.1 Union

It was shown by Gao et al. [GSY11] that

$$\mathrm{itc}(L_1 \cup L_2) \leq 2(\mathrm{itc}(L_1)\,\mathrm{itc}(L_2) + \mathrm{itc}(L_1) + \mathrm{itc}(L_2)).$$

The lower bound $\mathrm{itc}(L_1)\,\mathrm{itc}(L_2) + \mathrm{itc}(L_1) + \mathrm{itc}(L_2) - 1$ was given for particular ternary language families which state complexities are relatively prime. The authors conjec-

Table 4.3: Transition complexity of basic regularity preserving operations on regular languages.

Operation	itc	ntc
$L_1 \cup L_2$	$\mathrm{itc}(\mathbf{L_1})(\mathbf{1}+\mathbf{n}) + \mathrm{itc}(\mathbf{L_2})(\mathbf{1}+\mathbf{m}) - \sum_{\sigma\in\mathbf{\Sigma}} \mathrm{itc}_\sigma(\mathbf{L_2})\,\mathrm{itc}_\sigma(\mathbf{L_1})$	$\mathrm{ntc}(L_1)+\mathrm{ntc}(L_2)+s(L_1)+s(L_2)$
$L_1 \cap L_2$	$\mathrm{itc}(L_1)\,\mathrm{itc}(L_2)$	$\sum_{\sigma\in\Sigma} \mathrm{ntc}_\sigma(L_1)\,\mathrm{ntc}_\sigma(L_2)$
L^C	$\lvert\Sigma\rvert(\mathrm{itc}(L)+2)$	$\lvert\Sigma\rvert 2^{\mathrm{ntc}(L)+1}$
		$2^{\frac{\mathrm{ntc}(L)}{2}-2}-1$
L_1L_2	$\lvert\mathbf{\Sigma}\rvert(\mathbf{m}+\mathbf{1})\mathbf{2^n} - \lvert\mathbf{\Sigma_c^{L_2}}\rvert(\mathbf{f}\ \mathbf{2^{n-1}}+\mathbf{1}) - \sum_{\sigma\in\mathbf{\Sigma_i^{L_2}}} (\mathbf{2}^{\mathbf{u}_\sigma} + \mathbf{f}\ \mathbf{2}^{\mathrm{itc}_\sigma(\mathbf{L_2})}) - \sum_{\sigma\in\mathbf{\Sigma_{ii}}} \tilde{\mathbf{u}}_\sigma \mathbf{2}^{\mathbf{u}_\sigma} - \sum_{\sigma\in\mathbf{\Sigma_{ic}}} \tilde{\mathbf{u}}_\sigma$	$\mathrm{ntc}(L_1)+\mathrm{ntc}(L_2)+f_{in}(L_1)$
$L^\star$	$\lvert\mathbf{\Sigma}\rvert(\mathbf{2^{m-l-1}}+\mathbf{2^{m-1}}) + \sum_{\sigma\in\mathbf{\Sigma_i}} (\mathbf{s}_\sigma - \mathbf{2}^{\tilde{\mathbf{u}}_\sigma})$	$\mathrm{ntc}(L)+f_{in}(L)$
L^R	$\lvert\mathbf{\Sigma}\rvert(\mathbf{2^m}-\mathbf{1})$	$\mathrm{ntc}(L)+f(L)$

tured, also, that

$$\mathrm{itc}(L_1 \cup L_2) \leq \mathrm{itc}(L_1)\,\mathrm{itc}(L_2) + \mathrm{itc}(L_1) + \mathrm{itc}(L_2),$$

when $\mathrm{itc}(L_i) \geq 2$, $i = 1, 2$.

We will present an upper bound for the state complexity and we give a new upper bound for the transition complexity of the union of two regular languages. We also present families of languages for which these upper bounds are reached, witnessing, thus, that these bounds are tight.

Following, we describe the algorithm for the union of two DFAs, based on the usual product construction, that was presented by Gao et al. [GSY11, Lemma 3.1.]. Given two incomplete DFAs $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ and $B = \langle [0, n[, \Sigma, \delta_B, 0, F_B\rangle$, and considering Ω_A and Ω_B as the dead states of A and B, respectively, let $C = \langle ([0, m[\cup\{\Omega_A\}) \times ([0, n[\cup\{\Omega_B\})), \Sigma, \delta_C, (0, 0), (F_A \times ([0, n[\cup\{\Omega_B\})) \cup (([0, m[\cup\{\Omega_A\}) \times F_B)\rangle$ be a new DFA where for $\sigma \in \Sigma$, $i \in [0, m[\cup\{\Omega_A\}$, and $j \in [0, n[\cup\{\Omega_B\}$,

$$\delta_C((i,j),\sigma) = \begin{cases} (\delta_A(i,\sigma),\delta_B(j,\sigma)), & \text{if } \delta_A(i,\sigma)\downarrow \wedge\ \delta_B(j,\sigma)\downarrow; \\ (\delta_A(i,\sigma),\Omega_B), & \text{if } \delta_A(i,\sigma)\downarrow \wedge\ \delta_B(j,\sigma)\uparrow; \\ (\Omega_A,\delta_B(j,\sigma)), & \text{if } \delta_A(i,\sigma)\uparrow \wedge\ \delta_B(j,\sigma)\downarrow; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Note that $\delta_A(\Omega_A,\sigma)$ and $\delta_B(\Omega_B,\sigma)$ are always undefined, and the pair (Ω_A,Ω_B) never occurs in the image of δ_C. It is easy to see that DFA C accepts the language $\mathcal{L}(A)\cup\mathcal{L}(B)$. The number of states and transitions which are sufficient for any DFA C are obtained in the following theorem.

Theorem 4.1. *For any two regular languages L_1 and L_2 with $\mathrm{isc}(L_1)=m$ and $\mathrm{isc}(L_2)=n$, one has $\mathrm{isc}(L_1\cup L_2)\leq mn+m+n$ and*

$$\mathrm{itc}(L_1\cup L_2)\leq \mathrm{itc}(L_1)(1+n)+\mathrm{itc}(L_2)(1+m)-\sum_{\sigma\in\Sigma}\mathrm{itc}_\sigma(L_1)\,\mathrm{itc}_\sigma(L_2).$$

Proof. Let A and B be the minimal DFAs that recognise L_1 and L_2, respectively. Consider the DFA C such that $\mathcal{L}(C)=\mathcal{L}(A)\cup\mathcal{L}(B)$ and C is constructed using the algorithm described above. The result for the $\mathrm{isc}(L_1\cup L_2)$ is given by Gao et al. in [GSY11]. Let us prove the result for the $\mathrm{itc}(L_1\cup L_2)$. Consider the σ-transitions of A named by α_i ($i\in[1,t_\sigma(A)]$) and the undefined σ-transitions of A named by $\bar{\alpha}_l$ ($l\in[1,u_\sigma(A)+1]$). Consider also the σ-transitions of B named by β_j ($j\in[1,t_\sigma(B)]$) and the undefined σ-transitions named by $\bar{\beta}_z$ ($z\in[1,u_\sigma(B)+1]$). We need to consider one more undefined transition in each DFA which corresponds to Ω_A and Ω_B. The σ-transitions of the DFA C accepting $L_A\cup L_B$ can only have one of the following three forms: (α_i,β_j), $(\bar{\alpha}_l,\beta_j)$, and $(\alpha_i,\bar{\beta}_z)$. Thus the DFA C has $t_\sigma(A)t_\sigma(B)$ σ-transitions of the form (α_i,β_j); $t_\sigma(A)(u_\sigma(B)+1)$ σ-transitions of the form $(\bar{\alpha}_l,\beta_j)$; and $(u_\sigma(A)+1)t_\sigma(B)$ σ-transitions of the form $(\alpha_i,\bar{\beta}_z)$. As we know that $u_\sigma(A)=m-t_\sigma(A)$ and $u_\sigma(B)=n-t_\sigma(B)$, the number of σ-transitions is

$$t_\sigma(A)t_\sigma(B)+t_\sigma(A)(n-t_\sigma(B)+1)+t_\sigma(B)(m-t_\sigma(A)+1).$$

Therefore, with $\mathrm{itc}_\sigma(\mathcal{L}(A)) = t_\sigma(A)$ and $\mathrm{itc}_\sigma(\mathcal{L}(B)) = t_\sigma(B)$ the inequality holds. □

4.1.1.1 Worst-case Witnesses

In this section, we show that the upper bounds established in Theorem 4.1 are tight. We need to consider two cases, parametrised by the state complexities of the language operands: $m \geq 2$ and $n \geq 2$; and $m = 1$ and $n \geq 2$ (or vice versa). Note that, in this section, we consider automaton families over a binary alphabet, $\Sigma = \{a, b\}$.

Case 1: $m \geq 2$ and $n \geq 2$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, \{0\}\rangle$ with $\delta_A(m-1, a) = 0$, and $\delta_A(i, b) = i + 1,\ 1 \in [0, m-1[$; and $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\}\rangle$ with $\delta_B(i, a) = i + 1,\ i \in [0, n-1[$, and $\delta_B(i, b) = i,\ i \in [0, n[$. These minimal DFAs are represented in Figure 4.1 and Figure 4.2, respectively.

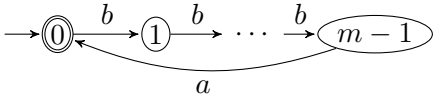


Figure 4.1: DFA A with m states.

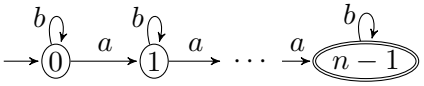


Figure 4.2: DFA B with n states.

Theorem 4.2. *For any integers $m \geq 2$ and $n \geq 2$, there exist an m-state DFA A with $r = m$ transitions and an n-state DFA B with $s = 2n - 1$ transitions such that any DFA accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$ needs, at least, $mn + m + n$ states and $(r + 1)(s + 1)$ transitions.*

Proof. Let us count the number of states of the DFA C accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$, constructed by the previous algorithm. Consider the pairs (i, j) representing states of that DFA C. Then for each (i, j) where $i \in ([0, m[\cup \Omega_A)$ and $j \in ([0, n[\cup \Omega_B)$ except the case when $(i, j) = (\Omega_A, \Omega_B)$, there exists a word

$$w = \begin{cases} (b^{m-1}a)^j b^i, & \text{if } i \neq \Omega_A \wedge j \neq \Omega_B; \\ (b^{m-1}a)^n b^i, & \text{if } i \neq \Omega_A \wedge j = \Omega_B; \\ b^m a^j, & \text{if } i = \Omega_A \wedge j \neq \Omega_B; \end{cases}$$

which represents each state, i.e., a different left quotient. Thus there are at least $mn + m + n$ distinct left quotients (states of C).

Let us consider the number of transitions of DFA C. If we name the defined and undefined transitions of the DFAs A and B as in the proof of the Theorem 4.1 then C has:

- $mn + n - m + 1$ a-transitions because there exist $n - 1$ a-transitions of the form (α_i, β_j); 2 a-transitions of the form $(\alpha_i, \bar{\beta}_j)$; and $m(n-1)$ a-transitions of the form $(\bar{\alpha}_i, \beta_j)$;
- $mn + m + n - 1$ b-transitions because there exist $(m-1)n$ b-transitions of the form (α_i, β_j); $m - 1$ b-transitions of the form $(\alpha_i, \bar{\beta}_j)$; and $2n$ b-transitions of the form $(\bar{\alpha}_i, \beta_j)$.

As $r = m$ and $s = 2n - 1$, DFA C has $(r + 1)(s + 1)$ transitions. □

The referred conjecture $\mathrm{itc}(L_1 \cup L_2) \leq \mathrm{itc}(L_1)\,\mathrm{itc}(L_2) + \mathrm{itc}(L_1) + \mathrm{itc}(L_2)$ fails for these families because, as we prove in the previous theorem, $\mathrm{itc}(L_1 \cup L_2) = (r + 1)(s + 1)$, where $r = \mathrm{itc}(L_1)$ and $s = \mathrm{itc}(L_2)$, then $\mathrm{itc}(L_1 \cup L_2) = \mathrm{itc}(L_1)\,\mathrm{itc}(L_2) + \mathrm{itc}(L_1) + \mathrm{itc}(L_2) + 1$.

Case 2: $m = 1$ and $n \geq 2$. Let $A = \langle \{0\}, \Sigma, \delta_A, 0, \{0\} \rangle$ with $\delta_A(0, a) = 0$, and consider the DFA B defined in the previous case.

Theorem 4.3. *For any integer $n \geq 2$, there exists an 1-state DFA A with one transition and an n-state DFA B with $s = 2n - 1$ transitions such that any DFA*

accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$ *has, at least,* $2n+1$ *states and* $2(s+1)$ *transitions.*

Proof. Consider the DFA C, accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$, constructed by the previous algorithm. As in the proof of Theorem 4.2, let us see the states of DFA C as pairs (i, j) where $i \in (\{0\} \cup \Omega_A)$ and $j \in ([0, n[\cup \Omega_B)$ except the case when $(i, j) = (\Omega_A, \Omega_B)$. For each of those pairs, there exists a word,

$$w = \begin{cases} a^j, & \text{if } i \neq \Omega_A \wedge j \neq \Omega_B; \\ ba^j, & \text{if } i = \Omega_A \wedge j \neq \Omega_B; \\ a^n, & \text{if } i \neq \Omega_A \wedge j = \Omega_B; \end{cases}$$

which represents a state of C, i.e., a different left quotient. Thus there are at least $2n+1$ distinct left quotients.

Let us consider the transitions named as in the proof of the Theorem 4.1, then DFA C has:

- $2n$ a-transitions because there exist $n-1$ a-transitions of the form (α_i, β_j); 2 a-transitions of the form $(\alpha_i, \bar{\beta}_j)$; and $n-1$ a-transitions of the form $(\bar{\alpha}_i, \beta_j)$;
- $2n$ b-transitions because by this symbol there are only transitions of the form $(\bar{\alpha}, \beta_j)$.

Thus, the DFA C has $4n$ transitions. As $r = 1$ and $s = 2n-1$, the DFA C has $2(s+1)$ transitions. Note that $r = 1$ and, thus, $2(s+1) = (r+1)(s+1)$. □

4.1.2 Concatenation

In this section we deal with the incomplete descriptional complexity of the concatenation of two regular languages.

The construction used is as follows. Given two incomplete DFAs, $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A \rangle$ and $B = \langle [0, n[, \Sigma, \delta_B, 0, F_B \rangle$, a DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ is $C = \langle R, \Sigma, \delta_C, r_0, F_C \rangle$

where for $\sigma \in \Sigma$, $i \in [0, m[$, and $P \subseteq [0, n[$, $R \subset ([0, m[\cup \{\Omega_A\}) \times 2^{[0,n[}$ (precisely defined in the proof of Theorem 4.4); r_0 is $(0, \emptyset)$ if $0 \notin F_A$, and is $(0, \{0\})$ otherwise; $F_C = \{(i, P) \in R \mid P \cap F_B \neq \emptyset\}$; and

$$\delta_C((q,T),\sigma) = \begin{cases} (\delta_A(q,\sigma), \delta_B(T,\sigma) \cup \{0\}), & \text{if } \delta_A(q,\sigma)\downarrow \wedge\ \delta_A(q,\sigma) \in F_A; \\ (\delta_A(q,\sigma), \delta_B(T,\sigma)), & \text{if } \delta_A(q,\sigma)\downarrow \wedge\ \delta_A(q,\sigma) \notin F_A; \\ (\Omega_A, \delta_B(T,\sigma)). & \text{if } \delta_A(q,\sigma)\uparrow \wedge\ \delta_B(T,\sigma) \neq \emptyset; \\ \uparrow, & \textit{otherwise.} \end{cases}$$

In the following, we determine the number of states and transitions that are sufficient for any DFA C resulting from the previous construction.

Given an automaton A, its alphabet can be partitioned in two sets, Σ_c^A and Σ_i^A, such that $\sigma \in \Sigma_c^A$ if A is σ-complete, and $\sigma \in \Sigma_i^A$ otherwise. In the same way, considering two automata A and B, the alphabet can be divided into four disjoint sets Σ_{ci}, Σ_{cc}, Σ_{ii} and Σ_{ic}. As before, these notations can be extended to regular languages considering their minimal DFAs.

Theorem 4.4. *For any regular languages L_1 and L_2 with* $\mathrm{isc}(L_1) = m$, $\mathrm{isc}(L_2) = n$, $u_\sigma = u_\sigma(L_2)$, $f = f(L_1)$ *and* $\tilde{u}_\sigma = \tilde{u}_\sigma(L_1)$, *one has* $\mathrm{isc}(L_1L_2) \leq (m+1)2^n - f2^{n-1} - 1$, *and*

$$\mathrm{itc}(L_1L_2) \leq |\Sigma|(m+1)2^n - |\Sigma_{ic} \cup \Sigma_{cc}|(f2^{n-1}+1) - \\ - \sum_{\sigma \in (\Sigma_{ci} \cup \Sigma_{ii})} (2^{u_\sigma} + f2^{\mathrm{itc}_\sigma(L_2)}) - \sum_{\sigma \in \Sigma_{ii}} \tilde{u}_\sigma 2^{u_\sigma} - \sum_{\sigma \in \Sigma_{ic}} \tilde{u}_\sigma.$$

Proof. Let A and B be the minimal DFAs that recognise L_1 and L_2, respectively. Consider the DFA C such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$, constructed using the algorithm described above. First, let us consider the problem of $\mathrm{isc}(L_1L_2)$. The set R is a set of pairs (s, P) where $s \in ([0, m[\cup \Omega_A)$, and $P \subseteq [0, n[$. There exist $(m+1)2^n$ such pairs. However, we know that R does not contain the pairs in which s is a final state of A and the set P does not contain the initial state of B. Thus, we need to remove

$f(A)2^{n-1}$ pairs from the first counting. As the pair $(\Omega_A, \emptyset)$ is not in R, we can also remove it. The resulting number of states is, thus, $(m+1)2^n - f(A)2^{n-1} - 1$.

Now, let us consider the problem of estimating itc(L_1L_2). We name the σ-transitions of A and B as in the proof of the Theorem 4.1 with a slight modification: $z \in [1, u_\sigma(B)]$. The σ-transitions of C are pairs (θ, γ) where θ is either an α_i or an $\bar{\alpha}_l$, and γ is a set of β_j or $\bar{\beta}_z$. By construction, C cannot have transitions where θ is an $\bar{\alpha}_l$, and γ is a set with only $\bar{\beta}_k$, because these pairs would correspond to undefined transitions. If $\sigma \in \Sigma_{ci}$, the number of C σ-transitions is $(t_\sigma(A)+1)2^{t_\sigma(B)+u_\sigma(B)} - 2^{u_\sigma(B)} - f(A)2^{t_\sigma(B)}$, because the number of θs is $t_\sigma(A)+1$ and the number of γs is $2^{t_\sigma(B)+u_\sigma(B)}$. We need to remove the $2^{u_\sigma(B)}$ sets of transitions of the form $(v, \emptyset)$ where v corresponds to the undefined σ-transition leaving the state Ω_A. If θ corresponds to a transition that leaves a final state of A, then γ needs to include the initial state of B. Thus we also remove $f(A)2^{t_\sigma(B)}$ pairs. If $\sigma \in \Sigma_{cc}$, C has $(t_\sigma(A)+1)2^{t_\sigma(B)} - 1 - f(A)2^{t_\sigma(B)-1}$ σ-transitions. In this case, $u_\sigma(B) = 0$. The only pair we need to remove is $(v, \emptyset)$ where v corresponds to the undefined σ-transition leaving the state Ω_A. Analogously, if $\sigma \in \Sigma_{ii}$, C has $(t_\sigma(A)+u_\sigma(A)+1)2^{t_\sigma(B)+u_\sigma(B)} - (\tilde{u}_\sigma(A)+1)2^{u_\sigma(B)} - f(A)2^{t_\sigma(B)}$ σ-transitions. Finally, if $\sigma \in \Sigma_{ic}$, C has $(t_\sigma(A)+u_\sigma(A)+1)2^{t_\sigma(B)} - (\tilde{u}_\sigma(A)+1) - f(A)2^{t_\sigma(B)-1}$ σ-transitions. Thus, after some simplifications, the right side of the inequality in the proposition holds. □

Corollary 4.5. *The* isc(L_1L_2) *in the Theorem 4.4 is maximal when* $f(L_1) = 1$.

4.1.2.1 Worst-case Witnesses

In the following we show that the complexity upper bounds found in Theorem 4.4 are tight. As in Section 4.1.1.1, we need to consider three different cases, according to the state and transition complexities of the operands. Although the tight bound for (complete) state complexity can be reached over a binary alphabet [Jir05], all automaton families used in this section have an alphabet $\Sigma = \{a, b, c\}$.

Case 1: $m \geq 2$ and $n \geq 2$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, \{m-1\}\rangle$ with $\delta_A(i, a) = i + 1 \bmod m$, if $i \in [0, m[$, $\delta_A(i, b) = 0$, if $i \in [1, m[$, and $\delta_A(i, c) = i$ if $i \in [0, m[$; and $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\}\rangle$ with $\delta_B(i, a) = i$ if $i \in [0, n[$, $\delta_B(i, b) = i + 1 \bmod n$, if $i \in [0, n[$, and $\delta_B(i, c) = 1$, $i \in [1, n[$. These automata are simple modifications of the ones presented in the proof of the Theorem 2.1 in [YZS94]: a b-transition from the state 0 to itself on DFA A, and a c-transition from the state 0 to the state 1 were eliminated. Both automata are represented in Figure 4.3.

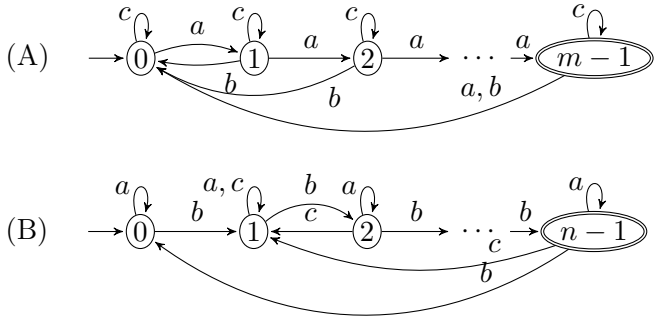


Figure 4.3: DFA A with m states and DFA B with n states.

Theorem 4.6. *For any integers $m \geq 2$ and $n \geq 2$, there exist an m-state DFA A with $r = 3m - 1$ transitions and an n-state DFA B with $s = 3n - 1$ transitions such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ has, at least, $(m + 1)2^n - 2^{n-1} - 1$ states and $(r + 1)2^{\frac{s+1}{3}} + 3.2^{\frac{s-2}{3}} - 5$ transitions.*

Proof. Consider the DFA C such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$ and C is constructed using the concatenation algorithm described above. First we prove the result for the number of states, following the proof of the Theorem 2.1 in [YZS94]. From each $w \in \{a, b\}^\star$, let $S(w) = \{\, i \mid w = w'w''$ such that $w' \in \mathcal{L}(A)$ *and* $i = |w''|_b \bmod n \,\}$, where $|w|_b$ denotes the number of occurrences of the symbol b in the word w. Consider $w, w' \in \{a, b\}^\star$ such that $S(w) \neq S(w')$. Let $k \in S(w) \setminus S(w')$ (or $S(w') \setminus S(w)$). It is clear that $wb^{n-1-k} \in \mathcal{L}(A)\mathcal{L}(B)$ but $w'b^{n-1-k} \notin \mathcal{L}(A)\mathcal{L}(B)$.

For each $w \in \{a, b\}^\star$, define $T(w) = max\{\, |w''| \mid w = w'w''$ *and* $w'' \in a^\star \,\}$. Consider $w, w' \in \{a, b\}^\star$ such that $S(w) = S(w')$ and $T(w) > T(w') \bmod m$. Let $i = T(w) \bmod$

m and $w'' = a^{m-1-i}b^{n-1}$. Therefore $ww'' \in \mathcal{L}(A)\mathcal{L}(B)$, but $w'w'' \notin \mathcal{L}(A)\mathcal{L}(B)$ because it has at least less one a than ww''.

For each subset $s = \{i_1, \ldots, i_t\} \subseteq [0, n[$, where $i_1 > \cdots > i_t$, and an integer $j \in [0, \ldots, m[\cup \{\Omega_A\}$ except the cases where $0 \notin s$ and $j = m-1$, and $s = \emptyset$ and $j = \Omega_B$, there exists a word

$$w = \begin{cases} a^{m-1}b^{i_1} \cdots a^{m-1}b^{i_t}a^j, & \text{if } j \neq \Omega_A; \\ a^{m-1}b^{i_1} \cdots a^{m-1}b^{i_t}b^n, & \text{if } j = \Omega_A; \end{cases}$$

such that $S(w) = s$ and $T(w) = j$, which represents a different left quotient induced by $\mathcal{L}(A)\mathcal{L}(B)$. Thus, C is minimal and has $(m+1)2^n - 2^{n-1} - 1$ states.

Considering, now, the number of transitions. As in the proof of Theorem 4.4, the transitions of C are pairs (θ, γ). Then, C has:

- $(m+1)2^n - 2^{n-1} - 1$, a-transitions. There are $m+1$ θs and 2^n γs, from which we need to remove the transition $(\Omega_A, \emptyset)$. If θ is a transition which leaves a final state of A, γ needs to include the transition that leaves the initial state of B. Thus, 2^{n-1} pairs are removed.
- $(m+1)2^n - 2^{n-1} - 2$, b-transitions. Here, the transition $(\bar{\theta}, \emptyset)$ is removed.
- $(m+1)2^n - 2^{n-1} - 2$, c-transitions. This is analogous to the previous case.

As $m = \frac{r+1}{3}$ and $n = \frac{s+1}{3}$, the DFA C has $(r+1)2^{\frac{s+1}{3}} + 3.2^{\frac{s-2}{3}} - 5$ transitions. □

Case 2: $m = 1$ and $n \geq 2$. Let $A = \langle \{0\}, \Sigma, \delta_A, 0, \{0\} \rangle$ with $\delta_A(0, b) = \delta_A(0, c) = 0$; and $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\} \rangle$ with $\delta_B(i, a) = i$ if $i \in [0, n[$, $\delta_B(i, b) = i + 1 \bmod n$ if $i \in [0, n[$, and $\delta_B(i, c) = i + 1 \bmod n$, if $i \in [1, n[$. The automata A and B are represented in Figure 4.4.

Theorem 4.7. *For any integer $n \geq 2$, there exist a 1-state DFA A with 2 transitions and an n-state DFA B with $s = 3n - 1$ transitions such that any DFA accepting*

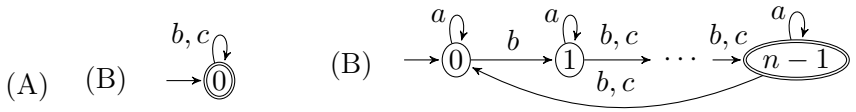


Figure 4.4: DFA A with 1 state and DFA B with n states.

$\mathcal{L}(A)\mathcal{L}(B)$ has, at least, $2^{n+1} - 2^{n-1} - 1$ states and $3(2^{\frac{s+4}{3}} - 2^{\frac{s-2}{3}}) - 4$ transitions.

Proof. Consider the DFA $C = \langle R, \Sigma, \delta, 0, F\rangle$, constructed by the concatenation algorithm previously defined, such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$. One needs to prove that C is minimal, i.e. all states are reachable from the initial state and are pairwise distinguishable. The automaton C has states (q, P) with $q \in \{\Omega_A, 0\}$, $P = \{i_1, \ldots, i_k\}$, $1 \leq k \leq n$, and $i_1 < \cdots < i_k$. There are two kinds of states: final states where $i_k = n-1$; and non-final states where $i_k \neq n-1$. Note that, whenever $q = 0$, we have $i_1 = 0$.

Let f be a final state of the form (q, P), where $P = \{i_1, \ldots, i_{k-1}, n-1\}$ and $\bar{P} = [0, n[\,\backslash P$. Let us construct a word w of size n, such that $\delta(0, w) = f$. We will count the positions (starting with zero) of the word w from the last to the first. If f has $q = \Omega_A$, w has an a in the position i_1; c's in the positions $j \in \bar{P} \setminus \{i_1 - 1\}$ if $i_1 \neq 0$, or $j \in \bar{P}$ otherwise; all the other positions are b's. For example, if $n = 5$, $P = \{4\}$ and $\bar{P} = \{0, 1, 2, 3\}$ then $w = abccc$. If f has $q = 0$ the word has c's in all positions $i_j - 1$, $i_j \in \bar{P}$; all the other positions are $b's$. For example, if $P = \{0, 4\}$, $\bar{P} = \{1, 2, 3\}$ and $n = 5$ then $w = bbccc$. Now, consider the non-final states p which have the same form (q, P), but $i_k \neq n-1$ and $\bar{P} = \{0, \ldots, n-2\} \setminus P$. The word w for these non-final states is constructed with the same rules described above for final states. This proves that all states are reachable from initial state.

Now let us prove that all states are pairwise distinguishable. Final states are trivially distinguishable from non-final states. We need to prove that states of the same kind are distinguishable. Consider $w, w' \in \Sigma^\star$ such that $\delta(0, w) = q$ and $\delta(0, w') = p$, $q \neq p$. Suppose that q and p are final. There are three cases to consider. Let $q = (0, \{0, i_2, \ldots, i_k, n-1\})$ and $p = (0, \{0, j_2, \ldots, j_{k'}, n-1\})$. Suppose $k \geq k'$ and $i \in$

$\{0, i_2, \ldots, i_k, n-1\} \setminus \{0, j_2, \ldots, j_{k'}, n-1\}$. Then $wc^{n-1-i} \in \mathcal{L}(C)$ but $w'c^{n-1-i} \notin \mathcal{L}(C)$. If $q = (\Omega_A, \{i_1, \ldots, i_k, n-1\})$ and $p = (\Omega_A, \{j_1, \ldots, j_{k'}, n-1\})$, we can take i as before and then $wb^{n-1-i} \in \mathcal{L}(C)$ but $w'b^{n-1-i} \notin \mathcal{L}(C)$. If $q = (0, \{0, i_2, \ldots, i_k, n-1\})$ and $p = (\Omega_A, \{j_1, \ldots, j_{k'}, n-1\})$, then $wc^nb^{n-1} \in \mathcal{L}(C)$ but $w'c^nb^{n-1} \notin \mathcal{L}(C)$. Now suppose that q and p are non-final. Let $q = (0, \{0, i_2, \ldots, i_k\})$ and $p = (0, \{0, j_2, \ldots, j_{k'}\})$. Consider, without loss of generality, $k \geq k'$ and $i \in \{0, i_2, \ldots, i_k\} \setminus \{0, j_2, \ldots, j_{k'}\}$. It is clear that $wc^{n-1-i} \in \mathcal{L}(C)$ but $w'c^{n-1-i} \notin \mathcal{L}(C)$. If $q = (\Omega_A, \{i_1, \ldots, i_k\})$ and $p = (\Omega_A, \{j_1, \ldots, j_{k'}\})$, we can take $i \in \{i_1, \ldots, i_k\} \setminus \{j_1, \ldots, j_{k'}\}$ and then $wb^{n-1-i} \in \mathcal{L}(C)$ but $w'b^{n-1-i} \notin \mathcal{L}(C')$. Finally, if $q = (0, \{0, i_2, \ldots, i_k\})$ and $p = (\Omega_A, \{j_1, \ldots, j_{k'}\})$, clearly $wc^nb^{n-1} \in \mathcal{L}(C)$ but $w'c^nb^{n-1} \notin \mathcal{L}(C)$. Thus C is minimal and has $2^{n-2} + 2^{n-1}$ final states and $2^{n-2} + 2^{n-1} - 1$ non-final states. Therefore, it has $2^{n+1} - 2^{n-1} - 1$ states.

The proof for the number of transitions is similar to the proof for the number of transitions of Theorem 4.6. □

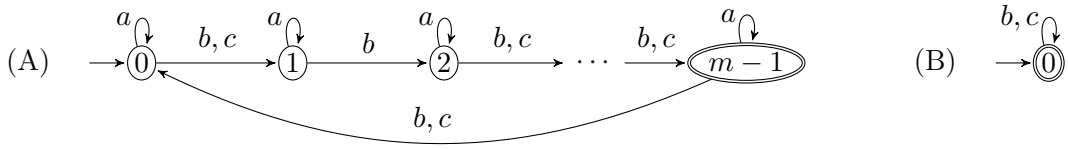


Figure 4.5: DFA A with m states and DFA B with 1 state.

Case 3: $m \geq 2$ and $n = 1$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, \{m-1\}\rangle$ with $\delta_A(i, a) = i$, if $i \in [0, m[$, $\delta_A(i, b) = i + 1 \bmod m$, if $i \in [0, m[$, $\delta_A(i, c) = i + 1 \bmod m$ if $i \in [0, m[\setminus[1]$; and $B = \langle \{0\}, \Sigma, \delta_B, 0, \{0\}\rangle$ with $\delta_B(0, b) = \delta_B(0, c) = 0$. A representation of these DFAs can be seen in Figure 4.5.

Theorem 4.8. *For any integer $m \geq 2$, there exist an m-state DFA A with $r = 3m-1$ transitions and an 1-state DFA B with 2 transitions such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ has at least $2m$ states and $2r$ transitions.*

Proof. Consider the DFA $C = \langle Q, \Sigma, \delta, 0, F\rangle$, such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$, con-

structed with the previous algorithm. We only present the proof for the number of states because the proof for the number of transitions is similar to the proof of Theorem 4.6. By construction we know that C has two kinds of p states:

- final states, which are of the form $(x, \{0\})$ where $x \in [0, m[\cup \{\Omega_A\}$.
- non-final states, which are of the form $(x, \emptyset)$ where $x \in [0, m-2]$.

For any state p we can find a word w for which $\delta(0, w) = p$. If p is a final state of the form $(x, \{0\})$ where $x \in [0, m[$ then $w = b^{m+x}$. In case $x = \Omega_A$ then $w = b^{m+1}c$. Finally, if p is a non-final state then $w = b^x$. Thus, all states are reachable from the initial state. Let us prove that the final states are distinguishable:

- The final states where $x \in [0, m[$ are not equivalent because they correspond to the states of the DFA A which is minimal.
- The final state where $x = \Omega_A$ is not equivalent to the other final state because it is the only final state which is σ-incomplete.

Let $(i, \emptyset)$ and $(j, \emptyset)$ be two distinct non-final states. Consider $w_i, w_j \in \Sigma^\star$ such that $\delta_C(r_0, w_i) = (i, \emptyset)$ and $\delta_C(r_0, w_j) = (j, \emptyset)$. It is clear that $w_i a^{i+1} b^{m-1-i} a^{i+1}$ belongs to $\mathcal{L}(A)\mathcal{L}(B)$ but $w_j a^{i+1} b^{m-1-i} a^{i+1}$ does not. Then w_i and w_j are in different left quotients induced by $\mathcal{L}(A)\mathcal{L}(B)$. Hence, the DFA C is minimal and has $2m$ states. □

4.1.3 Kleene Star

In this section we give a tight upper bound for the incomplete transition complexity of the star operation. The incomplete state complexity of this operation coincides with the one for the complete case.

Let $A = \langle [0, n[\,, \Sigma, \delta, 0, F \rangle$ be a DFA. Consider $F_0 = F \setminus \{0\}$ and suppose that $l = |F_0| \geq 1$. If $F = \{0\}$, then $\mathcal{L}(A)^\star = \mathcal{L}(A)$. The following algorithm constructs a DFA

for the Kleene star of A. Let $A' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ be a new DFA where $q_0' \notin Q$ is a new initial state, $Q' = \{q_0'\} \cup \{P \mid P \subseteq (Q \setminus F_0) \wedge P \neq \emptyset\} \cup \{P \mid P \subseteq Q \wedge 0 \in P \wedge P \cap F_0 \neq \emptyset\}$, $F' = \{q_0'\} \ \cup \{R \mid R \subseteq Q \wedge R \cap F \neq \emptyset\}$, and for $\sigma \in \Sigma$,

$$\delta'(q_0', \sigma) = \begin{cases} \{\delta(0,\sigma)\}, & \text{if } \delta(0,\sigma)\downarrow \ \wedge \delta(0,\sigma) \notin F_0; \\ \{\delta(0,\sigma), 0\}, & \text{if } \delta(0,\sigma)\downarrow \ \wedge \delta(0,\sigma) \in F_0; \\ \emptyset, & \text{if } \delta(0,\sigma)\uparrow; \end{cases}$$

and

$$\delta'(R, \sigma) = \begin{cases} \delta(R,\sigma), & \text{if } \delta(R,\sigma) \cap F_0 = \emptyset; \\ \delta(R,\sigma) \cup \{0\}, & \text{if } \delta(R,\sigma) \cap F_0 \neq \emptyset; \\ \emptyset, & \text{if } \delta(R,\sigma) \ = \emptyset. \end{cases}$$

It is easy to verify that A' recognises the language $\mathcal{L}(A)^\star$. In the following we present the upper bounds for the number of states and transitions for any DFA A' resulting from the algorithm described above.

Theorem 4.9. *For any regular language L, with* $\mathrm{isc}(L) = n$, $s_\sigma = s_\sigma(L)$, *one has* $\mathrm{isc}(L^\star) \leq 2^{n-1} + 2^{n-l-1}$ *and* $\mathrm{itc}(L^\star) \leq |\Sigma|(2^{n-1} + 2^{n-l-1}) + \sum_{\sigma \in \Sigma_i} (s_\sigma - 2^{\tilde{u}_\sigma})$.

Proof. Let A be the minimal DFA that recognises L. Consider the DFA A' such that $\mathcal{L}(A') = \mathcal{L}(A^\star)$ and A' is constructed using the algorithm defined above. Let us prove the result for the $\mathrm{isc}(L^\star)$. Note that Q' is defined as the union of three different sets. The first set contains only the initial state. The states generated by the second set of Q' are the non-empty parts of Q disjoint from F_0. So in this set we have $2^{n-l} - 1$ states (we also remove the empty set). The states in the third set of Q' are the parts of Q that contains the initial state of A and are non-disjoint from F_0. Those are at most $(2^l - 1)2^{n-l-1}$. Therefore the number of states is lesser or equal than $2^{n-1} + 2^{n-l-1}$.

Let us consider the $\mathrm{itc}(L^\star)$. Following the analysis done for the states, the number of σ-transitions of A' is the summation of:

1. s_σ σ-transitions leaving the initial state of A.

2. the number of sets of σ-transitions leaving only non-final states of A:

   (a) $(2^{t_\sigma - l}) - 1$, if A is σ-complete, because we have $t_\sigma - l$ σ-transitions of this kind, and we remove the empty set;

   (b) $2^{t_\sigma - l + u_\sigma} - 2^{\tilde{u}_\sigma}$, if A is σ-incomplete because we have $t_\sigma - l + u_\sigma$ of this kind, and we subtract the number of sets with only undefined σ-transitions of A.

3. the number of sets of σ-transitions leaving final and non-final states of A. We do not count the transition leaving the initial state of A because, by construction, if a transition of A' contains a transition leaving a final state of A then it also contains the one leaving the initial state of A. Thus, we have

   (a) $(2^l - 1)2^{t_\sigma - l - 1}$, if A is σ-complete;

   (b) $(2^l - 1)2^{t_\sigma - l - 1 + u_\sigma}$, if A is σ-incomplete.

Thus, the inequality in the proposition holds. □

Corollary 4.10. *The* $\mathrm{isc}(L^\star)$ *presented in Theorem 4.9 is maximal when* $l = 1$.

4.1.3.1 Worst-case Witnesses

Let us present an automaton family, with $\Sigma = \{a, b\}$, for which the upper bounds in Theorem 4.9 are reached.

Define $A = ([0, n[, \Sigma, \delta_A, 0, \{n-1\})$ with $\delta_A(i, a) = i + 1 \bmod n$ for $i \in [0, n[$, and $\delta_A(i, b) = i + 1 \bmod n$ for $i \in [1, n[$. This DFA is depicted in Figure4.6.

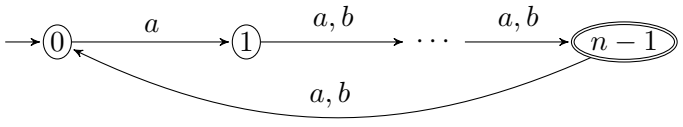


Figure 4.6: DFA A with n states.

Theorem 4.11. *For any integer $n \geq 2$, there exist an n-state DFA A with $r = 2n-1$ transitions such that any DFA accepting $\mathcal{L}(A)^\star$ has, at least, $2^{n-1} + 2^{n-2}$ states and $2^{\frac{r+1}{2}} + 2^{\frac{r-1}{2}} - 2$ transitions.*

Proof. For $n = 2$ it is clear that $\mathcal{L} = \{w \in \{a,b\}^\star \mid |w|_a \text{ is odd}\}$ is accepted by a two-state DFA, and $\mathcal{L}^\star = \{\varepsilon\} \cap \{w \in \{a,b\}^\star \mid |w|_a \geq 1\}$ cannot be accepted with less than 3 states. For $n > 2$, we consider the automaton family A which is shown in Figure 4.6. Consider the DFA A' such that $\mathcal{L}(A') = \mathcal{L}(A^\star)$. First we prove the result for the number of states, following the proof of the Theorem 3.3 in [YZS94]. In order to prove that A' is minimal, thus we need to prove the following.

- Every state is reachable from the start state. As each state of A' is a subset of states of A, we proceed by induction on the size of these states. If $|q| = 1$ we have:

$$q = \begin{cases} \{1\} = \delta'(q'_0, a); & (4.1) \\ \{i\} = \delta'(\{i-1\}, a), \qquad \text{for } 1 < i < n-1; & (4.2) \\ \{0\} = \delta'(\{n-1, 0\}, b). & (4.3) \end{cases}$$

  Note that we reach $q = \{0\}$ from a state with size two, but we reach the state $\{n-1, 0\}$ by $\delta'(\{n-2\}, a)$ and $\{n-2\}$ is already considered in (4.2). Thus we can reach all states such that $|q| = 1$. Now, assume that, for every state q, if $|q| < m$ then q is reachable. Let us prove that if $|q| = m$ then it is also reachable. Consider $q = \{i_1, i_2, \ldots, i_m\}$ such that $0 \leq i_1 < i_2 < \cdots < i_m < n-1$ if $n-1 \notin q$, $0 = i_1 < i_2 < \cdots < i_{m-1} < i_m = n-1$ otherwise. There are three cases to consider:

  (i) $\{n-1, 0, i_3, \ldots, i_m\} = \delta'(\{n-2, i_3-1, \ldots, i_m-1\}, a)$ where the state $\{n-2, i_3-1, \ldots, i_m-1\}$ contains $m-1$ states.

  (ii) $\{0, 1, i_3, \ldots, i_m\} = \delta'(\{n-1, 0, i_3-1, \ldots, i_m-1\}, a)$ where the state $\{n-1, 0, i_3-1, \ldots, i_m-1\}$ is considered in case (i).

(iii) $\{t, i_2, \ldots, i_m\} = \delta'(\{0, i_2 - t, \ldots, i_m - t\}, a^t)$, $t > 0$, where the state $\{0, i_2 - t, \ldots, i_m - t\}$ is considered in case (ii).

- Each state defines a different left quotient induced by $\mathcal{L}(A')$. Consider p, $q \in Q'$, $p \neq q$ and $i \in p \setminus q$. Then $\delta'(p, a^{n-1-i}) \in F'$ but $\delta'(q, a^{n-1-i}) \notin F'$.

Let us consider, now, the problem of the number of transitions. The DFA A' has:

- $2^{n-1} + 2^{n-2}$ a–transitions because it has one a–transition which corresponds to s_a, $2^{n-1} - 1$ a–transitions which corresponds to case 2. of Theorem 4.9 and 2^{n-2} a–transitions which corresponds to case 3. of Theorem 4.9.
- $2^{n-1} - 2 + 2^{n-2}$ b–transitions because it has $2^{n-2+1} - 2$ b–transitions which corresponds to case 2. of Theorem 4.9, and 2^{n-3+1} b–transitions which corresponds to case 3. of Theorem 4.9.

As $n = \frac{r+1}{2}$, A' has $2^{\frac{r+1}{2}} + 2^{\frac{r-1}{2}} - 2$ transitions. □

4.1.4 Reversal

It is known that when considering complete DFAs the state complexity of the reversal operation reaches the upper bound 2^n, where n is the state complexity of the operand language. By the subset construction, a (complete) DFA resulting from the reversal has a state which corresponds to $\emptyset$, which is a dead state. Therefore, if we remove that state the resulting automaton is not complete and the incomplete state complexity is $2^n - 1$. Consequently the transition complexity is $|\Sigma|(2^n - 1)$. It is easy to see that the worst case of the reversal operation is reached when the operand is complete.

4.1.5 Unary Languages

In the case of unary languages, if a DFA is not complete it represents a finite language. Thus, the worst-case state complexity of operations occurs when the operand DFAs are

complete. For these languages the (incomplete) transition complexity coincide with the (incomplete) state complexity. The study for union and intersection was made by Y. Gao et al. [GSY11], and using similar methods, it is not difficult to obtain the corresponding results for the other operations addressed in this article.

4.1.6 Experimental Results

Hitherto we studied the descriptional complexity of several operations considering the worst-case analysis. However, for practical applications, it is important to know how significant are these worst-case results, i.e. if these upper bounds are reached for a significant number of cases or, on the contrary, only rarely occur. To evaluate this, we performed some experimental tests in order to analyse how often the upper bounds were, in practice, achieved. Although we fixed the size of the alphabet and considered small values of n and m, the experiments are statistically significant and provide valuable information about the average case behaviour of these operations.

Almeida et al. [AMR07] presented an uniform random generator for complete DFAs. We can use this generator to obtain incomplete DFAs, if we consider the existence of a dead state. However, in this case, the probability that a state has a transition to the dead state is $\frac{1}{n+1}$, where n is the number of useful states of the generated incomplete DFA. Although this corresponds to a uniform distribution, for very large values of n, the referred probability is very low, and thus the generated DFAs are almost always complete. Therefore, in order to generate random incomplete DFAs, we can increase the number of void transitions in the generated DFAs to change the referred probability. For that, the generator accepts a parameter b that defines the multiplicity of dead states. Using b ($0 < b < 1$), we compute the integer part of $m = \frac{b \times n}{1-b}$, which indicates the number of dead states in the generated DFA. Note that the generated DFA becomes "more incomplete" when b tends to 1.

All the tests were performed using the random generator described above. The tests

and the generator were implemented in Python[1] using the FAdo system, and are both publicly available[2]. In the following experiments (Table 4.4) we consider $b = 0.7$.

As the DFAs were obtained with a uniform random generator, the size of each sample (20000 elements) is sufficient to ensure a 95% confidence level within a 1% error margin. Table 4.4 shows the results of experimental tests with 20000 pairs of incomplete DFAs as operands. We present the results for operands with $m, n \in \{2, 4, 6, 8, 10, 12, 14, 16, 18\}$ states, such that $m + n = 20$, over an alphabet of $k = 5$ symbols. As union and intersection are symmetric operations, we only present the results for $m \in \{10, 12, 14, 16, 18\}$ and $n \in \{10, 8, 6, 4, 2\}$. We considered the following measures for the DFA resulting from the operation: the state and transition complexity, sc and tc, respectively; the upper bounds for these measures, ubsc and ubtc, respectively; its transition density $d = \frac{\text{tc}}{k \cdot \text{sc}}$; and the ratios $\text{rs} = \frac{\text{sc}}{\text{ubsc}}$ and $\text{rt} = \frac{\text{tc}}{\text{ubtc}}$. Note that the results presented in this table are averages, i.e. we calculate all the referred measures for each pair of operands and then we compute the average of each measure. The columns labeled m_1, m_2, m_3 and m_4 give the maximal values of sc, ubsc, tc and ubtc, respectively. For example, considering $m = 10$ and $n = 10$ we calculate the ubsc for the concatenation of each pair of random incomplete DFAs. Then we do the average of the 20000 obtained values and the result is 8557.90, as we can see in the table. We need to do this because every measure depends of parameters that can be different in each pair of generated DFAs.

As it was expected, for the complement operation, the upper bound for the state complexity was always reached on the experiments. For all the other operations the number of states of the DFA obtained during the experimentation (sc) was much lower than the upper bounds. For example, for $m = 10$ and $n = 10$ the upper bound was 150 times larger than the number of states of the DFA resulting from the concatenation in the experiment. Even the largest DFA obtained during the experimentation has less

[1] http://www.python.org

[2] The code used to performed the tests is available at `http://khilas.dcc.fc.up.pt/~eva/` and the necessary library to perform the tests, including the referred DFA generator, can be obtained at http://fado.dcc.fc.up.pt.

Table 4.4: Experimental results for regular languages with $b = 0.7$.

b=0.7												
Concatenation												
m	n	sc	$ubsc$	rs	m_1	m_2	tc	$ubtc$	rt	m_3	m_4	d
2	18	54.2	604404.76	0.00009	416	655359	182.90	3141223.07	0.00006	1929	3792832	0.62
4	16	55.85	253077.73	0.0002	430	294911	190.69	1316341.55	0.0001	1962	1533056	0.64
6	14	59.81	88087.17	0.0007	303	106495	210.30	468266.14	0.0004	1377	537856	0.67
8	12	59.11	28115.99	0.002	431	34815	210.50	151521.51	0.001	1928	173280	0.68
10	10	54.79	8557.90	0.01	295	10751	194.56	46208.83	0.004	1378	53300	0.68
12	8	50.72	2523.72	0.02	219	3199	180.17	13481.26	0.01	1001	15568	0.69
14	6	44.73	725.28	0.06	179	927	156.79	3760.56	0.04	750	4336	0.68
16	4	36.35	204.44	0.18	117	263	121.18	1002.60	0.12	481	1171	0.65
18	2	28.16	56.31	0.50	54	71	88.10	250.02	0.35	231	289	0.62
Union												
10	10	33.08	120	0.28	89	120	90.46	378.97	0.24	346	480	0.53
12	8	33.33	116	0.29	89	116	91.87	367.46	0.25	323	463	0.53
14	6	32.38	104	0.31	90	104	88.74	326.77	0.27	336	414	0.53
16	4	29.96	84	0.36	73	84	79.87	255.68	0.31	283	340	0.52
18	2	27.84	56	0.50	55	56	73.23	162.12	0.45	209	225	0.51
Intersection												
10	10	7.98	100	0.08	59	100	9.74	46208.83	0.0002	120	53300	0.19
12	8	8.18	96	0.09	60	96	10.09	445.26	0.02	109	825	0.19
14	6	7.78	84	0.09	56	84	9.58	389.08	0.02	101	722	0.18
16	4	6.61	64	0.10	52	64	7.93	283.61	0.03	99	624	0.17
18	2	6.03	36	0.17	34	36	7.45	155.84	0.05	70	396	0.17
Star												
2		2.07	3.23	0.64	3	4	5.22	8.73	0.60	15	19	0.50
4		4.64	10.72	0.43	12	16	13.96	40.72	0.34	51	74	0.55
6		8.79	38.20	0.23	31	64	30.55	170.63	0.18	136	302	0.68
8		14.39	141.73	0.10	74	256	53.93	676.34	0.08	333	1219	0.73
10		21.61	542.92	0.040	113	1024	85.40	2662.98	0.03	493	4987	0.77
12		30.98	2118.42	0.015	156	4096	127.16	10510.73	0.01	723	19620	0.80
14		41.10	8346.26	0.005	226	12288	173.13	41603.90	0.004	1115	60981	0.83
16		53.20	33113.56	0.002	263	49152	228.74	165364.25	0.001	1209	244731	0.85
18		68.04	131851.28	0.001	304	196608	298.15	658938.51	0.0004	1466	974212	0.87
Reversal												
2		2.43	3	0.81	3	3	5.28	15	0.35	13	15	0.42
4		6.46	15	0.43	15	15	16.48	75	0.22	63	75	0.49
6		12.18	63	0.19	48	63	34.63	315	0.11	181	315	0.54
8		18.72	255	0.07	105	255	55.43	1275	0.043	468	1275	0.56
10		26.46	1023	0.0259	129	1023	80.79	5115	0.0158	536	5115	0.58
12		36.08	4095	0.0088	187	4095	113.74	20475	0.0056	804	20475	0.60
14		45.94	16383	0.0028	224	16383	146.93	81915	0.0018	989	81915	0.61
16		57.05	65535	0.0009	353	65535	185.02	327675	0.0006	1504	327675	0.62
18		70.55	262143	0.0003	337	262143	232.92	1310715	0.0002	1476	1310715	0.63
Complement												
2		3	3	1	3	3	9.62	29.79	0.32	15	55	0.64
4		5	5	1	5	5	22.11	50.81	0.44	25	85	0.88
6		7	8	1	7	7	33.91	73.84	0.46	35	120	0.97
8		9	9	1	9	9	44.61	95.35	0.47	45	155	0.99
10		11	11	1	11	11	54.87	116.90	0.47	55	175	1.00
12		13	13	1	13	13	64.94	140.16	0.46	65	205	1.00
14		15	15	1	15	15	74.99	162.05	0.46	75	235	1.00
16		17	17	1	17	17	85	183.96	0.46	85	265	1.00
18		19	19	1	19	19	95	207.13	0.46	95	280	1.00

states than what was expected in the worst case. Considering the same example, the largest DFA has 295 states and the upper bound is 8557.90. Nevertheless, for binary operations, whenever the difference between m and n increase, the number of states of the DFA resulting from the operations, in the experiment, was closer to the upper bound. For Kleene star and reversal operations, the upper bound was far from being reached. For $m = 18$ the upper bound for Kleene star was 1900 times larger than the number of states of the resulting DFA. Note that the DFAs resulting from all the operations in the experimentation (excluding the complement) were also incomplete.

The experimental results for the transition complexity were very similar to the previous ones. For the union, the difference was not so notorious, but for all the other operations it was very high, mainly for the Kleene star and the reversal operations. For example, considering $m = 10$ and $n = 10$, for union, the upper bound was only 4 times larger than the number of transitions of the resulting DFA. However, for the concatenation, the upper bound was 1300 times larger. For $m = 18$ the upper bound for reversal was 5600 times larger than the number of transitions of the resulting DFA. Note that, although the DFA resulting from the complement was complete, the upper bound for the transition complexity was much higher than the number of transitions of that DFA. This happens because Gao et al. chose to give an upper bound as a function of the transition complexity of the operand, and because of this the upper bound, in some situations, is greater than the $|\Sigma|(m+1)$, which is the maximal number of transitions of any DFA with $m+1$ states.

Although this sample was made for few values of n and m, we expect that the experimental results for other cases would be very similar. Thus, we can conjecture that the upper bounds for all operations studied are excessively pessimistic, when considering practical applications.

Table 4.5: State complexity of basic regularity preserving operations on finite languages.

Operation	isc	sc
$L_1 \cup L_2$	$mn-2$	$mn-(m+n)$
$L_1 \cap L_2$	$mn-2m-2n+6$	$mn-3(m+n)+12$
L^C	$m+1$	m
L_1L_2	$\sum_{i=0}^{m-1} \min\left\{k^i, \sum_{j=0}^{f(A,i)} \binom{n-1}{j}\right\} + \sum_{j=0}^{f(A)} \binom{n-1}{j} - 1$	$\sum_{i=0}^{m-2} \min\left\{k^i, \sum_{j=0}^{f(A,i)} \binom{n-2}{j}\right\} + \sum_{j=0}^{f(A)} \binom{n-2}{j}$
$L^\star$	$2^{m-f(A)-1}+2^{m-2}-1$	$2^{m-f(A)-2}+2^{m-3}$
L^R	$\sum_{i=0}^{l-1} k^i + 2^{m-l} - 1$	$\sum_{i=0}^{l-1} k^i + 2^{m-l-1}$

4.2 Finite Languages

In this section we give tight upper bounds for the state and transition complexity of all the operations considered in the last section, for incomplete DFAs representing finite languages, with an alphabet size greater than 1. Note that, for unary finite languages the incomplete transition complexity is equal to the incomplete state complexity of that language, which is always equal to the state complexity of the language minus one. For the concatenation, we correct the upper bound for the state complexity of complete DFAs [CCSY01], and show that if the *right* automaton is larger than the *left* one, the upper bound is only reached using an alphabet of variable size. In the Tables 4.5 and 4.6 we summarise the results of these section and the tight upper bounds for the state complexity on complete DFAs. As in the previous section, we also present some experimental results in order to compare the worst case with the average case for these operations.

Let A be a minimal DFA with n states accepting a finite language, where the states are assumed to be topologically ordered, *i.e.*, $p' = \delta(p,\sigma)$ implies that $p' > p$. We will denote by $\mathrm{in}_\sigma(A,i)$ the number of transitions reaching i, and omit argument A whenever there is no ambiguity. Then, $\sum_{\sigma\in\Sigma} \mathrm{in}_\sigma(0) = 0$ and there is exactly one final state which, because of the topological order is $n-1$, called *pre-dead*, such that

Table 4.6: Transition complexity of basic regularity preserving operations on finite languages.

Operation	itc
$L_1 \cup L_2$	$\sum_{\sigma\in\Sigma}(s_\sigma(L_1) \boxplus s_\sigma(L_2) - (\mathrm{itc}_\sigma(L_1) - s_\sigma(L_1))(\mathrm{itc}_\sigma(L_2) - s_\sigma(L_2))) + n(\mathrm{itc}(L_1) - s(L_1)) + m(\mathrm{itc}(L_2) - s(L_2))$
$L_1 \cap L_2$	$\sum_{\sigma\in\Sigma}(s_\sigma(L_1)s_\sigma(L_2) + (\mathrm{itc}_\sigma(L_1) - s_\sigma(L_1) - a_\sigma(L_1))(\mathrm{itc}_\sigma(L_2) - s_\sigma(L_2) - a_\sigma(L_2)) + a_\sigma(L_1)a_\sigma(L_2))$
L^C	$\lvert\Sigma\rvert(m+1)$
L_1L_2	$k\sum_{i=0}^{m-2}\min\left\{k^i, \sum_{j=0}^{f(L_1,i)}\binom{n-1}{j}\right\} + + \sum_{\sigma\in\Sigma}\left(\min\left\{k^{m-1} - \overline{s}_\sigma(L_2), \sum_{j=0}^{f(L_1)-1}\Delta_j\right\} + \sum_{j=0}^{f(L_1)}\Lambda_j\right)$
$L^\star$	$2^{m-f(L)-1}\left(k + \sum_{\sigma\in\Sigma}2^{e_\sigma(L)}\right) - \sum_{\sigma\in\Sigma}2^{\overline{t}_\sigma(L)-\overline{s}_\sigma(L)-\overline{e}_\sigma(L)} - \sum_{\sigma\in X}2^{\overline{t}_\sigma(L)-\overline{s}_\sigma(L)-\overline{e}_\sigma(L)}$
L^R	$\sum_{i=0}^{l}k^i - 1 + k2^{m-l} - \sum_{\sigma\in\Sigma}2^{\sum_{i=0}^{l-1}\overline{t}_\sigma(L,i)+1}$, m even
	$\sum_{i=0}^{l}k^i - 1 + k2^{m-l} - \sum_{\sigma\in\Sigma}\left(2^{\sum_{i=0}^{l-2}\overline{t}_\sigma(L,i)+1} - c_\sigma(l)\right)$, m odd

$\sum_{\sigma\in\Sigma}t_\sigma(n-1) = 0$. The *level* of a state i is the length of the shortest path from the initial state to i which never exceeds $n-1$. The level of A is the level of its pre-dead state. A DFA is called *linear* if its level is $n-1$.

4.2.1 Union

Consider the algorithm for the union operation based on the usual product construction already defined in the Section 4.1.1. Let $t_\sigma([k,l]) = \sum_{i\in[k,l]}t_\sigma(i)$. The following theorem presents the upper bounds for the number of states and transitions of any DFA accepting the union of two finite languages. Note that the result for the number of states is similar to the one for the complete case, omitting the dead state.

Theorem 4.12. *For any two finite languages L_1 and L_2 with* $\mathrm{isc}(L_1) = m$ *and* $\mathrm{isc}(L_2) = n$, *one has* $\mathrm{isc}(L_1 \cup L_2) \leq mn - 2$ *and*

$$\begin{aligned}\mathrm{itc}(L_1 \cup L_2) \leq &\sum_{\sigma\in\Sigma}(s_\sigma(L_1) \boxplus s_\sigma(L_2) - (\mathrm{itc}_\sigma(L_1) - s_\sigma(L_1))(\mathrm{itc}_\sigma(L_2) - s_\sigma(L_2))) \\ &+ n(\mathrm{itc}(L_1) - s(L_1)) + m(\mathrm{itc}(L_2) - s(L_2)),\end{aligned}$$

where for x, y Boolean values, $x \boxplus y = \min(x + y, 1)$.

Proof. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ and $B = \langle [0, n[, \Sigma, \delta_B, 0, F_B\rangle$ be the minimal DFAs that recognise L_1 and L_2, respectively. Let us consider, first, the counting of the number of states. In the product automaton, the set of states is a subset of $([0, m[\cup \{\Omega_A\}) \times ([0, n[\cup \{\Omega_B\})$. The states of the form $(0, i)$, where $i \in [\,1, n[\cup \{\Omega_B\}$, and of the form $(j, 0)$, where $j \in [1, m[\cup \{\Omega_A\}$, are not reachable from $(0, 0)$ because the operands represent finite languages; the states $(m-1, n-1)$, $(m-1, \Omega_B)$ and $(\Omega_A, n-1)$ are equivalent because they are final and they do not have out-transitions; the state (Ω_A, Ω_B) is the dead state and because we are dealing with incomplete DFAs we can ignore it. Therefore the number of states of the union of two incomplete DFAs accepting finite languages is at most $(m+1)(n+1) - (m+n) - 2 - 1 = mn - 2$.

Consider the number of transitions. In the product automaton, the σ-transitions can be represented as pairs (α_i, β_j) where α_i (respectively β_j) is 0 if there exists a σ-transition leaving the state i (respectively j) of DFA A (respectively B), or -1 otherwise. The resulting DFA can have neither transitions of the form $(-1, -1)$, nor of the form (α_0, β_j), where $j \in [\,1, n[\cup \{\Omega_B\}$, nor of the form (α_i, β_0), where $i \in [\,1, m[\cup \{\Omega_A\}$, as happened in the case of states. Thus, the number of σ-transitions for $\sigma \in \Sigma$ are:

$$\begin{aligned} s_\sigma(A) \boxplus s_\sigma(B) + t_\sigma(A, [1, m[\,) t_\sigma(B, [1, n[\,) + t_\sigma(A, [1, m[\,)(\bar{t}_\sigma(B, [1, n[\,) + 1) \\ + (\bar{t}_\sigma(A, [1, m[\,) + 1) t_\sigma(B, [1, n[\,) \; = \\ s_\sigma(A) \boxplus s_\sigma(B) + n t_\sigma(A, [1, m[\,) + m t_\sigma(B, [1, n[\,) - t_\sigma(A, [1, m[\,) t_\sigma(B, [1, n[\,). \end{aligned}$$

Because the DFAs are minimal, $\sum_{\sigma \in \Sigma} t_\sigma(A, [1, m[\,)$ corresponds to $\mathrm{itc}(L_1) - s(L_1)$, and analogously for B. Therefore the theorem holds. □

4.2.1.1 Worst-case Witnesses

In the following we show that the upper bounds described above are tight. Han & Salomaa proved [HS08, Lemma 3] that the upper bound for the number of states

can not be reached for any alphabet with a fixed size. The witness families for the incomplete complexities coincide with the ones that these authors presented for the state complexity. As we are not including the dead state, our representation is slightly different. Let $m, n \geq 1$ and $\Sigma = \{b, c\} \cup \{a_{ij} \mid i \in [\,1, m[, j \in [1, n[,\ (i,j) \neq (m-1, n-1)\}$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, \{m-1\}\rangle$ where $\delta_A(i, b) = i + 1$ for $i \in [0, m-2]$ and $\delta_A(0, a_{ij}) = i$ for $j \in [1, n[,\ (i,j) \neq (m-1, n-1)$. Let $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\}\rangle$, where $\delta_B(i, c) = i + 1$ for $i \in [0, n[$ and $\delta_B(0, a_{i,j}) = j$ for $j \in [1, n[, i \in [1, m[,\ (i,j) \neq (m-1, n-1)$. See Figure 4.7 for the case $m = 5$ and $n = 4$.

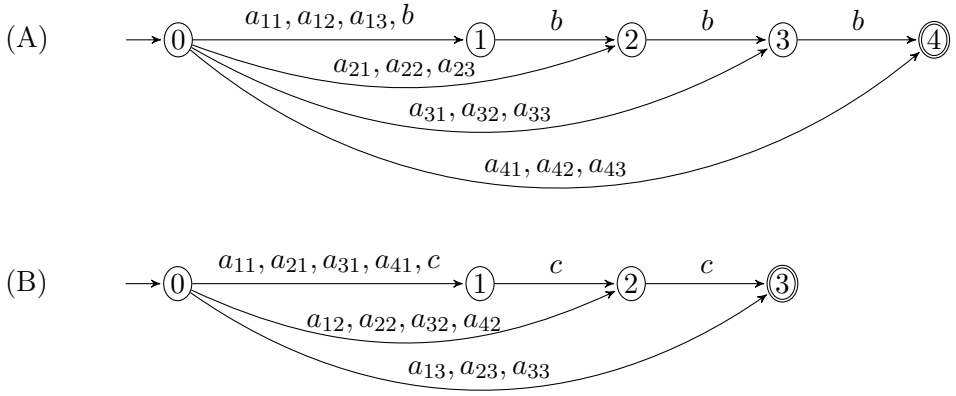


Figure 4.7: DFA A with $m = 5$ and DFA B with $n = 4$.

Theorem 4.13. *For any two integers $m \geq 2$ and $n \geq 2$, there exist an m-state DFA A and an n-state DFA B, both accepting finite languages, such that any DFA accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$ needs at least $mn - 2$ states and $3(mn - n - m) + 2$ transitions, with an alphabet of size depending on m and n.*

Proof. The proof for the number of states is the same as the proof of [HS08, Lemma 2], considering the language families above. Let us prove the result for the number of transitions. The DFA A has $m - 1$ b-transitions and one a_{ij}-transition, for each a_{ij}. The DFA B has $n - 1$ c-transitions and the same number of a_{ij}-transitions as A. Thus, the DFA resulting for the union operation has:

- $mn - 2n + 1$ b-transitions;
- $mn - 2n + 1$ c-transitions;

- one a_{ij}-transitions for each a_{ij} and there are $mn - n - m$ different a_{ij}.

Thus, the total number of transitions is $3(mn - n - m) + 2$. It is easy to prove that the resulting DFA is minimal. □

4.2.2 Intersection

Given two DFAs $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ and $B = \langle [0, n[, \Sigma, \delta_B, 0, F_B\rangle$, a DFA accepting $\mathcal{L}(A) \cap \mathcal{L}(B)$ can be also obtained by the product construction. Once more, the result for the state complexity is similar to the one for the complete case, omitting the dead state. Let $a_\sigma(A) = \sum_{i\in F} \mathrm{in}_\sigma(A, i)$, and $a(L) = \sum_{\sigma\in\Sigma} a_\sigma(L)$.

Theorem 4.14. *For any two finite languages L_1 and L_2 with* $\mathrm{isc}(L_1) = m$ *and* $\mathrm{isc}(L_2) = n$*, one has* $\mathrm{isc}(L_1 \cap L_2) \leq mn - 2m - 2n + 6$ *and*

$$\mathrm{itc}(L_1 \cap L_2) \leq \sum_{\sigma\in\Sigma} (s_\sigma(L_1)s_\sigma(L_2) + (\mathrm{itc}_\sigma(L_1) - s_\sigma(L_1) - a_\sigma(L_1))(\mathrm{itc}_\sigma(L_2) - s_\sigma(L_2) - a_\sigma(L_2)) + a_\sigma(L_1)a_\sigma(L_2))\,.$$

Proof. Let A and B be the minimal DFAs that recognise L_1 and L_2, respectively. Consider the DFA accepting $\mathcal{L}(A) \cap \mathcal{L}(B)$ obtained by the product construction. Let us prove the result for $\mathrm{isc}(L_1 \cap L_2)$. For the same reasons as in Theorem 4.12, we can eliminate the states of the form $(0, j)$, where $j \in [\,1, n[\cup \{\Omega_B\}$, and of the form $(i, 0)$, where $i \in [\,1, m[\cup \{\Omega_A\}$; the states of the form $(m - 1, j)$, where $j \in [1, n - 2]$, and of the form $(i, n - 1)$, where $i \in [1, m - 2]$ are equivalent to the state $(m - 1, n - 1)$ or to the state (Ω_A, Ω_B); the states of the form (Ω_A, j), where $j \in [1, n[\cup \{\Omega_B\}$, and of the form (i, Ω_B), where $i \in [1, m[\cup \{\Omega_A\}$ are equivalent to the state (Ω_A, Ω_B) which is the dead state of the DFA resulting from the intersection, and thus can be removed. Therefore, the number of states is at most $(m+1)(n+1) - 3((m+1)(n+1)) + 12 - 1 = mn - 2(m + n) + 6$.

Let us consider the $\mathrm{itc}(L_1 \cap L_2)$. Using the same technique as in Theorem 4.12 and considering that in the intersection we only have pairs of transitions where both

elements are different from -1, the number of σ-transitions is as follows, which proves the theorem,

$$s_\sigma(A)s_\sigma(B) + (t_\sigma(A, [1, m[\,) \setminus \mathrm{in}_\sigma(A, F_A))(t_\sigma(B, [1, n[\,) \setminus \mathrm{in}_\sigma(B, F_B)) + a_\sigma(A)a_\sigma(B).$$

□

4.2.2.1 Worst-case Witnesses

The next result shows that the complexity upper bounds found above are reachable. The witness languages for the tightness of the bounds for this operation are different from the families given by Han & Salomaa, because those families are not tight for the transition complexity. For $m \geq 2$ and $n \geq 2$, let $\Sigma = \{a_{ij} \mid i \in [1, m-2],\ j \in [1, n-2]\} \cup \{a_{ij} \mid i = m-1,\ j = n-1\}$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, \{m-1\}\rangle$ where $\delta_A(x, a_{ij}) = x + i$ for $x \in [\,0, m[,\ i \in [1, m-2]$, and $j \in [1, n-2]$, and let $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\}\rangle$ where $\delta_B(x, a_{ij}) = x + j$ for $x \in [0, n[,\ i \in [1, m-2]$, and $j \in [1, n-2]$. The new families are presented in Figure 4.8 for $m = 5$ and $n = 4$.

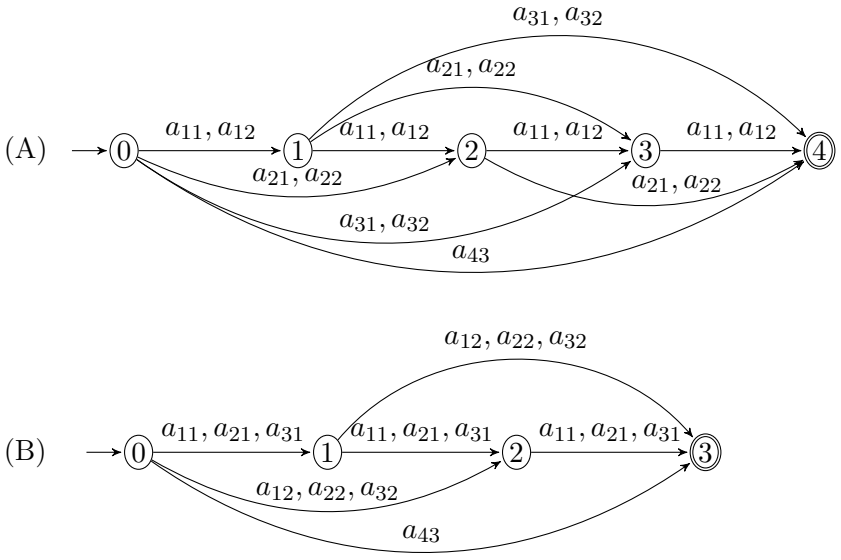


Figure 4.8: DFA A with $m = 5$ and DFA B with $n = 4$.

Theorem 4.15. *For any two integers $m \geq 2$ and $n \geq 2$, there exist an m-state DFA A and an n-state DFA B, both accepting finite languages, such that any DFA accepting $\mathcal{L}(A) \cap \mathcal{L}(B)$ needs at least $mn - 2(m+n) + 6$ states and $(m-2)(n-2)(2 + \sum_{i=1}^{\min(m,n)-3}(m-2-i)(n-2-i)) + 2$ transitions, with an alphabet of size depending on m and n.*

Proof. To prove that the minimal DFA accepting $L(A) \cap L(B)$ needs $mn - 2m - 2n + 6$ states we can use the same technique which is used in the proof of [HS08, Lemma 6]. For that, we define a set R of words which are not equivalent under $\equiv_{L(A)\cap L(B)}$. Let ε be the null string. We choose $R = R_1 \cup R_2 \cup R_3$, where $R_1 = \{\varepsilon\}$, $R_2 = \{a_{ij} \mid i = m-1,\ j = n-1\}$, and $R_3 = \{a_{ij} \mid i \in [1, m-2] \text{ and } j \in [1, n-2]\}$. It is easy to see that all words of each set are not equivalent to each other. As $|R_1| = |R_2| = 1$ and $|R_3| = (m-2)(n-2)$, we have that $|R| = mn - 2m - 2n + 6$. Thus the result for the number of states holds.

Let us consider the number of transitions. The DFA A has $(n-2)\sum_{i=0}^{m-3}(m-1-i)+1$ a_{ij}- transitions. The DFA B has $(m-2)\sum_{i=0}^{n-3}(n-1-i)+1$ a_{ij}- transitions. Let $k = (m-2)(n-2)+1$. As in proof of Theorem 4.14, the DFA resulting from the intersection operation has the following number of transitions:

- k, corresponding to the pairs of transitions leaving the initial states of the operands;
- $\sum_{i=1}^{\min(m,n)-3}(n-2)(m-2-i)(m-2)(n-2-i)$, corresponding to the pairs of transitions formed by transitions leaving non-final and non-initial states of the operands;
- k, corresponding to the pairs of transitions leaving the final states of the operands.

Thus the total number of transitions is $2k + (m-2)(n-2)\sum_{i=1}^{\min(m,n)-3}(m-2-i)(n-2-i)$. □

4.2.3 Complement

The state and transition complexity for this operation on finite languages are similar to the ones on regular languages [GSY11]. This happens because the DFA must be completed. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ be a DFA accepting the language L. The complement of L, L^c, is recognised by the DFA $C = \langle [0, m[\cup \{\Omega_A\}, \Sigma, \delta_C, 0, ([0, m[\setminus F_A) \cup \{\Omega_A\}\rangle$, where for $\sigma \in \Sigma$ and $i \in [0, m[$, $\delta_C(i, \sigma) = \delta_A(i, \sigma)$ if $\delta_A(i, \sigma) \uparrow$; $\delta_A(i, \sigma) = \Omega_A$ otherwise. Therefore one has,

Theorem 4.16. *For any finite language L with* $\mathrm{isc}(L) = m$ *one has* $\mathrm{isc}(L^C) \leq m + 1$ *and* $\mathrm{itc}(L^C) \leq |\Sigma|(m + 1)$.

Proof. Concerning the $\mathrm{isc}(L^C)$, it is only necessary to add a dead state to the operand DFA. The maximal number of σ-transitions is $m + 1$, because this is the number of states. Thus, the maximal number of transitions is $|\Sigma|(m + 1)$. □

Gao et al. [GSY11] gave the value $|\Sigma|(\mathrm{itc}(L) + 2)$ for the transition complexity of the complement. In some situations, this bound is higher than the bound here presented, but contrasting to that one, it gives the transition complexity of the operation as a function of the transition complexity of the operand.

The witness family for this operation is exactly the same presented in [GSY11], i.e. $\{b^m\}$, for $m \geq 1$. It is easy to see that the bounds are tight for this family.

4.2.4 Concatenation

Câmpeanu et al. [CCSY01] studied the state complexity of the concatenation of an m-state complete DFA A with an n-state complete DFA B over an alphabet of size k and proposed the upper bound

$$\sum_{i=0}^{m-2} \min\left\{k^i, \sum_{j=0}^{f(A,i)} \binom{n-2}{j}\right\} + \min\left\{k^{m-1}, \sum_{j=0}^{f(A)} \binom{n-2}{j}\right\}, \tag{4.4}$$

where $f(A,i)$ is the larger number of final states of any path from the initial state to the state i. They proved that this upper bound is tight for $m > n-1$. It is easy to see that the second term of (4.4) is $\sum_{j=0}^{f(A)} \binom{n-2}{j}$ if $m > n-1$, and k^{m-1}, otherwise. The value k^{m-1} indicates that the DFA resulting from the concatenation has states with level at most $m-1$. But that is not always the case, as we can see by the example[3] in Figure 4.9. This implies that (4.4) is not an upper bound if $m < n$. Thus, we have

Theorem 4.17. *For any two finite languages L_1 and L_2 with $\mathrm{sc}(L_1) = m$ and $\mathrm{sc}(L_2) = n$ over an alphabet of size $k \geq 2$, one has*

$$\mathrm{sc}(L_1L_2) \leq \sum_{i=0}^{m-2} \min\left\{k^i, \sum_{j=0}^{f(L_1,i)} \binom{n-2}{j}\right\} + \sum_{j=0}^{f(L_1)} \binom{n-2}{j}. \tag{4.5}$$

Proof. The proof follows the one in [CCSY01] considering the changes described above. □

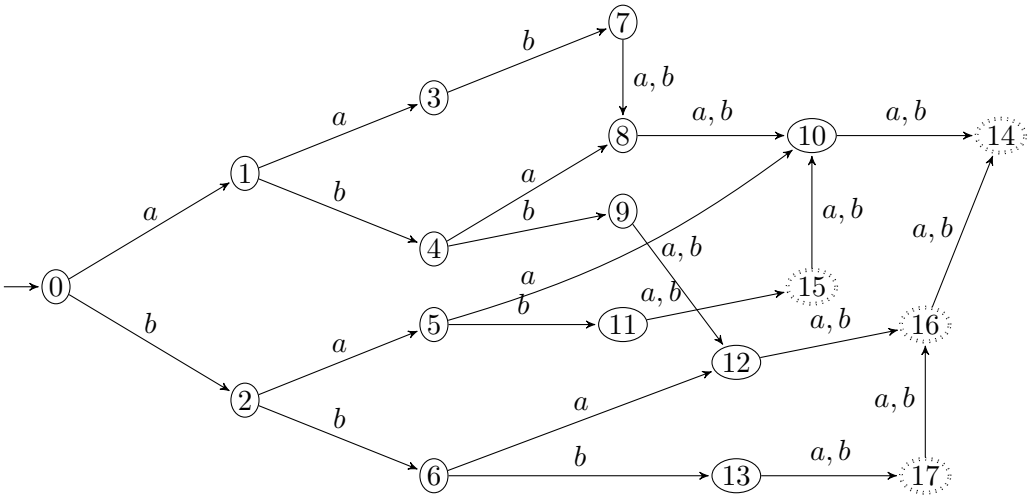


Figure 4.9: DFA resulting from the concatenation of DFA A with $m = 3$ and DFA B with $n = 5$, of Figure 4.11. The states with dashed lines have level > 3 and are not accounted by formula (4.4).

Consider the algorithm for the concatenation presented in the Section 4.1.2, and let $\overline{s}_\sigma(A) = \overline{t}_\sigma(A,0)$. The next theorem presents the upper bounds for the number of states

[3]Note that we are omitting the dead state in the figures.

and transitions of any DFA accepting L_1L_2. Note that the result for the number of states is similar to the Theorem 4.17, omitting the dead state.

Theorem 4.18. *For any two finite languages L_1 and L_2 with $\mathrm{isc}(L_1) = m$ and $\mathrm{isc}(L_2) = n$ over an alphabet of size $k \geq 2$, and making $\Lambda_j = \binom{n-1}{j} - \binom{\overline{t}_\sigma(L_2)-\overline{s}_\sigma(L_2)}{j}$, $\Delta_j = \binom{n-1}{j} - \left(\binom{\overline{t}_\sigma(L_2)-\overline{s}_\sigma(L_2)}{j} * \overline{s}_\sigma(L_2)\right)$ one has*

$$\mathrm{isc}(L_1L_2) \leq \sum_{i=0}^{m-1} \min\left\{k^i, \sum_{j=0}^{f(L_1,i)} \binom{n-1}{j}\right\} + \sum_{j=0}^{f(L_1)} \binom{n-1}{j} - 1 \tag{4.6}$$

and

$$\begin{aligned}\mathrm{itc}(L_1L_2) \leq k \sum_{i=0}^{m-2} \min\left\{k^i, \sum_{j=0}^{f(L_1,i)} \binom{n-1}{j}\right\} + \\ + \sum_{\sigma\in\Sigma}\left(\min\left\{k^{m-1} - \overline{s}_\sigma(L_2), \sum_{j=0}^{f(L_1)-1} \Delta_j\right\} + \sum_{j=0}^{f(L_1)} \Lambda_j\right).\end{aligned} \tag{4.7}$$

Proof. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ and $B = \langle [0, n[, \Sigma, \delta_B, 0, F_B\rangle$ be the minimal DFAs that recognise L_1 and L_2. Consider the DFA C accepting $\mathcal{L}(A)\mathcal{L}(B)$. Let us prove the result for $\mathrm{isc}(L_1L_2)$. Each state of the DFA C has the form (x, P) where $x \in [0, m[\cup \{\Omega_A\}$ and $P \subseteq [0, n[$. The first term of (4.6) corresponds to the maximal number of states of the form (i, P) with $i \in [0, m[$. Such a state (i, P) is at a level $\leq i$, which has at most k^{i-1} predecessors. Thus, the level i has at most k^i states. The maximal size of the set P is $f(A, i)$. For a fixed i, the initial state of the DFA B either belongs to all sets P (if $i \in F_A$) or it is not in any of them. Thus, the number of distinct sets P is at most $\sum_{j=0}^{f(A,i)} \binom{n-1}{j}$. The number of states of the form (i, P) is the minimal of these two values. The second term of (4.6) corresponds to the maximal number of states where the first component is Ω_A. In this case, the size of P is at most $f(A)$. Lastly, we remove the dead state.

Consider now the result for $\mathrm{itc}(L_1L_2)$. The σ-transitions of the DFA C have three forms: (i, β) where i represents the transition leaving the state $i \in [0, m[$; $(-1, \beta)$

where -1 represents the absence of the transition from state $m-1$ to Ω_A; and $(-2,\beta)$ where -2 represents any transition leaving Ω_A. In all forms, β is a set of transitions of DFA B. The number of σ-transitions of the form (i,β) is at most $\sum_{i=0}^{m-2}\min\{k^i,\sum_{j=0}^{f(L_1,i)}\binom{n-1}{j}\}$ which corresponds to the number of states of the form (i,P), for $i\in[0,m[$ and $P\subseteq[0,n[$. The number of σ-transitions of the form $(-1,\beta)$ is $\min\{k^{m-1}-\overline{s}_\sigma(L_2),\sum_{j=0}^{f(L_1)-1}\Delta_j\}$. We have at most k^{m-1} states in this level. However, if $s_\sigma(B,0)=0$ we need to remove the transition $(-1,\emptyset)$ which leaves the state $(m-1,\{0\})$. On the other hand, the size of β is at most $f(L_1)-1$ and we know that β has always the transition leaving the initial state by σ, if it exists. If this transition does not exist, *i.e.* $\overline{s}_\sigma(B,0)=1$, we need to remove the sets with only non-defined transitions, because they originate transitions of the form $(-1,\emptyset)$. The number of σ-transitions of the form $(-2,\beta)$ is $\sum_{j=0}^{f(L_1)}\Lambda_j$ and this case is similar to the previous one. □

4.2.4.1 Worst-case Witnesses

To prove that the bounds are reachable, we consider two cases depending whether $m+1\geq n$ or not.

Case 1: $m+1\geq n$**.** The witness languages are the ones presented by Câmpeanu et al. (see Figure 4.10).

Figure 4.10: DFA A with m states and DFA B with n states.

Theorem 4.19. *For any two integers $m\geq 2$ and $n\geq 2$ such that $m+1\geq n$, there exist an m-state DFA A and an n-state DFA B, both accepting finite languages, such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ needs at least $(m-n+3)2^{n-1}-2$ states and $2^n(m-n+3)-8$ transitions.*

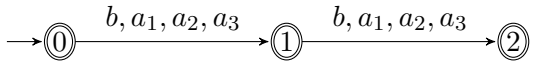
Proof. The proof for the number of states is similar to the proof of [CCSY01, Theorem 4]. Let us consider the number of transitions. The DFA A has $m-1$ σ-transitions for each $\sigma \in \{a, b\}$. The number of final states in the DFA A is m. The DFA B has $n-2$ a-transitions and $n-1$ b-transitions. Consider $m \geq n$. If we analyse the transitions as we did in the proof of the Theorem 4.18 we have $2^{n-1}(m-n+1)-1$ a-transitions and $2^{n-1}(m-n+1)-1$ b-transitions that correspond to the transitions of the form (i, β); $2^{n-1}-2$ a-transitions and 2^{n-1} b-transitions that correspond to the transitions of the form $(-1, \beta)$; and $2^{n-1}-2$ a-transitions and $2^{n-1}-2$ b-transitions that correspond to the transitions of the form $(-2, \beta)$. Thus, we calculate that the total number of transitions is

$$\begin{aligned} &2(2^{n-1}(m-n+1)-1)+2^{n-1}-2+2^{n-1}-2+2^{n-1}+2^{n-1}-2 \\ &= 2^n(m-n+3)-8. \end{aligned}$$

□

Case 2: $m+1 < n$**.** Let $\Sigma = \{b\} \cup \{a_i \mid i \in [1, n-2]\}$. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, [0, m[\,\rangle$ where $\delta_A(i, \sigma) = i+1$, for any $\sigma \in \Sigma$. Let $B = \langle [0, n[, \Sigma, \delta_B, 0, \{n-1\}\rangle$ where $\delta_B(i, b) = i+1$, for $i \in [0, n-2]$, $\delta_B(i, a_j) = i+j$, for $i, j \in [1, n-2]$, $i+j \in [2, n[$, and $\delta_B(0, a_j) = j$, for $j \in [2, n-2]$.

(A)



(B)

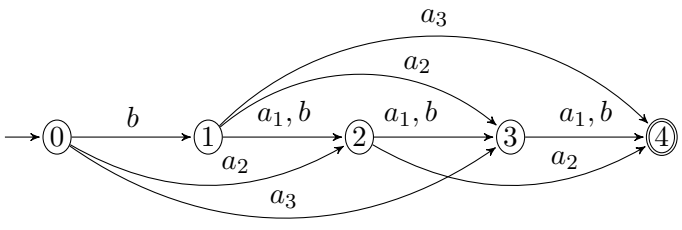


Figure 4.11: DFA A with $m = 3$ states and DFA B with $n = 5$ states.

Theorem 4.20. *For any two integers $m \geq 2$ and $n \geq 2$, with $m+1 < n$, there exist an m-state DFA A and an n-state DFA B, both accepting finite languages over an*

alphabet of size depending on m and n, such that the number of states and transitions of any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ reaches the upper bounds.

Proof. We need to show that the DFA C, resulting from the concatenation algorithm already defined and accepting $\mathcal{L}(A)\mathcal{L}(B)$, is minimal, i.e. (i) every state of C is reachable from the initial state; (ii) each state of C defines a distinct equivalence class. To prove (i), we first show that all states $(i, P) \subseteq R$ with $i \in [1, m[$ are reachable. The following facts hold for the automaton C:

1. every state of the form $(i+1, P')$ is reached by a transition from a state (i, P) (by the construction of A) and $|P'| \leq |P| + 1$, for $i \in [1, m-2]$;

2. every state of the form (Ω_A, P') is reached by a transition from a state $(m-1, P)$ (by the construction of A) and $|P'| \leq |P| + 1$;

3. for each state (i, P), $P \subseteq [0, n[$, $|P| \leq i + 1$ and $0 \in P$, $i \in [1, m[$;

4. for each state (Ω_A, P), $\emptyset \neq P \subseteq [0, n[$, $|P| \leq m$ and $0 \notin P$.

Suppose that for a $i \in [1, m-2]$, all states (i, P) are reachable. The number of states of the form $(1, P)$ is $m-1$ and of the form (i, P) with $i \in [2, m-2]$ is $\sum_{j=0}^{i} \binom{n-1}{j}$. Let us consider the states $(i+1, P')$. If $P' = \{0\}$, then $\delta_C((i, \{0\}), a_1) = (i+1, P')$. Otherwise, let $l = \min(P' \setminus \{0\})$ and $S_l = \{s - l \mid s \in P' \setminus \{0\}\}$. Then,

$$\begin{aligned} \delta_C((i, S_l), a_l) &= (i+1, P') \quad \text{if} \quad 2 \leq l \leq n-2, \\ \delta_C((i, \{0\} \cup S_1), a_1) &= (i+1, P') \quad \text{if} \quad l = n-1, \\ \delta_C((i, S_1), b) &= (i+1, P') \quad \text{if} \quad l = 1. \end{aligned}$$

Thus, all $\sum_{j=0}^{i+1} \binom{n-1}{j}$ states of the form $(i+1, P')$ are reachable. Let us consider the states (Ω_A, P'). P' is always a non-empty set by construction of C. Let $l = \min(P')$

and $S_l = \{s - l \mid s \in P'\}$. Thus,

$$\begin{aligned}
\delta_C((m-1, S_l), a_l) &= (\Omega_A, P') \quad \text{if} \quad 2 \leq l \leq n-2, \\
\delta_C((m-1, \{0\} \cup S_1), a_1) &= (\Omega_A, P') \quad \text{if} \quad l = n-1, \\
\delta_C((m-1, S_1), b) &= (\Omega_A, P') \quad \text{if} \quad l = 1.
\end{aligned}$$

Thus, all $\sum_{j=0}^{m} \binom{n-1}{j} - 1$ states of the form (Ω_A, P') are reachable.

To prove (ii), consider two distinct states $(i, P_1), (j, P_2) \in R$. If $i \neq j$, then $\delta_C((i, P_1), b^{n+m-2-i}) \in F_C$ but $\delta_C((j, P_2), b^{n+m-2-i}) \notin F_C$. If $i = j$, suppose that $P_1 \neq P_2$ and both are final or non-final. Let $P_1' = P_1 \setminus P_2$ and $P_2' = P_2 \setminus P_1$. Without loss of generality, let P_1' be the set which has the minimal value, let us say l. Thus $\delta_C((i, P_1), a_1^{n-1-l}) \in F_C$ but $\delta_C((i, P_2), a_1^{n-1-l}) \notin F_C$. Thus C is minimal.

Let us consider the number of transitions. The DFA A has $m-1$ σ-transitions, for $\sigma \in \Sigma$. The DFA B has $n-1$ b-transitions, $n-2$ a_1-transitions, and $n-i$ a_i-transitions, with $i \in [2, n-2]$. Thus DFA A has $|\Sigma|(m-1)$ transitions, DFA B has $2n-3+\sum_{i=2}^{n-2}(n-i)$ transitions and $|\Sigma| = n-1$. The proof is similar to the proof of Theorem 4.18. □

Theorem 4.21. *The upper bounds for state and transition complexity of concatenation presented in Theorem 4.18 cannot be reached for any alphabet with a fixed size for $m \geq 0$, $n > m+1$.*

Proof. Consider the construction for the concatenation presented in the Section 4.1.2. Let us define the subset $S = \{(\Omega_A, P) \mid 1 \in P\}$ of R. In order for a state (Ω_A, P) to belong to S it has to satisfy the following condition:

$$\exists i \in F_A \exists P' \subseteq 2^{[0,n[} \exists \sigma \in \Sigma : \delta_C((i, P' \cup \{0\}), \sigma) = (\Omega_A, P).$$

The maximal size of S is $\sum_{j=0}^{f(A)-1} \binom{n-2}{j}$, because by construction $1 \in P$ and $0 \notin P$. Assume that Σ has a fixed size $k = |\Sigma|$. Then, the maximal number of words that

reach states of S from r_0 is $\sum_{i=0}^{f(A)} k^{i+1}$ since the words that reach a state $s \in S$ are of the form $w_A\sigma$, where $w_A \in L(A)$ and $\sigma \in \Sigma$. As $n > m$, for some $l \geq 0$ we have $n = m + l$. Thus for an l sufficiently large $\sum_{i=0}^{f(A)} k^{i+1} \ll \sum_{j=0}^{f(A)-1} \binom{m+l-2}{j}$, which is absurd and resulted from supposing that k is fixed. □

4.2.5 Kleene Star

Consider the algorithm for the Kleene star operation presented in the Section 4.1.3. If $f(A) = 1$ then $\mathcal{L}(A)^\star = \mathcal{L}(A)$. Thus, we will consider DFAs with at least two final states. Let $e_\sigma(A) = \sum_{i\in F} t_\sigma(A, i)$ and $\overline{e}_\sigma(A) = \sum_{i\in F} \overline{t}_\sigma(A, i)$. The following results give the number of states and transitions which are sufficient for any DFA B accepting $\mathcal{L}(A)^\star$.

Theorem 4.22. *For any finite language L with* $\mathrm{isc}(L) = m$ *and* $f(L) \geq 2$*, one has* $\mathrm{isc}(L^\star) \leq 2^{m-f(L)-1} + 2^{m-2} - 1$ *and*

$$\mathrm{itc}(L^\star) \leq 2^{m-f(L)-1}\left(k + \sum_{\sigma\in\Sigma} 2^{e_\sigma(L)}\right) - \sum_{\sigma\in\Sigma} 2^{n_\sigma} - \sum_{\sigma\in X} 2^{n_\sigma},$$

where $n_\sigma = \overline{t}_\sigma(L) - \overline{s}_\sigma(L) - \overline{e}_\sigma(L)$ *and* $X = \{\sigma \in \Sigma \mid s_\sigma(L) = 0\}$.

Proof. The proof for the states is similar to the proof presented by Câmpeanu et al. [CCSY01]. Let $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A\rangle$ be the minimal DFA that recognise L. Note that in the star operation the states of the resulting DFA are sets of states of the DFA A. The minimal DFA B accepting $L(A)^\star$, obtained by the referred algorithm, has at most the following states:

(i) the initial state 0_B which corresponds to the initial state of A: 1 state;

(ii) all $P \subseteq [1, m[\setminus F_A$ and $P \neq \emptyset$: $2^{m-f(A)-1} - 1$ states;

(iii) all $P \subseteq [0, m-2]$ such that $P \cap F_A \neq \emptyset$ and $0 \in P$: $2^{m-f(A)-1}(2^{f(A)-1} - 1)$ states;

(iv) all $P = P' \cup \{m-1, 0\}$ where $P' \subseteq [1, m[\setminus F_A$ and $P' \neq \emptyset$: $2^{m-f(A)-1} - 1$ states.

Therefore, the number of states of the DFA B is at most $2^{m-f(A)-1} + 2^{m-2} - 1$. As in [CCSY01, Theorem 1], in the above description we are considering that $0 \notin F_A$. If $0 \in F_A$ the values suffer a few changes but the formula which is obtained, when reaches its maximum, is the same.

The proof for the itc($L^\star$) is similar to the one for the isc($L^\star$). Enumerating the σ-transitions as done for the states, we have that:

(i) the presence or the absence of the transition leaving the initial state: $s_\sigma(L)$ σ-transitions;

(ii) the set of transitions leaving non-initial and non-final states: $2^{m-f(L)-1} - 2^{\overline{t}_\sigma(L) - \overline{s}_\sigma(L) - \overline{e}_\sigma(L)}$;

(iii) the set of transitions leaving the final states (excluding the pre-dead): $2^{m-f(L)-1}(2^{e_\sigma(L)} - 1)$ σ-transitions;

(iv) the set of transitions leaving the pre-dead state: $2^{m-f(L)-1} - 1$ σ-transitions if there exists a σ-transition leaving the initial state, $2^{m-f(L)-1} - 2^{n_\sigma}$ σ-transitions otherwise, where $n_\sigma = \overline{t}_\sigma(L) - \overline{s}_\sigma(L) - \overline{e}_\sigma(L)$.

Thus, the upper bound for itc($L^\star$) holds. □

4.2.5.1 Worst-case Witnesses

The theorem below shows that the previous upper bounds are reachable. The witness family for this operation is the same as the one presented by Câmpeanu et al., but we have to exclude the dead state.

Let $A = \langle [0, m[, \{a, b, c\}, \delta_A, 0, \{m-2, m-1\}\rangle$, $m \geq 4$, be a incomplete DFA accepting a finite language (see Figure 4.12) where:

$$\begin{aligned}
&\delta(i, a) = i + 1, \text{ for } i \in [0, m[,\\
&\delta(i, b) = i + 1, \text{ for } i \in [\,1, m[, \text{ and } \delta(0, b) = m - 1,\\
&\delta(i, c) = i + 1, \text{ for } i \in [\,0, m[\text{ and } m - i \text{ is even.}
\end{aligned}$$

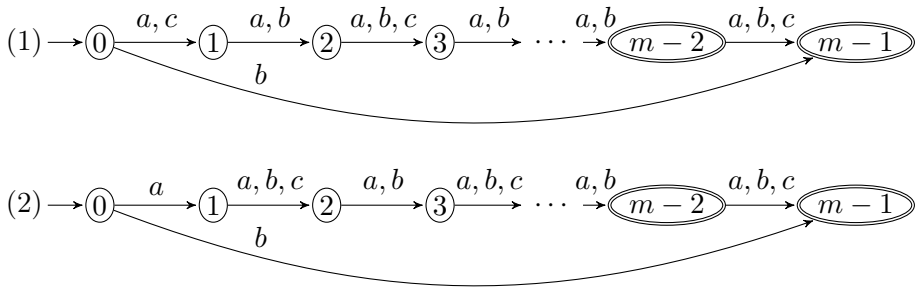


Figure 4.12: DFA A with m states, with m even (1) and odd (2).

Theorem 4.23. *For any integer $m \geq 4$, there exist an m-state DFA A accepting a finite language, such that any DFA accepting $\mathcal{L}(A)^\star$ needs at least $2^{m-2} + 2^{m-3} - 1$ states and $9 \cdot 2^{m-3} - 2^{m/2} - 2$ transitions if m is odd, or $9 \cdot 2^{m-3} - 2^{(m-2)/2} - 2$ transitions, otherwise.*

Proof. The proof for the states is the same as presented by Câmpeanu et al.. Note that we do not count the dead states, and because of this we have one state less in A and in the resulting DFA. Considering the transitions as in the proof of Theorem 4.22, the DFA resulting for the star operation has: $3 \cdot 2^{m-3} - 1$ a-transitions, $3 \cdot 2^{m-3} - 1$ b-transitions, and $3 \cdot 2^{m-3} - 2^{m/2}$ c-transitions if m is odd, or $3 \cdot 2^{m-3} - 2^{(m-2)/2}$ transitions otherwise. Therefore the resulting DFA has $9 \cdot 2^{m-3} - 2^{m/2} - 2$ transitions if m is odd, or $9 \cdot 2^{m-3} - 2^{(m-2)/2} - 2$ transitions, otherwise. □

4.2.6 Reversal

Given an incomplete DFA $A = \langle [0, m[, \Sigma, \delta_A, 0, F_A \rangle$, to obtain a DFA B that accepts $\mathcal{L}(A)^R$, we first reverse all transitions of A and then determinize the resulting NFA. Let $c_\sigma(A, i) = 0$ if $\text{in}_\sigma(A, i) > 0$ and 1 otherwise. In the following result we present upper bounds for the number of states and transitions of B.

Theorem 4.24. *For any finite languages L with $\text{isc}(L) = m$, $m \geq 3$, and over an alphabet of size $k \geq 2$, , where l is the smallest integer such that $2^{m-l} \leq k^l$, one has*

$\mathrm{isc}(L^R) \leq \sum_{i=0}^{l-1} k^i + 2^{m-l} - 1$ *and if* m *is odd,*

$$\mathrm{itc}(L^R) \leq \sum_{i=0}^{l} k^i - 1 + k2^{m-l} - \sum_{\sigma\in\Sigma} 2^{\sum_{i=0}^{l-1} \bar{t}_\sigma(L,i)+1},$$

or, if m *is even,*

$$\mathrm{itc}(L^R) \leq \sum_{i=0}^{l} k^i - 1 + k2^{m-l} - \sum_{\sigma\in\Sigma} \left(2^{\sum_{i=0}^{l-2} \bar{t}_\sigma(L,i)+1} - c_\sigma(L,l)\right).$$

Proof. Let A be the minimal DFA accepting L. The proof for $\mathrm{isc}(L^R)$ is similar to the proof of [CCSY01, Theorem 5]. We only need to remove the dead state.

Let us prove the result for $\mathrm{itc}(L^R)$. The smallest l that satisfies $2^{m-l} \leq k^l$ is the same for m and $m+1$, and because of that we have to consider whether m is even or odd.

Suppose m odd. Let T_1 be the set of transitions corresponding to the first $\sum_{i=0}^{l-1} k^i$ states and T_2 be the set corresponding to the other $2^{m-l} - 1$ states. We have that $|T_1| = \sum_{i=0}^{l-1} k^i - 1$, because the initial state has no transition reaching it. As the states of DFA B are sets of states of DFA A, we also consider each σ-transition of B as a set of σ-transitions of A. If all σ-transitions were defined in A, T_2 would have 2^{m-l} σ-transitions. But, as not all σ-transitions are defined, we remove from 2^{m-l} the sets which only have undefined σ-transitions of A. As the initial state of A has no transitions reaching it, we need to add one to the number of undefined σ-transitions. Thus, $|T_2| = \sum_{\sigma\in\Sigma} 2^{m-l} - 2^{(\sum_{i=0}^{l-1}(\bar{t}_\sigma(i)))+1}$.

Let us consider m even. In this case we also need to consider the set of transitions that connect the states with the highest level in the first set (T_1) with the states with the lowest level in the second set (T_2). As the highest level is $l-1$, we have to remove the possible transitions that reach the state l in DFA A. □

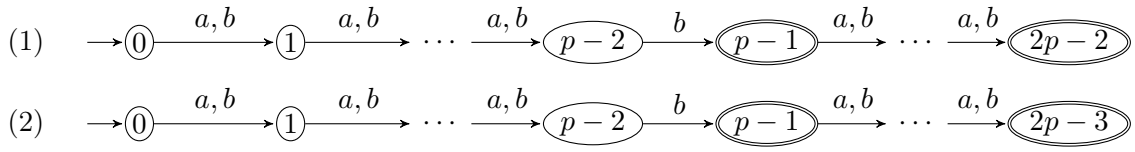


Figure 4.13: DFA A with $m = 2p - 1$ states (1) and with $m = 2p - 2$ (2).

4.2.6.1 Worst-case Witnesses

The following result proves that the upper bounds presented above are tight. The witness family for this operation is the one presented by Câmpeanu et al. but we omit the dead state. It is depicted in Figure 4.13.

Theorem 4.25. *For any integer $m \geq 4$, there exist an m-state DFA A accepting a finite language, such that any DFA accepting $\mathcal{L}(A)^R$ needs at least $3 \cdot 2^{p-1} + 2$ states and $3 \cdot 2^p - 8$ transitions if $m = 2p - 1$ or $2^{p+1} - 2$ states and $2^{p+2} - 7$ transitions if $m = 2p$.*

Proof. The proof for the states is the same as the one presented by Câmpeanu et al. [CCSY01]. Considering the transitions as in the proof of Theorem 4.24, the DFA resulting for the reversal operation, in case $m = 2p - 1$, has:

- $(\sum_{i=0}^{p-1} 2^i) - 1$ transitions in T_1;
- $2^p - 2^2$ a-transitions in T_2;
- $2^p - 2$ b-transitions in T_2.

Thus, the resulting DFA has $3 \cdot 2^p - 8$ transitions. In the other case, the resulting DFA has:

- $(\sum_{i=0}^{p-1} 2^i) - 1$ transitions in T_1;
- $2^p - 2$ a-transitions in T_2;
- $2^{p-1} - 1$ a-transitions in the intermediate set;

- $2^p - 2$ b-transitions in T_2;
- 2^{p-1} b-transitions in the intermediate set.

Therefore the resulting DFA has $2^{p+2} - 7$ transitions. □

4.2.7 Experimental Results

Similarly to the previous section, we performed some experimental tests in order to analyse the practical behaviour of the operations over finite languages. All the tests were performed with uniformly random generated acyclic DFAs.

Table 4.7 shows the results of 20000 experimental tests. The number of states of the operands and the measures are the same as used in Section 4.1.6.

The results obtained were similar to the ones for regular languages. However, for finite languages, the difference between the worst and the average case was not as high as for regular languages. For example, for reversal operation, considering $m = 18$ and regular languages, the upper bound for the number of states was 3700 times larger than the number of states observed and the upper bound for the number of transitions was 5600 times larger than the number of transitions, whereas for finite languages the upper bound for states was only 43 times larger and for transitions 53 times larger. As for regular languages, the DFAs resulting from all the operations (excluding the complement) were also incomplete.

Thus, as what happened for regular languages, we can conjecture that the upper bounds are seldom reached in practical applications.

Table 4.7: Experimental results for finite languages.

m	n	sc	$ubsc$	rs	m_1	m_2	tc	$ubtc$	rt	m_3	m_4	d
Concatenation												
2	18	37.11	88.64	0.42	108	159	163.43	417.57	0.39	530	786	0.85
4	16	63.26	634.06	0.1	236	2096	289.27	2913.99	0.10	1147	10471	0.89
6	14	76.09	2480.07	0.03	268	7256	350.93	8249.58	0.04	1305	36267	0.91
8	12	78.28	3803.77	0.02	252	7024	360.60	11050.80	0.03	1236	35105	0.91
10	10	73.52	2670.73	0.03	260	3296	336.23	8314.19	0.04	1285	16463	0.91
12	8	63.8	1158.59	0.06	170	1208	287.97	4143.13	0.07	837	6031	0.90
14	6	51.2	396.78	0.13	123	398	226.61	1615.26	0.14	600	1981	0.88
16	4	37.69	122.99	0.31	75	123	162.18	540.69	0.30	363	610	0.86
18	2	25.09	36	0.70	33	36	104.01	165.38	0.63	152	175	0.83
Union												
10	10	30.95	98	0.32	57	98	125.41	8314.19	0.02	260	16463	0.81
12	8	29.86	94	0.32	52	94	120.75	416.94	0.29	225	455	0.81
14	6	26.55	82	0.32	47	82	106.98	360.17	0.30	203	395	0.80
16	4	21.84	62	0.35	36	62	88.5	267.03	0.33	151	297	0.81
18	2	18.8	34	0.55	22	34	77.06	142.41	0.54	97	163	0.82
Intersection												
10	10	12.93	66	0.20	33	66	29.54	110.7	0.27	106	256	0.44
12	8	11.91	62	0.19	33	62	26.71	102.39	0.26	92	239	0.43
14	6	9.02	50	0.18	25	50	18.96	79.85	0.24	79	168	0.40
16	4	5.02	30	0.17	14	30	8.76	43.92	0.20	39	114	0.33
18	2	1.78	2	0.89	2	2	1.29	2.47	0.52	5	5	0.13
Star												
2		1	0.75	1.33	1	1	2.59	1.94	1.33	5	5	0.52
4		3.05	4.67	0.65	5	7	11.81	15.07	0.78	25	35	0.78
6		7.43	18.82	0.40	23	31	32.86	65.17	0.50	112	154	0.88
8		14.59	71.46	0.20	73	127	68.05	241.79	0.28	362	631	0.93
10		25.11	274.14	0.092	121	511	120.19	888.33	0.135	598	2549	0.955
12		38.75	1066.12	0.036	192	2047	188.05	3297.08	0.057	949	10226	0.969
14		57.18	4190.58	0.014	416	8191	279.82	12436.48	0.023	2078	40896	0.977
16		79.35	16599.54	0.005	481	32767	390.42	47644.04	0.008	2400	163810	0.982
18		108.37	66019.6	0.002	751	98303	535.28	184747.27	0.003	3745	491492	0.986
Reversal												
2		2	2	1	2	2	2.59	2.59	1	5	5	0.26
4		5.58	7.99	0.70	8	8	12.93	31.72	0.41	29	35	0.46
6		11.87	20.10	0.57	20	21	33.96	96.08	0.35	76	100	0.57
8		21.99	62.00	0.35	44	62	70.66	298.14	0.24	182	305	0.63
10		37.35	158	0.24	94	158	129.88	779.05	0.17	401	785	0.69
12		59.34	411	0.14	144	411	217.31	2042.01	0.11	640	2050	0.72
14		89.91	1179	0.08	247	1179	342.91	5882.71	0.06	1115	5890	0.75
16		130.19	2828	0.05	355	2828	511.46	14126.25	0.04	1629	14135	0.78
18		184.32	8001	0.02	460	8001	742.56	39989.92	0.02	2057	40000	0.80
Complement												
2		3	3	1	3	3	7.77	8	1	8	8	1
4		5	5	1	5	5	24.12	25	1	25	25	1
6		7	7	1	7	7	34.97	35	1	35	35	1
8		9	9	1	9	9	45	45	1	45	45	1
10		11	11	1	11	11	55	55	1	55	55	1
12		13	13	1	13	13	65	65	1	65	$6s5$	1
14		15	15	1	15	15	75	75	1	75	75	1
16		17	17	1	17	17	85	85	1	85	85	1
18		19	19	1	19	19	95	95	1	95	95	1

Chapter 5

Simulation Complexity of Regular Expressions by Non-Deterministic Finite Automata

The development and analysis of algorithms to simulate regular expressions using finite automata is a problem that has been widely studied. The solution to this problem allows the efficient implementation of useful tools in fields like text processing, such as scanner generators (as `lex`), editors (as `emacs`), or API's of programming languages (as `Java` or `Python`).

One of the first simulation methods is due to Thompson. Given a regular expression α, it is defined an inductive tool which construct an ε-NFA for the basic regular expressions together with rules to construct the ε-NFA for the different operators in α, as we illustrated in Section 2.3.1.1.

Another classical simulation method is the position (or Glushkov) automaton which consider a linearised version of the given regular expression α and construct an NFA without transitions labelled by the empty word, as we described in Section 2.3.1.2.

Other simulations, such as partial derivative automata (see Section 2.3.2.2), follow

automata ($\mathcal{A}_f$) [IY03a], or the construction given by Garcia et al. ($\mathcal{A}_u$) [GLRA11], were proved to be quotients of the position automata, by specific right-equivalence relations [CZ02, IY03a].

Recently, Yamamoto [Yam14] presented a new simulation method based on Thompson automaton ($\mathcal{A}_\mathsf{T}$). Given a $\mathcal{A}_\mathsf{T}$, two automata are constructed by merging $\mathcal{A}_\mathsf{T}$ states: in one, the suffix automaton ($\mathcal{A}_{\text{Suf}}$), states with the same right languages are merged and in the other, the prefix automaton ($\mathcal{A}_{\text{Pre}}$), states with the same left languages are merged. $\mathcal{A}_{\text{Suf}}$ corresponds to $\mathcal{A}_{\text{PD}}$, which is not a surprise because, as we already referred, it is known that the $\mathcal{A}_{\text{Pos}}$ is obtained if ε-transitions are eliminated from $\mathcal{A}_\mathsf{T}$. $\mathcal{A}_{\text{Pre}}$ is a quotient of $\mathcal{A}_\mathsf{T}$ by a left-invariant relation.

In this chapter we start by studying several properties of $\mathcal{A}_{\text{PD}}$ automaton (Section 5.1). We also introduce the right derivatives, with which we construct the right derivative automaton, and show its relation with Brzozowski's automaton (Section 5.2). Using the notion of right-partial derivatives, in Section 5.3 we define the right-partial derivative automaton $\overleftarrow{\mathcal{A}}_{\text{PD}}$, characterise its relation with $\mathcal{A}_{\text{PD}}$ and $\mathcal{A}_{\text{Pos}}$, and study its average size. In Section 5.4, we construct the $\mathcal{A}_{\text{Pre}}$ automaton directly from the regular expression without use the $\mathcal{A}_\mathsf{T}$ automaton, and show that it also is a quotient of the $\mathcal{A}_{\text{Pos}}$ automaton. However, experimental results suggest that, on average, the reduction on the size of the $\mathcal{A}_{\text{Pos}}$ is not large. Considering the framework of analytic combinatorics we study this reduction (Section 5.5).

The work presented in Section 5.1 was partially published in an extended abstract from Maia et al. [MMR14]. Another paper from the same authors [MMR15b] expands the work presented in Section 5.3 and Section 5.4.

5.1 Partial Derivative automaton

The partial derivative automaton (Section 2.3.2.2) is a widely studied method of conversion from REs to equivalent NFAs. It is known that the automaton resulting

from this simulation (or conversion) method is a quotient of the position automata, by a specific bisimulation. When REs are in (normalised) star normal form, i.e. when subexpressions of the star operator do not accept the empty word, the resulting $\mathcal{A}_{\mathrm{PD}}$ automaton is a quotient of the follow automaton [COZ04].

The bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$ was studied by Ilie & Yu [IY03b], and of course it is always not larger than all other quotients by bisimulations. Nevertheless, experimental results with uniform random generated REs suggested that, for REs in (normalised) star normal form, the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$ automata almost always coincide with the $\mathcal{A}_{\mathrm{PD}}$ automata [GMR10].

Our goal is to have a better characterization of the $\mathcal{A}_{\mathrm{PD}}$ automata and their relation with the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$. This may help to obtain an algorithm that computes, directly from a regular expression, the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$. For that, we analyse how close the $\mathcal{A}_{\mathrm{PD}}$ is to the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$.

In this section, we present an inductive construction of $\mathcal{A}_{\mathrm{PD}}$ and study several of its properties. For regular expressions without Kleene star we characterise the $\mathcal{A}_{\mathrm{PD}}$ automata and we prove that the $\mathcal{A}_{\mathrm{PD}}$ automaton is isomorphic to the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$, under certain conditions. Thus, for these special regular expressions, we conclude that the $\mathcal{A}_{\mathrm{PD}}$ is an optimal conversion method using right-invariant relations. We close by considering the difficulties of relating the two automata for general regular expressions.

5.1.1 Inductive Characterization of $\mathcal{A}_{\mathrm{PD}}$

Mirkin's construction of the $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is based in a system of equations $\alpha_i = \sigma_1\alpha_{i1} + \cdots + \sigma_k\alpha_{ik} + \varepsilon(\alpha_i)$, with $\alpha_0 \equiv \alpha$ and α_{ij}, $1 \leq j \leq k$, linear combinations of α_i, $0 \leq i \leq n$, $n \geq 0$. A solution $\pi(\alpha) = \{\alpha_1, \ldots, \alpha_n\}$ can be obtained inductively on the structure of α as follows:

$$\pi(\emptyset) = \emptyset, \qquad \pi(\alpha + \beta) = \pi(\alpha) \cup \pi(\beta),$$

$$\begin{aligned} \pi(\varepsilon) &= \emptyset, & \pi(\alpha\beta) &= \pi(\alpha)\beta \cup \pi(\beta), \\ \pi(\sigma) &= \{\varepsilon\}, & \pi(\alpha^\star) &= \pi(\alpha)\alpha^\star. \end{aligned} \tag{5.1}$$

Champarnaud & Ziadi [CZ01] proved that $\mathsf{PD}(\alpha) = \pi(\alpha) \cup \{\alpha\}$ and that the two constructions lead to the same automaton.

As noted by Broda et al. [BMMR12], Mirkin's algorithm to compute $\pi(\alpha)$ also provides an inductive definition of the set of transitions of $\mathcal{A}_{\mathrm{PD}}(\alpha)$. Let $\varphi(\alpha) = \{(\sigma, \gamma) \mid \gamma \in \partial_\sigma(\alpha), \sigma \in \Sigma\}$ and $\lambda(\alpha) = \{\alpha' \mid \alpha' \in \pi(\alpha),\ \varepsilon(\alpha') = \varepsilon\}$, where both sets can be inductively defined as follows:

$$\begin{aligned} \varphi(\emptyset) &= \emptyset, & \varphi(\alpha + \beta) &= \varphi(\alpha) \cup \varphi(\beta), \\ \varphi(\varepsilon) &= \emptyset, & \varphi(\alpha\beta) &= \varphi(\alpha)\beta \cup \varepsilon(\alpha)\varphi(\beta), \\ \varphi(\sigma) &= \{(\sigma, \varepsilon)\},\ \sigma \in \Sigma, & \varphi(\alpha^\star) &= \varphi(\alpha)\alpha^\star; \end{aligned} \tag{5.2}$$

$$\begin{aligned} \lambda(\emptyset) &= \emptyset, & \lambda(\alpha + \beta) &= \lambda(\alpha) \cup \lambda(\beta), \\ \lambda(\varepsilon) &= \emptyset, & \lambda(\alpha\beta) &= \lambda(\beta) \cup \varepsilon(\beta)\lambda(\alpha)\beta, \\ \lambda(\sigma) &= \{\varepsilon\},\ \sigma \in \Sigma, & \lambda(\alpha^\star) &= \lambda(\alpha)\alpha^\star. \end{aligned}$$

We have, $\delta_{pd} = \{\alpha\} \times \varphi(\alpha) \cup F(\alpha)$ where the result of the $\times$ operation is seen as a set of triples $(\alpha', \sigma, \beta')$ and the set F is defined inductively by:

$$\begin{aligned} F(\emptyset) = F(\varepsilon) = F(\sigma) &= \emptyset,\ \sigma \in \Sigma, \\ F(\alpha + \beta) &= F(\alpha) \cup F(\beta), \\ F(\alpha\beta) &= F(\alpha)\beta \cup F(\beta) \cup \lambda(\alpha)\beta \times \varphi(\beta), \\ F(\alpha^\star) &= F(\alpha)\alpha^\star \cup (\lambda(\alpha) \times \varphi(\alpha))\alpha^\star. \end{aligned} \tag{5.3}$$

Note that the concatenation of a transition (α, σ, β) with a regular expression γ is defined by $(\alpha, \sigma, \beta)\gamma = (\alpha\gamma, \sigma, \beta\gamma)$. Then, we can inductively construct the partial derivative automaton of α using the following results.

Proposition 5.1. *For all* $\alpha \in RE$, $F(\alpha) = \{(\tau, \sigma, \tau') \mid \tau \in \mathsf{PD}^+(\alpha) \wedge \tau' \in \partial_\sigma(\tau)\}$.

Proof. As we know that $\mathsf{PD}(\alpha) = \pi(\alpha) \cup \{\alpha\}$ and $\mathsf{PD}(\alpha) = \mathsf{PD}^+(\alpha) \cup \{\alpha\}$ we can conclude that $\mathsf{PD}^+(\alpha) = \pi(\alpha)$. Thus we want to prove that $F(\alpha) = \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha) \wedge \tau' \in \partial_\sigma(\tau)\}$. Let us proceed by induction on the structure of α. For the base cases the equality is obvious.

If $\alpha \equiv \alpha_1 + \alpha_2$ then

$$\begin{aligned}
&\{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1 + \alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\} \\
&= \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1) \cup \pi(\alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\} \\
&= \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1) \wedge \tau' \in \partial_\sigma(\tau)\} \cup \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\} \\
&= F(\alpha_1) \cup F(\alpha_2) = F(\alpha_1 + \alpha_2).
\end{aligned}$$

Let $\alpha \equiv \alpha_1\alpha_2$ then

$$\{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1\alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\} = \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1)\alpha_2 \cup \pi(\alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\}$$

$$= \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1)\alpha_2 \wedge \tau' \in \partial_\sigma(\tau)\} \cup \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\}$$

If $\tau \in \pi(\alpha_1)\alpha_2$, then $\tau = \alpha_1'\alpha_2$, where $\alpha_1' \in \pi(\alpha_1)$, and $\tau' \in \partial_\sigma(\alpha_1'\alpha_2)$.

Thus $\tau' \in \partial_\sigma(\alpha_1')\alpha_2$, or $\tau' \in \partial_\sigma(\alpha_2)$ if $\varepsilon(\alpha_1') = \varepsilon$.

Then $(\tau, \sigma, \tau') \in F(\alpha_1)\alpha_2$ or $(\tau, \sigma, \tau') \in \lambda(\alpha_1)\alpha_2 \times \varphi(\alpha_2)$.

Thus, we have $\{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1 + \alpha_2) \wedge \tau' \in \partial_\sigma(\tau)\} = F(\alpha_1\alpha_2)$.

Considering $\alpha \equiv \alpha_1^\star$ then

$$\{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1^\star) \wedge \tau' \in \partial_\sigma(\tau)\} = \{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1)\alpha_1^\star \wedge \tau' \in \partial_\sigma(\tau)\}$$

If $\tau \in \pi(\alpha_1)\alpha_1^\star$, then $\tau = \alpha_1'\alpha_1^\star$, where $\alpha_1' \in \pi(\alpha_1)$.

As before, $\tau' \in \partial_\sigma(\alpha_1')\alpha_1^\star$ or $\tau' \in \partial_\sigma(\alpha_1^\star)$ if $\varepsilon(\alpha_1') = \varepsilon$.

Thus $(\tau, \sigma, \tau') \in F(\alpha_1)\alpha_1^\star$ or $(\tau, \sigma, \tau') \in (\lambda(\alpha_1) \times \varphi(\alpha_1))\alpha_1^\star$.

Then $\{(\tau, \sigma, \tau') \mid \tau \in \pi(\alpha_1^\star) \wedge \tau' \in \partial_\sigma(\tau)\} = F(\alpha_1^\star)$. □

Proposition 5.2. *For all $\alpha \in RE$, and $\lambda'(\alpha) = \lambda(\alpha) \cup \varepsilon(\alpha)\{\alpha\}$,*

$$\mathcal{A}_{\mathrm{PD}}(\alpha) = \langle \pi(\alpha) \cup \{\alpha\}, \Sigma, \{\alpha\} \times \varphi(\alpha) \cup F(\alpha), \alpha, \lambda'(\alpha) \rangle.$$

Proof. We want to prove that the right-hand side of this equality corresponds to the definition of $\mathcal{A}_{\mathrm{PD}}$ previously presented in Section 2.3.2.2 on page 36. The set of states of both automata is obviously the same, because we know that $\mathsf{PD}(\alpha) = \pi(\alpha) \cup \{\alpha\}$. The same happens for initial and final states. The transition function $\delta_{pd} = \{(\tau, \sigma, \tau') \mid \tau \in \mathsf{PD}(\alpha) \wedge \tau' \in \partial_\sigma(\tau)\}$ can be written as the following union:

$$\{(\alpha, \sigma, \tau') \mid \tau' \in \partial_\sigma(\alpha)\} \cup \{(\tau, \sigma, \tau') \mid \tau \in \mathsf{PD}^+(\alpha) \wedge \tau' \in \partial_\sigma(\tau)\}.$$

The set $\{(\alpha, \sigma, \tau') \mid \tau' \in \partial_\sigma(\alpha)\}$ is clearly equal to $\{\alpha\} \times \varphi(\alpha)$, and by Proposition 5.1 $\{(\tau, \sigma, \tau') \mid \tau \in \mathsf{PD}^+(\alpha) \wedge \tau' \in \partial_\sigma(\tau)\} = F(\alpha)$. Therefore, the automaton here defined is the previous one. □

Figure 5.1 illustrates this inductive construction, where we assume that states are merged whenever they correspond to equal REs.

We can relate the function π and the states of $\mathcal{A}_c$ automaton. Let π' be a function that coincides with π except that $\pi'(\sigma) = \{(\sigma, \varepsilon)\}$ and, in the two last rules, the regular expression, either β or $\alpha^\star$, is concatenated to the second component of each pair in π', i.e.,

$$\begin{aligned} \pi'(\emptyset) &= \emptyset, & \pi'(\alpha + \beta) &= \pi'(\alpha) \cup \pi'(\beta), \\ \pi'(\varepsilon) &= \emptyset, & \pi'(\alpha\beta) &= \pi'(\alpha)\beta \cup \pi'(\beta), \\ \pi'(\sigma) &= \{(\sigma, \varepsilon)\}, & \pi'(\alpha^\star) &= \pi'(\alpha)\alpha^\star. \end{aligned} \tag{5.4}$$

Proposition 5.3. *Let α be a linear regular expression,*

$$\pi'(\alpha) = \{(\sigma_i, c_{\sigma_i}(\alpha)) | i \in \mathsf{pos}(\alpha)\}.$$

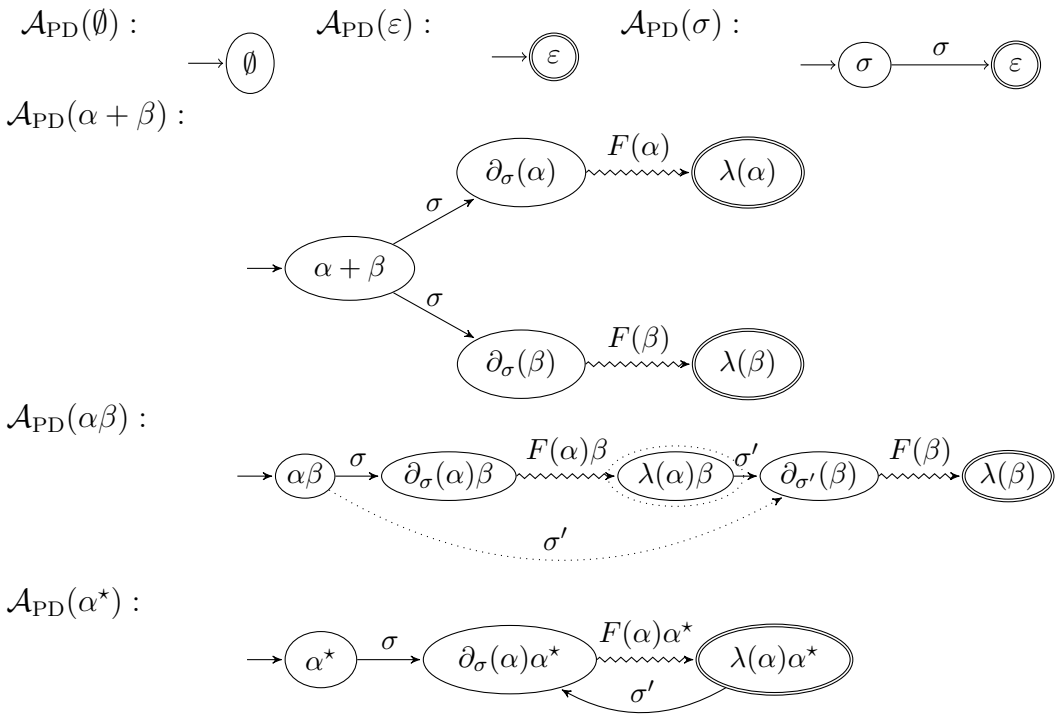


Figure 5.1: Inductive construction of $\mathcal{A}_{\mathrm{PD}}$. The initial states are final if ε belongs to the language of its label. Note that only if $\varepsilon(\beta) = \varepsilon$ the dotted arrow in $\mathcal{A}_{\mathrm{PD}}(\alpha\beta)$ exists and the state $\lambda(\alpha)\beta$ is final.

Proof. Let us proceed by induction on α. For $\alpha \equiv \emptyset$ and $\alpha \equiv \varepsilon$ it is easy to prove that the proposition holds. Considering $\alpha = \sigma_i$, we have that $\{(\sigma_i, c_{\sigma_i}(\sigma_i))\} = \{(\sigma_i, \varepsilon)\} = \pi'(\sigma_i)$. Suppose that the proposition holds for γ and β. If $\alpha \equiv \gamma + \beta$, then

$$\begin{aligned}\{(\sigma_i, c_{\sigma_i}(\alpha)) \mid i \in \mathsf{pos}(\alpha)\} &= \{(\sigma_i, c_{\sigma_i}(\gamma+\beta) \mid i \in \mathsf{pos}(\gamma+\beta)\} \\ &\qquad \text{by the rules in (2.23)} \\ &= \{(\sigma_i, c_{\sigma_i}(\gamma)) \mid i \in \mathsf{pos}(\gamma))\} \cup \{(\sigma_i, c_{\sigma_i}(\beta)) \mid i \in \mathsf{pos}(\beta)\} \\ &= \pi'(\gamma) \cup \pi'(\beta) \;=\; \pi'(\gamma+\beta).\end{aligned}$$

If $\alpha \equiv \gamma\beta$, then

$$\begin{aligned}\{(\sigma_i, c_{\sigma_i}(\alpha)) \mid i \in \mathsf{pos}(\alpha)\} &= \{(\sigma_i, c_{\sigma_i}(\gamma\beta)) \mid i \in \mathsf{pos}(\gamma\beta)\} \\ &\qquad \text{by the rules in (2.23)} \\ &= \{(\sigma_i, c_{\sigma_i}(\gamma)\beta) \mid i \in \mathsf{pos}(\gamma)\} \cup \{(\sigma_i, c_{\sigma_i}(\beta)) \mid i \in \mathsf{pos}(\beta)\} \\ &= \{(\sigma_i, c_{\sigma_i}(\gamma)) \mid i \in \mathsf{pos}(\gamma)\}\beta \;\cup\; \{(\sigma_i, c_{\sigma_i}(\beta)) \mid i \in \mathsf{pos}(\beta)\} \\ &= \pi'(\gamma)\beta \;\cup\; \pi'(\beta) = \pi'(\gamma\beta).\end{aligned}$$

If $\alpha \equiv \gamma^\star$, then

$$\begin{aligned}\{(\sigma_i, c_{\sigma_i}(\alpha)) \mid i \in \mathsf{pos}(\alpha)\} &= \{(\sigma_i, c_{\sigma_i}(\gamma^\star)) \mid i \in \mathsf{pos}(\gamma^\star)\} \\ &\qquad \text{by the rules in (2.23)} \\ &= \{(\sigma_i, c_{\sigma_i}(\gamma)\gamma^\star) \mid i \in \mathsf{pos}(\gamma^\star)\} \\ &= \{(\sigma_i, c_{\sigma_i}(\gamma)) \mid i \in \mathsf{pos}(\gamma)\}\gamma^\star \\ &= \pi'(\gamma)\gamma^\star \;=\; \pi'(\gamma^\star).\end{aligned}$$

Thus, the proposition holds. □

By Proposition 5.3, we can conclude that if we compute $\pi'(\overline{\alpha})$ we obtain exactly[1] the set of states $Q_c \setminus \{(0, c_\varepsilon)\}$ of the c-continuation automaton $\mathcal{A}_c(\alpha)$. Then it is easy

[1]Considering, for each position i, the marked letter σ_i.

(a) N (b) C (c) C^R (d) Q

Figure 5.2: Set of digraphs F.

to see that $\pi(\alpha)$ is obtained by unmarking the c-continuations and removing the first component of each pair, and thus ${}^{Q_c}\!/_{\equiv_c} = \pi(\alpha) \cup \{\alpha\}$.

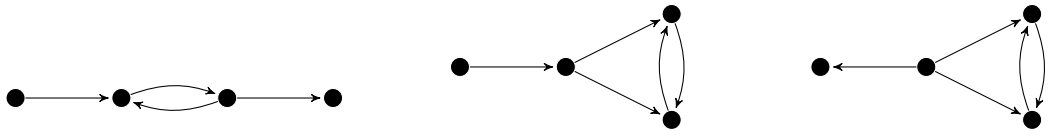
Considering $\overline{\tau} = (a_1 b_2^\star + b_3)^\star a_4$, $\pi'(\overline{\tau}) = \{(a_1, b_2^\star\overline{\tau}), (b_2, b_2^\star\overline{\tau}), (b_3, \overline{\tau}), (a_4, \varepsilon)\}$, which corresponds exactly to the set of states (excluding the initial) of $\mathcal{A}_c(\tau)$, presented in Figure 2.10 on page 35. The set $\pi(\tau)$ is $\{b^\star\tau, \tau, \varepsilon\}$.

5.1.2 $\mathcal{A}_{\mathrm{PD}}$ Minors

As we have already seen, there are several polynomial-time algorithms to transform regular expression into finite automaton with linear size in the size of the input. The dual conversion is, however more difficult: an exponential blowup in size can not be avoided, in general.

Gulan [Gul13] characterise finite automata that can be converted to an expression that is linear in the size of the automata. For that, he studied the interrelations between the size of regular expressions and the digraph structures of the corresponding automata. A similar work was already done for Glushkov automaton [CZ00] and for series-parallel automata [MR09]. Gulan shows that an automaton that require regular expressions of superlinear size to represent it after the conversion, must contain some of the seven substructures (minors) represented in Figure 5.2 and in Figure 5.3. In this section we will show examples for which the partial derivative automaton contains these substructures.

First, let us introduce some notation following Gulan. Formally, a digraph is a 4-tuple $G = (V_G, A_G, t_G, h_G)$ where V and A are finite disjoint sets, called the vertices and the



(a) ϕ (b) ψ (c) ψ^R

Figure 5.3: Set of digraphs K.

arcs of G, and t and h are maps from A to V. The image of $a \in A_G$ under t_G (h_G) is called the tail (head) of a in G. If $t(a) = x$ and $h(a) = y$, we say that a leaves x and enters y, or that a is an xy-arc.

The digraphs G and G^R arise from the other by arc-reversal. A digraph F is a *subgraph* of a digraph G if the removal of vertices and arcs from G yields F.

An (x, y)-*walk* W of length n in G is a sequence $W = a_1, \ldots, a_n \in A_G$, where $t(a_1) = x$, $h(a_i) = t(a_{i+1})$, for $1 \leq i < n$, and $h(a_n) = y$. The vertices x and y are the *endpoints* of W, and every $h(a_i)$ for $1 \leq i < n$ is an *internal* vertex of W.

An (x, y)-*path* in a digraph G is an (x, y)-walk such that neither x nor y is an internal vertex and every internal vertex occurs exactly only once. Two paths $P_1 = a_1, \ldots, a_n$, $P_2 = b_1, \ldots, b_m \subseteq G$ are *internally disjoint* if $\{a_1, \ldots, a_n\} \cap \{b_1, \ldots, b_m\}$ contains no internal vertex of either path.

An *embedding* of F in G is an injection $e : V_F \to V_G$ satisfying that if $a = xy \in A_F$, then G contains an $e(x)e(y)$-path P_a, and that P_a and $P_{a'}$ are internally disjoint for distinct $a, a' \in A_F$. If an embedding of F in G exists, we call F a *minor* of G realised by the embedding.

The digraph *underlying* an automaton $A = \langle Q, \Sigma, \delta, I, F \rangle$ is $G(A) = (Q, \delta, t, h)$, where $t(p, \sigma, q) = p$ and $h(p, \sigma, q) = q$.

A useful acyclic NFA with one final state is *series parallel* if N (see Figure 5.2) is a minor of its underlying digraph.

As we already refered, Gulan proved that, given an automaton A, an exponential blowup cannot be avoided in the size of an regular expression equivalent to A, if at

least one of the digraphs from F (Figure 5.2) or K (Figure 5.3) is a minor of the underlying digraph of A.

In Figure 5.4 and Figure 5.5 we present some examples of partial derivative automata for which the digraphs from F and K are minors of its underlying digraph. We consider partial derivative automata constructed from linear regular expressions (Figure 5.4) and from general regular expressions (Figure 5.5).

Given a regular expression α for which we know that a certain minor X occurs in the underlying digraph of $\mathcal{A}_{\mathrm{PD}}(\alpha)$, to find a family of regular expression α_n for which that minor still occurs in the underlying digraph of $\mathcal{A}_{\mathrm{PD}}(\alpha_n)$ it is sufficient to add disjunctions to α with new alphabet symbols, i.e., symbols which do not appear in α. For example, we know that the minor N occurs in the underlying digraph of $\mathcal{A}_{\mathrm{PD}}(a(ac+b)+bc)$. Thus it also occurs in the underlying digraph of $\mathcal{A}_{\mathrm{PD}}(a(ac+b)+bc+\sum_{i=1}^{k} a_i)$ for any $k \geq 1$.

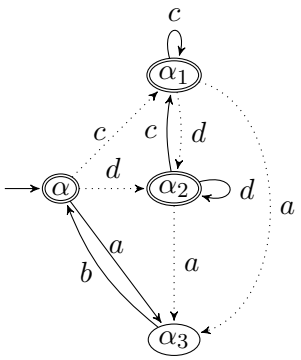
Following Gulan and considering these examples we can conclude that, in general, a partial derivative automaton A cannot be converted to a regular expression that is linear in the size of A.

5.1.3 $\mathcal{A}_{\mathrm{PD}}$ Characterisations

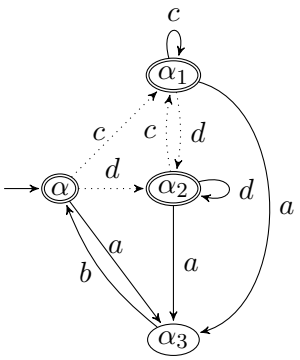
We want to characterise the $\mathcal{A}_{\mathrm{PD}}$ automaton and determine when it coincides with the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}$. In this section and in the following one (Section 5.1.4), we only consider REs *normalised* under the following conditions:

- The expression α is *reduced* according to:
  - the equations $\emptyset + \alpha = \alpha + \emptyset = \alpha$, $\varepsilon \cdot \alpha = \alpha \cdot \varepsilon = \alpha$, $\emptyset \cdot \alpha = \alpha \cdot \emptyset = \emptyset$;
  - and the rule, for all subexpressions β of α, $\beta = \gamma + \varepsilon \implies \varepsilon(\gamma) = \emptyset$.
- The expression α is in *star normal form* (snf).

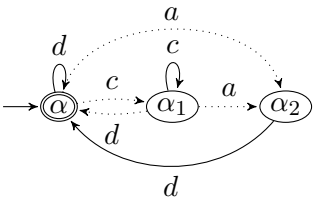
Every regular expression can be converted into an equivalent normalised RE in linear time.



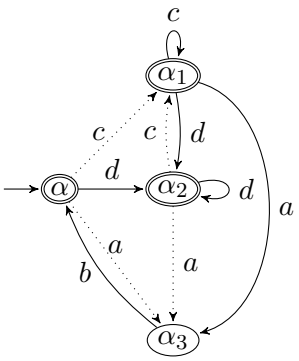
(a) N. $\alpha = (c^\star(ab + d^\star))^\star$, $\alpha_1 = (c^\star(ab + d^\star))\alpha$, $\alpha_2 = d^\star\alpha$, $\alpha_3 = b\alpha$



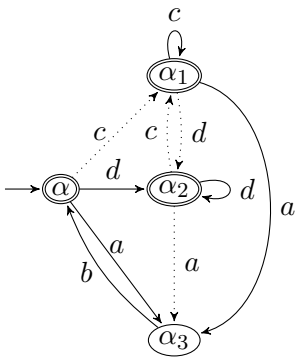
(b) C. $\alpha = (c^\star(ab + d^\star))^\star$, $\alpha_1 = (c^\star(ab + d^\star))\alpha$, $\alpha_2 = d^\star\alpha$, $\alpha_3 = b\alpha$



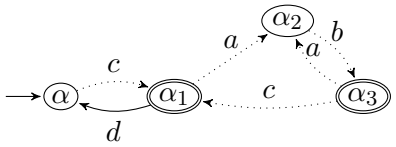
(c) C^R. $\alpha = (c^\star(ab + d))^\star$, $\alpha_1 = (c^\star(ab + d))^\star\alpha$, $\alpha_2 = b\alpha$



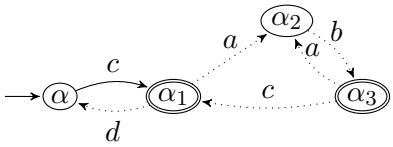
(d) Q. $\alpha = (c^\star(ab + d^\star))^\star$, $\alpha_1 = (c^\star(ab + d^\star))\alpha$, $\alpha_2 = d^\star\alpha$, $\alpha_3 = b\alpha$



(e) ϕ. $\alpha = (c^\star(ab + d^\star))^\star$, $\alpha_1 = (c^\star(ab + d^\star))\alpha$, $\alpha_2 = d^\star\alpha$, $\alpha_3 = b\alpha$

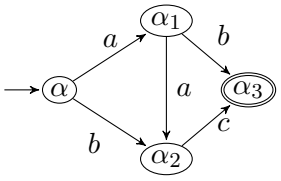


(f) ψ. $\alpha = (c((ab)^\star + d))^\star$, $\alpha_1 = ((ab)^\star + d)\alpha$, $\alpha_2 = b(ab)^\star\alpha$, $\alpha_3 = (ab)^\star\alpha$

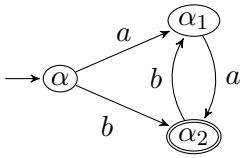


(g) ψ^R. $\alpha = (c((ab)^\star + d))^\star$, $\alpha_1 = ((ab)^\star + d)\alpha$, $\alpha_2 = b(ab)^\star\alpha$, $\alpha_3 = (ab)^\star\alpha$

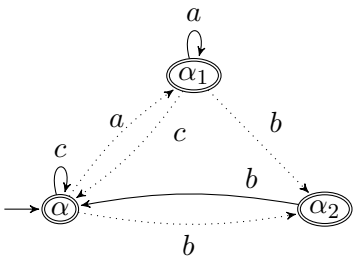
Figure 5.4: $\mathcal{A}_{\text{PD}}$ for which minors from F and K occur (linear REs).



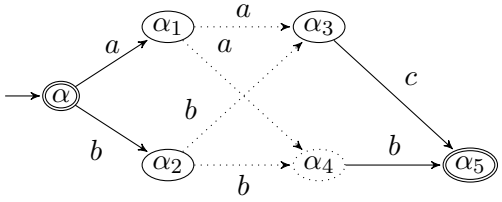
(a) N. $\alpha = a(ac + b) + bc$, $\alpha_1 = (ac + b)$, $\alpha_2 = c$, $\alpha_3 = \varepsilon$



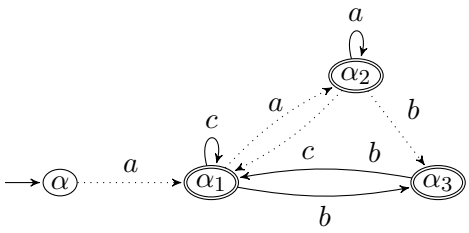
(b) C. $\alpha = a(ab)^\star + ab(ab)^\star$, $\alpha_1 = b(ab)^\star$ $\alpha_2 = (ab)^\star$



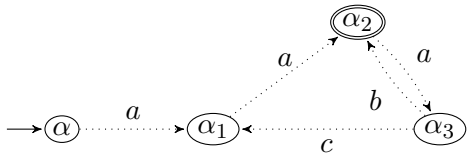
(c) C^R. $\alpha = (a^\star(bb + c))^\star$, $\alpha_1 = a^\star(bb + c)\alpha$ $\alpha_2 = b\alpha$



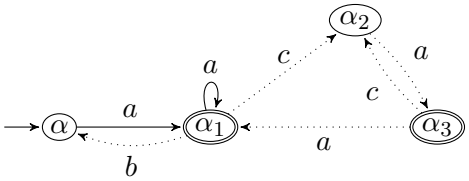
(d) Q. $\alpha = a(ab + ac) + (bb + bc)$, $\alpha_1 = (ab) + (ac)$, $\alpha_2 = (bb) + (bc)$, $\alpha_3 = b$, $\alpha_4 = c$, $\alpha_5 = \varepsilon$



(e) ϕ. $\alpha = a\alpha_1$, $\alpha_1 = (a^\star(bb + c))^\star$, $\alpha_2 = a^\star(bb + c)\alpha_1$ $\alpha_2 = b\alpha_1$



(f) ψ. $\alpha = a\alpha_1$ $\alpha_1 = a\alpha_2$, $\alpha_2 = (a(b + ca))^\star$, $\alpha_3 = (b + ca)\alpha_2$



(g) ψ^R. $\alpha = (a(b + (ca)^\star))^\star$, $\alpha_1 = (b + (ca)^\star)\alpha$, $\alpha_2 = (a(ca)^\star)\alpha$, $\alpha_3 = (ca)^\star\alpha$

Figure 5.5: $\mathcal{A}_{\mathrm{PD}}$ for which minors from F and K occur.

It is known that if α is a normalised regular expression, the $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is a quotient of the Follow automaton of α, and so $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is the smaller known direct ε-free automaton construction from a regular expression. As we discuss in Subsection 5.1.4.2, to solve the problem in the general case it is difficult, mainly because the lack of unique normal forms. Here, we give some partial solutions. First, we consider linear regular expressions and, in Subsection 5.1.3.2, we solve the problem for regular expressions representing finite languages.

5.1.3.1 Linear Regular Expressions

Given a linear regular expression α, it is obvious that the $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ is deterministic. In this case, all positions correspond to distinct letters and transitions from a same state have distinct labels. Thus, $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is also deterministic. The following result is proved by Champarnaud & Ziadi [COZ04].

Proposition 5.4. *Let σ_i and σ_j be two distinct letters of a normalised linear regular expression α. Then the following equivalence holds:*

$$c_{\sigma_i}(\alpha) \equiv c_{\sigma_j}(\alpha) \Leftrightarrow \forall \sigma \in \Sigma,\ d_\sigma(c_{\sigma_i}(\alpha)) \equiv d_\sigma(c_{\sigma_j}(\alpha)).$$

Proposition 5.5. *If α is a normalised linear regular expression, $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is minimal.*

Proof. By Proposition 5.4 we know that

$$c_{\sigma_i}(\alpha) \not\equiv c_{\sigma_j}(\alpha) \Leftrightarrow \{\sigma \mid d_\sigma(c_{\sigma_i}(\alpha)) \neq \emptyset\} \neq \{\sigma \mid d_\sigma(c_{\sigma_j}(\alpha)) \neq \emptyset\}$$

where α is a normalised linear regular expression and σ_i and σ_j are two distinct letters. Recall that $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {\mathcal{A}_c(\alpha)}/{\equiv_c}$. We want to prove that any two states $c_{\sigma_i}(\alpha)$ and $c_{\sigma_j}(\alpha)$ of ${\mathcal{A}_c(\alpha)}/{\equiv_c}$ are distinguishable. We know that $c_{\sigma_i}(\alpha) \not\equiv c_{\sigma_j}(\alpha)$, because $c_{\sigma_i}(\alpha)$ and $c_{\sigma_j}(\alpha)$ are different states of ${\mathcal{A}_c(\alpha)}/{\equiv_c}$. Consider $\sigma' \in \Sigma$ such that $\sigma' \in \{\sigma \mid d_\sigma(c_{\sigma_i}(\alpha)) \neq \emptyset\}$ but $\sigma' \notin \{\sigma \mid d_\sigma(c_{\sigma_j}(\alpha)) \neq \emptyset\}$. Let δ represent ${\delta_c}/{\equiv_c}$. Then $\delta(c_{\sigma_i}(\alpha), \sigma') = c_{\sigma'}(\alpha)$. By construction, we know that $\exists w \in \Sigma^\star$ such that

$\delta(c_{\sigma'}(\alpha), w) \in {}^{F}\!/\!_{\equiv_c}$. Let $w' = \sigma' w$. Therefore $\delta(c_{\sigma_i}(\alpha), w') = \delta(c_{\sigma'}(\alpha), w) \in {}^{F}\!/\!_{\equiv_c}$ and either δ is not defined for $(c_{\sigma_j}(\alpha), w')$ or $\delta(c_{\sigma_j}(\alpha), w')$ is a non final dead state. Thus, the two states are distinguishable. □

5.1.3.2 Finite Languages

In this section, we consider normalised regular expressions without the Kleene star operator, i.e., that represent finite languages. These regular expressions are named *finite regular expressions*.

The following results present some properties of $\mathcal{A}_{\mathrm{PD}}$ automaton.

Proposition 5.6. *The $\mathcal{A}_{\mathrm{PD}}(\alpha) = \langle \mathsf{PD}(\alpha), \Sigma, \delta_\alpha, \alpha, F_\alpha \rangle$ automaton of any finite regular expression $\alpha \not\equiv \emptyset$ has the following properties:*

1. *The state labeled by ε always exists and is a final state;*
2. *The state labeled by ε is reachable from any other state;*
3. *$|F_\alpha| \leq |\alpha|_\varepsilon + 1$;*
4. *The size of each element of $\mathsf{PD}(\alpha)$ is not greater than $|\alpha|$.*

Proof. We use the inductive construction of $\mathcal{A}_{\mathrm{PD}}(\alpha)$.

1. The state labeled by ε is a final state by definition. Let us prove that it always exists. For the base cases this is obviously true. If $\alpha \equiv \gamma + \beta$, then $\pi(\alpha) = \pi(\gamma) \cup \pi(\beta)$. As $\varepsilon \in \pi(\gamma)$ and $\varepsilon \in \pi(\beta)$, by inductive hypothesis, then $\varepsilon \in \pi(\alpha)$. If $\alpha \equiv \gamma\beta$, then $\pi(\alpha) = \pi(\gamma)\beta \cup \pi(\beta)$. As $\varepsilon \in \pi(\beta)$, $\varepsilon \in \pi(\alpha)$.
2. Recall that any state β is reachable if $\exists w \in \Sigma^\star \ \varepsilon \in \partial_w(\beta)$. If α is ε or σ the proposition is obviously true. Let α be $\gamma + \beta$. The states of $\mathcal{A}_{\mathrm{PD}}(\alpha)$ are $\{\alpha\} \cup \pi(\gamma) \cup \pi(\beta)$. By construction, there exists at least a transition from the state α to a (distinct) state in $\pi(\gamma) \cup \pi(\beta)$. Let α be $\gamma\beta$. The states

of $\mathcal{A}_{\mathrm{PD}}(\alpha)$ are $\{\alpha\} \cup \pi(\gamma)\beta \cup \pi(\beta)$. For $\beta' \in \{\beta\} \cup \pi(\beta)$, $\exists w_\beta\ \varepsilon \in \partial_{w_\beta}(\beta')$. In the same way, for $\gamma' \in \{\gamma\} \cup \pi(\gamma)$, $\exists w_\gamma\ \varepsilon \in \partial_{w_\gamma}(\gamma')$. Thus, for $\alpha' = \gamma'\beta \in \pi(\gamma)\beta$, we can conclude that $\varepsilon \in \partial_{w_\gamma w_\beta}(\alpha')$, because $\partial_{w_\gamma w_\beta}(\alpha') = \partial_{w_\beta}(\partial_{w_\gamma}(\gamma'\beta)) \supseteq \partial_{w_\beta}(\partial_{w_\gamma}(\gamma')\beta) \supseteq \partial_{w_\beta}(\beta)$. From the state α we can reach the state ε because the transitions leaving it go to states in $\pi(\gamma)\beta \cup \pi(\beta)$ which reach the state ε.

3. For the base cases it is obviously true. Let α be $\gamma + \beta$. We know that $|\alpha|_\varepsilon = |\gamma|_\varepsilon + |\beta|_\varepsilon$, because $\varepsilon(\alpha) = \varepsilon$ if either $\varepsilon(\gamma)$ or $\varepsilon(\beta)$ are ε, and $|F_\alpha| \leq |F_\gamma| + |F_\beta| - 1$, because $\varepsilon \in F_\gamma \cap F_\beta$. Then $|F_\alpha| \leq |\gamma|_\varepsilon + |\beta|_\varepsilon + 1 = |\alpha|_\varepsilon + 1$. If α is $\gamma\beta$ we also know that $|\alpha|_\varepsilon = |\gamma|_\varepsilon + |\beta|_\varepsilon$ and that $\varepsilon(\alpha) = \varepsilon$ if $\varepsilon(\gamma)$ and $\varepsilon(\beta)$ are ε. If $\varepsilon(\beta) = \varepsilon$, then $|F_\alpha| \leq |F_\gamma| + |F_\beta| - 1$. Otherwise, $|F_\alpha| = |F_\beta|$. We have, in the both cases, $|F_\alpha| \leq |\gamma|_\varepsilon + |\beta|_\varepsilon + 1 \leq |\alpha|_\varepsilon + 1$.

4. If $\alpha \equiv \varepsilon$ or σ the proposition is obviously true. Let α be $\gamma + \beta$. For all $\alpha_i \in \pi(\alpha) = \pi(\gamma) \cup \pi(\beta)$, $|\alpha_i| \leq |\gamma|$ or $|\alpha_i| \leq |\beta|$, and thus $|\alpha_i| \leq |\alpha|$. If α is $\gamma\beta$, then $\pi(\alpha) = \pi(\gamma)\beta \cup \pi(\beta)$. For $\gamma_i \in \pi(\gamma)$, $|\gamma_i| \leq |\gamma|$. If $\alpha_i \in \pi(\gamma)\beta$, $\alpha_i = \gamma_i\beta$. Then, $|\alpha_i| \leq |\gamma| + |\beta| = |\alpha|$ if $\gamma_i \neq \varepsilon$, or $|\alpha_i| = |\beta| \leq |\alpha|$ otherwise . If $\alpha_i \in \pi(\beta)$, $|\alpha_i| \leq |\beta| \leq |\alpha|$.

□

Caron & Ziadi [CZ00] characterised the position automaton in terms of the properties of the underlying digraph. We consider a similar approach to characterise the $\mathcal{A}_{\mathrm{PD}}$ for finite languages. We will restrict the analysis to acyclic NFAs. We first observe that $\mathcal{A}_{\mathrm{Pos}}$ are series-parallel automata (see page 112) which is not the case for all $\mathcal{A}_{\mathrm{PD}}$, as can be seen considering $\mathcal{A}_{\mathrm{PD}}(a(ac + b) + bc)$ (see Figure 5.6).

Let $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ be an acyclic NFA. A is an *hammock* if it has the following properties. If $|Q| = 1$, A has no transitions. Otherwise, there exists an unique $f \in F$ such that for any state $q \in Q$ one can find a path from q_0 to f going through q. The state q_0 is called the *root* and f the *anti-root*. The *rank* of a state $q \in Q$, named $rk(q)$, is the length of the longest word $w \in \Sigma^\star$ such that $\delta(q, w) \in F$. In an hammock, the

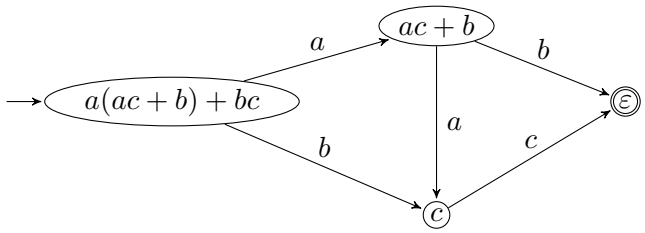


Figure 5.6: $\mathcal{A}_{\mathrm{PD}}(a(ac + b) + bc)$.

anti-root has rank 0. Each state q of rank $r \geq 1$, has only transitions to states in smaller ranks and at least one transition for a state in rank $r - 1$.

Proposition 5.7. *For every finite regular expression α, $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is an hammock.*

Proof. If the partial derivative automaton has a unique state then it is the $A_{pd}(\varepsilon)$ or $A_{pd}(\emptyset)$ which has no transitions. Otherwise, for all $q \in \mathsf{PD}(\alpha)$ there exists at least one path from $q_0 = \alpha$ to q because $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is initially connected; also, there exists at least one path from q to ε, the anti-root, by Proposition 5.6, item 2. □

Proposition 5.8. *An acyclic NFA $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ is a partial derivative automaton of some finite regular expression α, if the following conditions holds:*

1. *A is an hammock;*
2. *$\forall q, q' \in Q\ rk(q) = rk(q') \implies \exists \sigma \in \Sigma\ \delta(q, \sigma) \neq \delta(q', \sigma)$.*

Proof. First we give an algorithm that allows to associate to each state of an hammock A a regular expression. Then, we show that if the second condition holds, A is the $\mathcal{A}_{\mathrm{PD}}(\alpha)$ where α is the RE associated to the initial state.

We label each state q with a regular expression $RE(q)$, considering the states by increasing rank order. We define for the anti-root f, $RE(f) = \varepsilon$. Suppose that all states of ranks smaller than n are already labelled. Let $q \in Q$ with $rk(q) = n$. For $\sigma \in \Sigma$, with $\delta(q, \sigma) = \{q_1, \ldots, q_m\}$ and $RE(q_i) = \beta_i$ we construct the regular expression $\sigma(\beta_1 + \cdots + \beta_m)$. Then,

$$RE(q) = \sum_{\sigma \in \Sigma} \sigma(\beta_1 + \cdots + \beta_m)$$

where we omit all $\sigma \in \Sigma$ such that $\delta(q, \sigma) = \emptyset$. We have, $RE(q_0) = \alpha$

To show that if A satisfies condition 2. then $A \simeq \mathcal{A}_{\mathrm{PD}}(\alpha)$, we need to prove that $RE(q) \not\equiv RE(q')$ for all $q, q' \in Q$ with $q \neq q'$. We proceed by induction on the rank. For rank 0, it is obvious. Suppose that all states with rank $m < n$ are labelled by different regular expressions. Let $q \in Q$, with $rk(q) = n$. We must prove that $RE(q) \not\equiv RE(q')$ for all q' with $rk(q') \leq n$. Suppose that $rk(q) = rk(q')$, $RE(q) = \sigma_1(\alpha_1 + \cdots + \alpha_n) + \cdots + \sigma_i(\beta_1 + \cdots + \beta_m)$, and $RE(q') = \sigma'_1(\alpha'_1 + \cdots + \alpha'_{n'}) + \cdots + \sigma'_j(\beta'_1 + \cdots + \beta'_{m'})$. We know that $\exists \sigma \delta(q, \sigma) \not\equiv \delta(q', \sigma)$. Suppose that $\sigma = \sigma_1 = \sigma'_1$. Then we know that $\exists t, t'\ \alpha_t \not\equiv \alpha'_{t'}$, thus $RE(q) \not\equiv RE(q')$. If $rk(q) > rk(q')$, then there exists a $w \in \Sigma^\star$ with $|w| = n$ such that $\delta(q, w) \cap F \neq \emptyset$ and $\delta(q', w) \cap F = \emptyset$. Thus $RE(q) \not\equiv RE(q')$. □

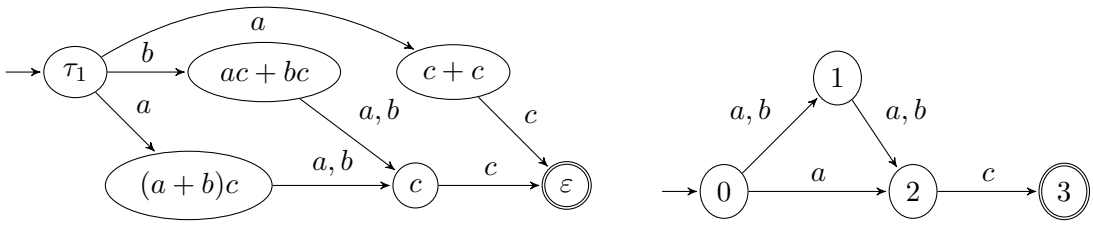
5.1.4 Comparing $\mathcal{A}_{\mathrm{PD}}$ and ${}^{\mathcal{A}_{\mathrm{Pos}}}\!/_{\equiv_b}$

As we already mentioned, there are many (normalised) regular expressions α for which $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {}^{\mathcal{A}_{\mathrm{Pos}}(\alpha)}\!/_{\equiv_b}$. Moreover, if we consider linear regular expressions α, it follows from the Proposition 5.5 that $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {}^{\mathcal{A}_{\mathrm{Pos}}(\alpha)}\!/_{\equiv_b}$, because if A is a DFA then ${}^{A}\!/_{\equiv_b}$ is the minimal DFA equivalent to A. However, even for REs representing finite languages this is not always true. Taking, for example, $\tau_1 = a(a + b)c + b(ac + bc) + a(c + c)$, the corresponding $\mathcal{A}_{\mathrm{PD}}(\tau_1)$ is the one represented in Figure 5.7. The states that are bisimilar are equivalent modulo the + idempotence and left-distributivity. It is also easy to see that two states are bisimilar if they are equivalent modulo + associativity or + commutativity.

5.1.4.1 Finite Languages

In this section we establish some conditions for which the $\mathcal{A}_{\mathrm{PD}}$ automaton of a finite regular expression α is isomorphic to the bisimilarity of $\mathcal{A}_{\mathrm{Pos}}(\alpha)$.

Considering an order $<$ on Σ and assuming that $\cdot < +$, we can extend $<$ to REs.



(a) $\mathcal{A}_{\mathrm{PD}}(\tau_1)$. (b) $\mathcal{A}_{\mathrm{Pos}}(\tau_1)/_{\equiv_b}$.

Figure 5.7: $\tau_1 = a(a+b)c + b(ac+bc) + a(c+c)$.

Then, the following rewriting system is confluent and terminating:

$$\begin{aligned}
\alpha + (\beta + \gamma) &\to (\alpha + \beta) + \gamma && && (+ \textbf{A}\text{ssociativity}),\\
\alpha + \beta &\to \beta + \alpha && \text{if } \beta < \alpha && (+ \textbf{C}\text{ommutativity}),\\
\alpha + \alpha &\to \alpha && && (+ \textbf{I}\text{dempotence}),\\
(\alpha\beta)\gamma &\to \alpha(\beta\gamma) && && (.\ \textbf{A}\text{ssociativity}),\\
(\alpha + \gamma)\beta &\to \alpha\beta + \gamma\beta && && (\textbf{L}\text{eft distributivity}).
\end{aligned}$$

A (normalised) regular expression α that can not be rewritten anymore by this system is called an *irreducible regular expression modulo* ACIAL.

Remark 1. *An irreducible regular expression modulo* ACIAL α *is of the form:*

$$\sum_{i=1}^{n} w_i + \sum_{j=1}^{m} w'_j \alpha_j \tag{5.5}$$

where w_i, w'_j *are words for* $1 \leq i \leq n$, $1 \leq j \leq m$, *and* α_j *are expressions of the same form of* α, *for* $1 \leq j \leq m$. *For each normalised RE without the Kleene star operator, there exits a unique normal form.*

For example, considering $a < b < c$, the normal form for the RE τ_1 given above is $\tau_2 = ac + a(ac+bc) + b(ac+bc)$ and $\mathcal{A}_{\mathrm{PD}}(\tau_2) \simeq \mathcal{A}_{\mathrm{Pos}}(\tau_2)/_{\equiv_b}$. As we will see next, for normal forms this isomorphism always holds.

The following lemmas are needed to prove the main result.

Lemma 5.9. *For $\sigma \in \Sigma$, the function ∂_σ is closed modulo* ACIAL.

Proof. We know that α has the form $w_1 + \cdots + w_n + w'_1\alpha_1 + \cdots + w'_m\alpha_m$, where $w_i = \sigma_i v_i, v_i \in \Sigma^\star$, $w'_j = \sigma_j v'_j, v'_j \in \Sigma^\star$, $i \in \{1, \cdots, n\}$, $j \in \{1, \cdots, m\}$. Thus, $\forall \sigma \in \Sigma\ \partial_\sigma(\alpha) = \partial_\sigma(w_1) \cup \cdots \cup \partial_\sigma(w_n) \cup \partial_\sigma(w'_1)\alpha_1 \cup \cdots \cup \partial_\sigma(w'_i)\alpha_m$, where $\partial_\sigma(w_i) = v_i$ if $\sigma_i = \sigma$ or $\partial_\sigma(w_i) = \emptyset$ otherwise; and $\partial_\sigma(w'_j)\alpha_j = v'_j\alpha_j$, if $\sigma_j = \sigma$ or $\partial_\sigma(w'_j)\alpha_j = \emptyset$ otherwise. Then it is obvious that the possible results are irreducible modulo ACIAL, and the proposition holds. □

Lemma 5.10. *For $w, w' \in \Sigma^\star$,*

1. $(\forall \sigma \in \Sigma)\ |\partial_\sigma(w)| \leq 1$.
2. $w \neq w' \implies (\forall \sigma \in \Sigma)\ (\partial_\sigma(w) \neq \partial_\sigma(w')\ \vee\ \partial_\sigma(w) = \partial_\sigma(w') = \emptyset)$.
3. $(\forall \sigma \in \Sigma)\partial_\sigma(w\alpha) = \partial_\sigma(w)\alpha = \{w'\alpha\}$, *if* $w = \sigma w'$.

Proposition 5.11. *Given α and β irreducible finite regular expressions modulo* ACIAL,

$$\alpha \not\equiv \beta \implies \exists \sigma \in \Sigma\ \partial_\sigma(\alpha) \neq \partial_\sigma(\beta).$$

Proof. Let $\alpha \not\equiv \beta$. We know that $\alpha = \sum_{i=1}^{n} w_i + \sum_{i=1}^{m} w'_i\alpha_i$ and $\beta = \sum_{i=1}^{n'} x_i + \sum_{i=1}^{m'} x'_i\beta_i$. The sets of partial derivatives of α and β w.r.t a $\sigma \in \Sigma$ can be written as:

$$\partial_\sigma(\alpha) = A \cup \bigcup_{t=1}^{j} \partial_\sigma(w_{i_t}) \cup \bigcup_{t=1}^{v} \partial_\sigma(w'_{l_t})\alpha_{l_t},$$
$$\partial_\sigma(\beta) = A \cup \bigcup_{t=1}^{j'} \partial_\sigma(x_{i'_t}) \cup \bigcup_{t=1}^{v'} \partial_\sigma(x'_{l'_t})\alpha_{l'_t},$$

where A is the set of all partial derivatives φ such that $\varphi \in \partial_\sigma(\gamma)$ if, and only if, γ is a common summand of α and β, i.e. if $\gamma \equiv w_i \equiv x_j$ or $\gamma \equiv w'_l\alpha_l \equiv x'_k\beta_k$ for some i, j, l, and k. Without loss of generality, consider the following three cases:

1. If $j \neq 0$ and $j' \neq 0$, we know that for $k \in \{i'_1, \ldots, i'_{j'}\}$, $w_{i_1} \neq x_k$ and, by

Lemma 5.10, $\partial_\sigma(w_{i_1}) \neq \partial_\sigma(x_k)$, and $\partial_\sigma(w_{i_1}) \neq \partial_\sigma(x'_k)\beta_k$, for $k \in \{l'_1, \ldots, l'_{v'}\}$. Thus, $\partial_\sigma(w_1) \cap \partial_\sigma(\beta) = \emptyset$.

2. If $j \neq 0$ and $j' = 0$, this case corresponds to the second part of the previous one.

3. If $j = j' = 0$, for $k \in \{l'_1, \ldots, l'_{v'}\}$, we have $w'_{l_1}\alpha_{l_1} \neq x'_k\alpha_k$ and then either $w'_{l_1} \neq x'_k$ or $\alpha_{l_1} \neq \beta_k$. If $w'_{l_1} \neq x'_k$ then $\partial_\sigma(w'_{l_1}) \neq \partial_\sigma(x'_k)$ and thus $\partial_\sigma(w'_{l_1})\alpha_{l_1} \neq \partial_\sigma(x'_k)\alpha_k$. If $\alpha_{l_1} \neq \beta_k$ it is obvious that $\partial_\sigma(w'_l)\alpha_l \neq \partial_\sigma(x'_k)\alpha_k$. Thus, $\partial_\sigma(w'_{l_1})\alpha_{l_1} \cap \partial_\sigma(\beta) = \emptyset$.

□

Theorem 5.12. *Let α be a irreducible finite regular expression modulo* ACIAL. *Then,* $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_b}$.

Proof. Let $\mathcal{A}_{\mathrm{PD}}(\alpha) = \langle \mathsf{PD}(\alpha), \Sigma, \delta_{pd}, \alpha, F_{pd}\rangle$. We want to prove that no pair of states of $\mathcal{A}_{\mathrm{PD}}(\alpha)$ is bisimilar. As in Proposition 5.8, we proceed by induction on the rank of the states. The only state in rank 0 is ε, for which the proposition is obvious. Suppose that all pair of states with rank $m < n$ are not bisimilar. Let $\gamma, \beta \in \mathsf{PD}(\alpha)$ with $n = rk(\gamma) \geq rk(\beta)$. Then, there exists $\gamma' \in \partial_\sigma(\gamma)$ that is distinct of every $\beta' \in \partial_\sigma(\beta)$, by Proposition 5.11. Because $rk(\beta') < n$ and $rk(\gamma') < n$, by inductive hypothesis, $\gamma' \not\equiv_b \beta'$. Thus $\gamma \not\equiv_b \beta$. □

Despite $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_b}$, for irreducible REs modulo ACIAL, these NFAs are not necessarily minimal. For example, if $\tau_3 = ba(a+b)+c(aa+ab)$, both NFAs have seven states, as can be seen in Figure 5.8, and a minimal equivalent NFA has four states.

Finally, note that for regular expressions representing finite languages, in general, the automaton ${\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_b}$ can be arbitrarily more succinct than $\mathcal{A}_{\mathrm{PD}}$. For example, considering the family of REs

$$\alpha_n = \sum_{i=1}^{n} a \sum_{j=1}^{i} a,$$

the $\mathcal{A}_{\mathrm{PD}}(\alpha_n)$ has $n+2$ states, and ${\mathcal{A}_{\mathrm{Pos}}(\alpha)}/{\equiv_b}$ has three states independently of n. Considering $n = 3$, $\mathcal{A}_{\mathrm{PD}}(\alpha_3)$ and ${\mathcal{A}_{\mathrm{Pos}}(\alpha_3)}/{\equiv_b}$ are represented in Figure 5.9.

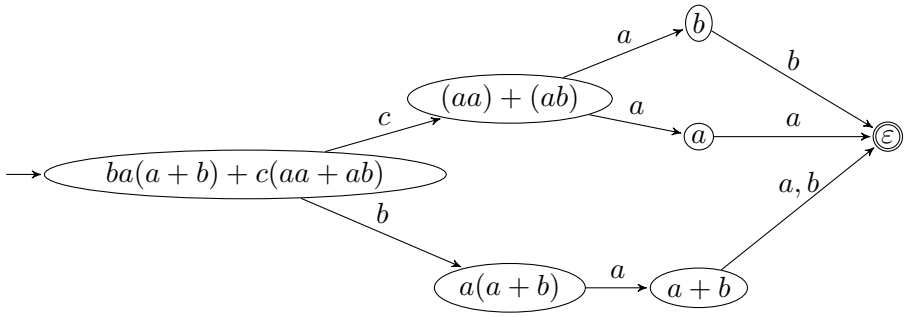
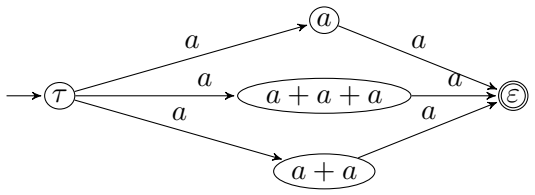
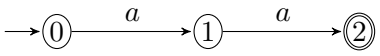


Figure 5.8: $\mathcal{A}_{\mathrm{PD}}(ba(a+b)+c(aa+ab)) \simeq {\mathcal{A}_{\mathrm{Pos}}(ba(a+b)+c(aa+ab)}/_{\equiv_b}$.



(a) $\mathcal{A}_{\mathrm{PD}}(\alpha_3)$.



(b) ${\mathcal{A}_{\mathrm{Pos}}(\alpha_3)}/_{\equiv_b}$.

Figure 5.9: $\alpha_3 = aa + a(a+a) + a(a+a+a)$.

5.1.4.2 Regular Languages

If we consider regular expressions with the Kleene star operator, it is easy to find REs α such that $\mathcal{A}_{\mathrm{PD}}(\alpha) \not\simeq {\mathcal{A}_{\mathrm{Pos}}(\alpha)}/_{\equiv_b}$. This is true even if $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ is a DFA, i.e., if α is one-unambiguous [BK93]. For example, for $\alpha = aa^\star + b(\varepsilon + aa^\star)$ the $\mathcal{A}_{\mathrm{PD}}(\alpha)$ has one more state than ${\mathcal{A}_{\mathrm{Pos}}(\alpha)}/_{\equiv_b}$. Ilie & Yu [IY03b] presented a family of REs

$$\alpha_n = (a+b+\varepsilon)(a+b+\varepsilon)\cdots(a+b+\varepsilon)(a+b)^\star,$$

where $(a+b+\varepsilon)$ is repeated n times, for which $\mathcal{A}_{\mathrm{PD}}(\alpha_n)$ has $n+1$ states and ${\mathcal{A}_{\mathrm{Pos}}(\alpha_n)}/_{\equiv_b}$ has one state independently of n. Considering $n = 3$ the $\mathcal{A}_{\mathrm{PD}}(\alpha_3)$ are represented in Figure 5.10.

In concurrency theory, the characterization of regular expressions for which equivalent NFAs are bisimilar has been extensively studied. Baeten et al. [BCG07] defined a normal form that corresponds to the normal form (5.5), in the finite case. For regular expressions with Kleene star operator the normal form defined by those authors is

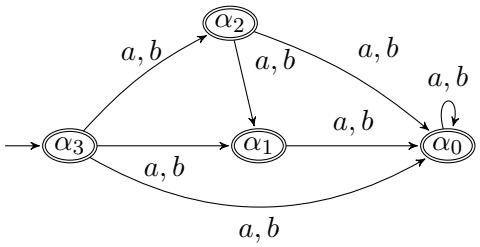


Figure 5.10: $\mathcal{A}_{\mathrm{PD}}((a+b+\varepsilon)(a+b+\varepsilon)(a+b+\varepsilon)(a+b)^\star)$.

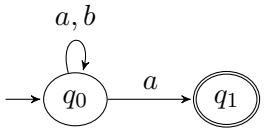


Figure 5.11: $\mathcal{A}_{\mathrm{Pos}}((ab^\star+b)^\star)_{/\equiv_b}$.

neither irreducible nor unique. In that case, we can find regular expressions α in normal form such that $\mathcal{A}_{\mathrm{PD}}(\alpha) \not\simeq \mathcal{A}_{\mathrm{Pos}}(\alpha)_{/\equiv_b}$. For example, for $\tau = (ab^\star + b)^\star$ the $\mathcal{A}_{\mathrm{PD}}(\tau)$ has three states, as seen before in Figure 2.11, while $\mathcal{A}_{\mathrm{Pos}}(\tau)_{/\equiv_b}$ has two states, as shown in Figure 5.11. Other example is $\tau_4 = a(\varepsilon + aa^\star) + ba^\star$, where $|\mathsf{PD}(\tau_4)| = 3$, and in $\mathcal{A}_{\mathrm{Pos}}(\tau_4)_{/\equiv_b}$ a state is saved because $(\varepsilon + aa^\star) \equiv_b a^\star$. This corresponds to an instance of one of the axioms of Kleene algebra (for the star operator).

As no confluent or even terminating rewrite system modulo these axioms is known, for general REs it will be difficult to obtain a characterization similar to the one of Theorem 5.12.

5.2 Right Derivative Automaton

Brzozowski proposed a conversion method from regular expressions to DFAs, based on derivatives of regular expressions, as we refer on Section 2.3.2. These derivatives can be named *left derivatives* because they denote a left quotient of a language. In this section we present the notion of right derivative and its relation with the left derivatives.

The *right derivative* of a regular expression α with respect to a symbol $\sigma \in \Sigma$, denoted $\alpha\sigma^{-1}$, is defined recursively on the structure of α as follows:

$$\begin{aligned} \emptyset\sigma^{-1} &= (\varepsilon)\sigma^{-1} = \emptyset, \\ \sigma'\sigma^{-1} &= \begin{cases} \{\varepsilon\} & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} \end{aligned} \qquad \begin{aligned} (\alpha+\beta)\sigma^{-1} &= (\alpha)\sigma^{-1} + (\beta)\sigma^{-1}, \\ (\alpha\beta)\sigma^{-1} &= \begin{cases} \alpha(\beta\sigma^{-1}) & \text{if } \varepsilon(\beta) \neq \varepsilon, \\ \alpha(\beta\sigma^{-1}) + \alpha\sigma^{-1} & \text{otherwise,} \end{cases} \\ (\alpha^\star)\sigma^{-1} &= \alpha^\star(\alpha\sigma^{-1}). \end{aligned} \tag{5.6}$$

This definition can be naturally extended to words in the following way, where $w \in \Sigma^\star$:

$$\begin{aligned} \alpha\varepsilon^{-1} &= \alpha, \\ \alpha(w\sigma)^{-1} &= (\alpha\sigma^{-1})w^{-1}. \end{aligned}$$

More generally we can use: $\alpha(ps)^{-1} = (\alpha s^{-1})p^{-1}$, for every factorisation $w = ps,\ p, s \in \Sigma^\star$.

The two following results establish a relationship between the right and the left derivatives, w.r.t. letters and words.

Proposition 5.13. *For any regular expression $\alpha \in RE$ and any $\sigma \in \Sigma$,*

$$\alpha\sigma^{-1} = (\sigma^{-1}\alpha^R)^R.$$

Proof. Let us prove the result by induction on α. For the base cases the result is obviously true. Assume that the equality holds for $\alpha_1, \alpha_2 \in RE$.

Let $\alpha \equiv \alpha_1 + \alpha_2$, then:

$$\begin{aligned} (\alpha_1+\alpha_2)\sigma^{-1} &= \alpha_1\sigma^{-1} + \alpha_2\sigma^{-1} \text{ by (5.6)} \\ &= (\sigma^{-1}\alpha_1^R)^R + (\sigma^{-1}\alpha_2^R)^R \text{ by inductive hypothesis} \\ &= (\sigma^{-1}\alpha_1^R + \sigma^{-1}\alpha_2^R)^R \text{ by (2.3)} \\ &= (\sigma^{-1}(\alpha_1^R + \alpha_2^R))^R \text{ by (2.21)} \end{aligned}$$

$$= (\sigma^{-1}(\alpha_1 + \alpha_2)^R)^R \text{ by (2.3).}$$

If $\alpha \equiv \alpha_1\alpha_2$, then we have:

$$(\alpha_1\alpha_2)\sigma^{-1} = \begin{cases} \alpha_1(\alpha_2\sigma^{-1}) & \text{If } \varepsilon(\alpha_2) \neq \varepsilon \\ \alpha_1(\alpha_2\sigma^{-1}) + \alpha_1\sigma^{-1} & \text{otherwise} \end{cases} \text{ by (5.6)}$$

$$= \begin{cases} \alpha_1(\sigma^{-1}\alpha_2^R)^R & \text{If } \varepsilon(\alpha_2) \neq \varepsilon \\ \alpha_1(\sigma^{-1}\alpha_2^R)^R + (\sigma^{-1}\alpha_1^R)^R & \text{otherwise} \end{cases} \text{ by inductive hypothesis}$$

$$= \begin{cases} ((\sigma^{-1}\alpha_2^R)\alpha_1^R)^R & \text{If } \varepsilon(\alpha_2) \neq \varepsilon \\ ((\sigma^{-1}\alpha_2^R)\alpha_1^R + \sigma^{-1}\alpha_1^R)^R & \text{otherwise} \end{cases} \text{ by (2.3)}$$

$$= (\sigma^{-1}(\alpha_2^R\alpha_1^R))^R = (\sigma^{-1}(\alpha_1\alpha_2)^R)^R \text{ by (2.21) and (2.3), respectively.}$$

Finally, if $\alpha \equiv \alpha_1^\star$, then:

$$\begin{aligned} \alpha_1^\star\sigma^{-1} &= \alpha_1^\star(\alpha_1\sigma^{-1}) \text{ by (5.6)} \\ &= \alpha_1^\star(\sigma^{-1}\alpha_1^R)^R \text{ by inductive hypothesis} \\ &= (\sigma^{-1}(\alpha_1^R)(\alpha_1^R)^\star)^R \text{ by (2.3)} \\ &= (\sigma^{-1}(\alpha_1^R)^\star)^R \text{ by (2.21)} \\ &= (\sigma^{-1}(\alpha_1^\star)^R)^R \text{ by (2.3).} \end{aligned}$$

□

Proposition 5.14. *For any regular expression $\alpha \in RE$ and any $w \in \Sigma^+$, $\alpha w^{-1} = ((w^R)^{-1}\alpha^R)^R$.*

Proof. Let us prove the result by induction on $|w|$. If $|w| = 1$, then $w = \sigma$. Thus, in this case, the equality is true by Proposition 5.13. Assuming that the equality holds for some $w \in \Sigma^+$, let us prove it for $w' = \sigma w$:

$$\begin{aligned} \alpha(\sigma w)^{-1} &= (\alpha w^{-1})\sigma^{-1} \\ &= ((w^R)^{-1}\alpha^R)^R\sigma^{-1} \text{ by inductive hypothesis} \end{aligned}$$

$$
\begin{aligned}
&= (\sigma^{-1}(((w^R)^{-1}\alpha^R)^R)^R)^R \text{ by Proposition 5.13}\\
&= (\sigma^{-1}((w^R)^{-1}\alpha^R))^R\\
&= ((w^R\sigma)^{-1}\alpha^R)^R \text{ by definition of derivatives}\\
&= (((\sigma w)^R)^{-1}\alpha^R)^R \text{ by (2.3).}
\end{aligned}
$$

□

Using these relations is not difficult to prove that:

Proposition 5.15. *For any regular expression $\alpha \in RE$ and any word $w \in \Sigma^\star$, $L(\alpha w^{-1}) = L(\alpha)w^{-1}$.*

Proof. It is known that $L(w^{-1}\alpha) = w^{-1}L(\alpha)$. Thus, we have:

$$
\begin{aligned}
L(\alpha w^{-1}) &= L(((w^R)^{-1}\alpha^R)^R), \text{ because } \alpha w^{-1} = ((w^R)^{-1}\alpha^R)^R\\
&= (L((w^R)^{-1}\alpha^R))^R\\
&= ((w^R)^{-1}L(\alpha^R))^R, \text{ because } L(w^{-1}\alpha) = w^{-1}L(\alpha)\\
&= L(\alpha)w^{-1}, \text{ because } Lw^{-1} = (w^R)^{-1}L^R.
\end{aligned}
$$

□

Let $\overleftarrow{\mathcal{D}}(\alpha)$ be the quotient of the set of all right derivatives of a regular expression α modulo the ACI-equivalence relation. Using the Proposition 5.13 and Proposition 5.14 we can easily conclude that:

Corollary 5.16. *For any regular expression $\alpha \in RE$, $\overleftarrow{\mathcal{D}}(\alpha) = (\mathcal{D}(\alpha^R))^R$.*

As we know that the set $\mathcal{D}(\alpha)$ of derivatives, modulo ACI-equivalence, is finite, by the Corollary 5.16 we can conclude that the set $\overleftarrow{\mathcal{D}}(\alpha)$ is also finite.

The *right derivative automaton* of a regular expression α is defined by

$$\overleftarrow{A}_{\mathcal{B}}(\alpha) = \langle \overleftarrow{\mathcal{D}}(\alpha), \Sigma, \delta, I, \{[\alpha]\}\rangle,$$

where $I = \{[d] \in \overleftarrow{\mathcal{D}}(\alpha) \mid \varepsilon(d) = \varepsilon\}$, and $\delta([q], \sigma) = \{[q'] \in \overleftarrow{\mathcal{D}}(\alpha) \mid [q'\sigma^{-1}] = [q]\}$.

Using the previous relations it is obvious that:

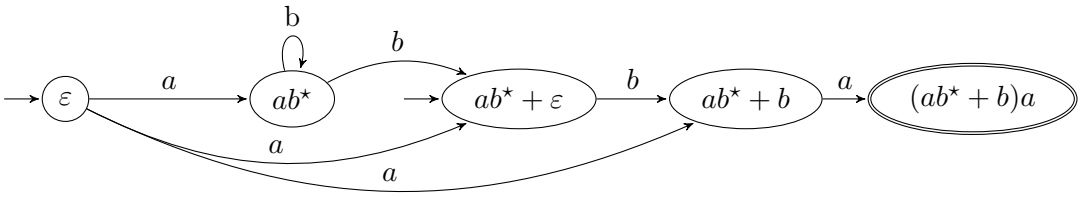


Figure 5.12: $\overleftarrow{A}_{\mathcal{B}}((ab^\star + b)a)$.

Corollary 5.17. *For any regular expression* $\alpha \in RE$, $\overleftarrow{A}_{\mathcal{B}}(\alpha) \simeq (A_{\mathcal{B}}(\alpha^R))^R$ *and* $L(\overleftarrow{A}_{\mathcal{B}}(\alpha)) = L(\alpha)$.

An NFA A is disjoint [Sen92] or a partial átomaton [BT14] if and only if A^R is deterministic. As $(\overleftarrow{A}_{\mathcal{B}}(\alpha))^R = A_{\mathcal{B}}(\alpha^R)$ and $A_{\mathcal{B}}(\alpha^R)$ is deterministic, for any regular expression $\alpha \in RE$, $\overleftarrow{A}_{\mathcal{B}}$ is a disjoint NFA or a partial átomaton. In Figure 5.12 is represented $\overleftarrow{A}_{\mathcal{B}}((ab^\star + b)a)$.

5.3 Right Partial Derivate Automaton

In the same way as for derivatives, the partial derivatives (defined in Section 2.3.2.2) can be called *left-partial derivatives*.

The concept of *right-partial derivative* was introduced by Champarnaud *et. al* [CDJM13]. For a regular expression $\alpha \in RE$ and a symbol $\sigma \in \Sigma$, the set of right-partial derivatives of α w.r.t. σ, $\overleftarrow{\partial}_\sigma(\alpha)$, is defined inductively as follows:

$$\begin{array}{rcl@{\qquad}rcl}
\overleftarrow{\partial}_\sigma(\emptyset), & = & \overleftarrow{\partial}_\sigma(\varepsilon) = \emptyset, & \overleftarrow{\partial}_\sigma(\alpha+\beta) & = & \overleftarrow{\partial}_\sigma(\alpha) \cup \overleftarrow{\partial}_\sigma(\beta), \\
\overleftarrow{\partial}_\sigma(\sigma') & = & \begin{cases} \{\varepsilon\} & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} & \overleftarrow{\partial}_\sigma(\alpha\beta) & = & \alpha\overleftarrow{\partial}_\sigma(\beta) \cup \varepsilon(\beta)\overleftarrow{\partial}_\sigma(\alpha), \\
 & & & \overleftarrow{\partial}_\sigma(\alpha^\star) & = & \alpha^\star\overleftarrow{\partial}_\sigma(\alpha).
\end{array} \tag{5.7}$$

The definition of right-partial derivative can be extended in a natural way to sets of regular expressions, words, and languages. The set of all right-partial derivatives of α

w.r.t. words is denoted by $\overleftarrow{\mathsf{PD}}(\alpha) = \bigcup_{w\in\Sigma^\star} \overleftarrow{\partial}_w(\alpha)$.

The next results relate the left and the right-partial derivatives.

Proposition 5.18. *For any regular expression α and any symbol $\sigma \in \Sigma$,*

$$(\partial_\sigma(\alpha^R))^R = \overleftarrow{\partial}_\sigma(\alpha).$$

Proof. Let us prove the result by induction on the structure of α. For the base cases the equality is obvious.

Let $\alpha \equiv \alpha_1 + \alpha_2$, then

$$\begin{aligned}(\partial_\sigma((\alpha_1+\alpha_2)^R))^R &= (\partial_\sigma(\alpha_1^R) \cup \partial_\sigma(\alpha_2^R))^R \\ &= (\partial_\sigma(\alpha_1^R))^R \cup (\partial_\sigma(\alpha_2^R))^R \\ &= \overleftarrow{\partial}_\sigma(\alpha_1) \cup \overleftarrow{\partial}_\sigma(\alpha_2) = \overleftarrow{\partial}_\sigma(\alpha).\end{aligned}$$

Let $\alpha \equiv \alpha_1\alpha_2$, then

$$\begin{aligned}(\partial_\sigma((\alpha_1\alpha_2)^R))^R &= (\partial_\sigma(\alpha_2^R\alpha_1^R))^R \\ &= (\partial_\sigma(\alpha_2^R)\alpha_1^R \cup \varepsilon(\alpha_2^R)\partial_\sigma(\alpha_1^R))^R \\ &= (\partial_\sigma(\alpha_2^R)\alpha_1^R)^R \cup (\varepsilon(\alpha_2^R)\partial_\sigma(\alpha_1^R))^R \\ &= \alpha_1\overleftarrow{\partial}_\sigma(\alpha_2) \cup \varepsilon(\alpha_2)\overleftarrow{\partial}_\sigma(\alpha_1) = \overleftarrow{\partial}_\sigma(\alpha).\end{aligned}$$

Let $\alpha \equiv \alpha_1^\star$

$$\begin{aligned}(\partial_\sigma((\alpha_1^\star)^R))^R = (\partial_\sigma((\alpha_1^R)^\star))^R &= (\partial_\sigma(\alpha_1^R)(\alpha_1^R)^\star)^R \\ &= \alpha_1^\star(\partial_\sigma(\alpha_1^R))^R = \alpha_1^\star\overleftarrow{\partial}_\sigma(\alpha_1) = \overleftarrow{\partial}_\sigma(\alpha).\end{aligned}$$

Thus the equality in the proposition holds. □

Proposition 5.19. *For any regular expression α and any word $w \in \Sigma^\star$, $(\partial_{w^R}(\alpha^R))^R = \overleftarrow{\partial}_w(\alpha)$.*

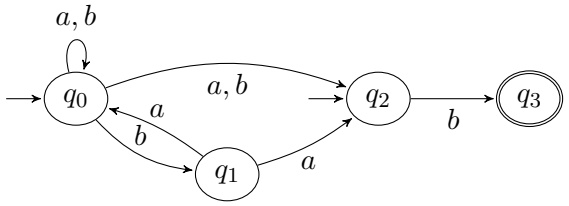


Figure 5.13: $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha) : q_0 = (a^\star b + a^\star ba + a^\star)^\star a^\star, q_1 = (a^\star b + a^\star ba + a^\star)^\star a^\star b, q_2 = (a^\star b + a^\star ba + a^\star)^\star, q_3 = (a^\star b + a^\star ba + a^\star)^\star b.$

Proof. Let us proceed by induction on the size of w. If $w = \varepsilon$ the result is obviously true. If $w = \sigma$ the result is true by Proposition 5.18. Assuming that the result is true for w, let us prove it for $w' = \sigma w$:

$$\begin{aligned}\overleftarrow{\partial}_{\sigma w}(\alpha) &= \overleftarrow{\partial}_{\sigma}(\overleftarrow{\partial}_{w}(\alpha)) = \overleftarrow{\partial}_{\sigma}((\partial_{w^R}(\alpha^R))^R) \\ &= (\partial_\sigma(((\partial_{w^R}(\alpha^R))^R)^R))^R = (\partial_\sigma(\partial_{w^R}(\alpha^R)))^R \\ &= (\partial_{w^R\sigma}(\alpha^R))^R = (\partial_{(\sigma w)^R}(\alpha^R))^R.\end{aligned}$$

□

Corollary 5.20. *For any regular expression α, $\overleftarrow{\mathsf{PD}}(\alpha) = (\mathsf{PD}(\alpha^R))^R$.*

Using the last result is not difficult to conclude that $\overleftarrow{\mathsf{PD}}$ is finite. The right partial derivative automaton of α is

$$\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha) = \langle \overleftarrow{\mathsf{PD}}(\alpha), \Sigma, \overleftarrow{\delta}_{pd}, \overleftarrow{F}_{pd}(\alpha), \alpha \rangle,$$

where $\overleftarrow{\delta}_{pd} = \{(q', \sigma, q) \mid q \in \overleftarrow{\mathsf{PD}}(\alpha),\ q' \in \overleftarrow{\partial}_{\sigma}(q), \text{ and } \sigma \in \Sigma\}$, $\overleftarrow{F}_{pd} = \{q \in \overleftarrow{\mathsf{PD}}(\alpha) \mid \varepsilon(q) = \varepsilon\}$. Note that $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$ has always just one final state although it can have more than one initial state. In Fig. 5.13 is represented the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(a^\star b + a^\star ba + a^\star)^\star b)$.

It is important to observe the two following propositions.

Lemma 5.21. *For any $\alpha \in RE$ and $w \in \Sigma^\star$, the following holds: $L(\overleftarrow{\partial}_{w}(\alpha)) = L(\alpha)w^{-1}$.*

Proof. We know that $L(\partial_w(\alpha)) = w^{-1}L(\alpha)$. Thus,

$$L(\overleftarrow{\partial}_{w}(\alpha)) = L((\partial_{w^R}(\alpha^R))^R) = (L(\partial_{w^R}(\alpha^R)))^R = ((w^R)^{-1}L(\alpha^R))^R = L(\alpha)w^{-1}.$$

□

Lemma 5.22. *For any $\alpha \in RE$ and $w \in \Sigma^\star$, the following holds: $\alpha w^{-1} = \sum \overleftarrow{\partial}_w(\alpha)$.*

Proof. It is known that $w^{-1}\alpha = \sum \partial_w(\alpha)$. Thus,

$$\begin{aligned}\sum \overleftarrow{\partial}_w(\alpha) &= \sum \left((\partial_{w^R}(\alpha^R))^R\right) = (\sum \partial_{w^R}(\alpha^R))^R \\ &= ((w^R)^{-1}\alpha^R)^R = \alpha w^{-1}.\end{aligned}$$

□

As what happens for $\mathcal{A}_{\mathrm{PD}}$, the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$ can also be defined inductively by a left system of expression equations, $\alpha_i = \alpha_{i1}\sigma_1 + \cdots + \alpha_{ik}\sigma_k + \varepsilon(\alpha_i)$, $i \in [0,n]$, $\alpha_0 \equiv \alpha$, $\alpha_{ij} \equiv \sum_{l \in I \subseteq [1,n]} \alpha_l$ is a linear combination of α_l, $l \in [1,n]$ and $j \in [1,k]$.

Proposition 5.23. *The set of regular expressions $\overleftarrow{\pi}(\alpha)$ is a solution of a left system of expression equations,*

$$\begin{aligned} &\overleftarrow{\pi}(\emptyset) = \emptyset, && \overleftarrow{\pi}(\alpha+\beta) = \overleftarrow{\pi}(\alpha) \cup \overleftarrow{\pi}(\beta), \\ &\overleftarrow{\pi}(\varepsilon) = \emptyset, && \overleftarrow{\pi}(\alpha\beta) = \alpha\overleftarrow{\pi}(\beta) \cup \overleftarrow{\pi}(\alpha), \\ &\overleftarrow{\pi}(\sigma) = \{\varepsilon\}, && \overleftarrow{\pi}(\alpha^\star) = \alpha^\star\overleftarrow{\pi}(\alpha). \end{aligned} \tag{5.8}$$

Proof. As $\alpha_i = \alpha_{i1}\sigma_1 + \cdots + \alpha_{ik}\sigma_k + \varepsilon(\alpha_i)$ and $\alpha_i^R = \sigma_1\alpha_{i1}^R + \cdots + \sigma_k\alpha_{ik}^R + \varepsilon(\alpha_i^R)$ the definition of $\overleftarrow{\pi}$ follows directly from the definition of π. □

The following result states a relation between the sets π and $\overleftarrow{\pi}$.

Proposition 5.24. *Let α be a regular expression. Then $(\pi(\alpha^R))^R = \overleftarrow{\pi}(\alpha)$.*

Proof. Let us proceed by induction on the structure of α. For $\alpha \equiv \varepsilon$, $\alpha \equiv \emptyset$ and $\alpha \equiv \sigma \in \Sigma$ it is obvious. Suppose that the equality is true for any subexpression of α, and let us prove that it is also true for α.

If $\alpha \equiv \alpha_1 + \alpha_2$, then

$$\begin{aligned}(\pi(\alpha^R))^R &= (\pi(\alpha_1^R) \cup \pi(\alpha_2^R))^R \\ &= (\pi(\alpha_1^R))^R \cup (\pi(\alpha_2^R))^R \\ &= \overleftarrow{\pi}(\alpha_1) \cup \overleftarrow{\pi}(\alpha_2) = \overleftarrow{\pi}(\alpha_1 + \alpha_2).\end{aligned}$$

If $\alpha \equiv \alpha_1\alpha_2$, then

$$\begin{aligned}(\pi((\alpha_1\alpha_2)^R))^R &= (\pi(\alpha_2^R\alpha_1^R))^R \\ &= (\pi(\alpha_2^R)\alpha_1^R \cup \pi(\alpha_1^R))^R \\ &= (\pi(\alpha_2^R)\alpha_1^R)^R \cup (\pi(\alpha_1^R))^R \\ &= (\alpha_1^R)^R(\pi(\alpha_2^R))^R \cup \overleftarrow{\pi}(\alpha_1) \\ &= \alpha_1\overleftarrow{\pi}(\alpha_2) \cup \overleftarrow{\pi}(\alpha_1) = \overleftarrow{\pi}(\alpha_1\alpha_2).\end{aligned}$$

If $\alpha \equiv \alpha_1^\star$, then

$$\begin{aligned}(\pi((\alpha_1^\star)^R))^R &= (\pi^(\alpha_1^R)^\star))^R \\ &= (\pi(\alpha_1^R)(\alpha_1^R)^\star)^R \\ &= ((\alpha_1^\star)^R)^R(\pi(\alpha_1^R))^R = \alpha_1^\star\overleftarrow{\pi}(\alpha_1) = \overleftarrow{\pi}(\alpha_1^\star).\end{aligned}$$

□

Note that the sizes of $\pi(\alpha)$ and $\overleftarrow{\pi}(\alpha)$ are not comparable in general. For example, if $\alpha = (a^\star b + a^\star ba + a^\star)^\star b$ then $|\pi(\alpha)| > |\overleftarrow{\pi}(\alpha)|$, but if we consider $\beta = b(ba^\star + aba^\star + a^\star)^\star$ then $|\pi(\beta)| < |\overleftarrow{\pi}(\beta)|$.

Corollary 5.25. *For any regular expression α, $\overleftarrow{\mathsf{PD}}(\alpha) = \overleftarrow{\pi}(\alpha) \cup \{\alpha\}$.*

Proof. For any regular expression $\alpha \in RE$ we know that

$$\begin{aligned} & \mathsf{PD}(\alpha) && = \pi(\alpha) \cup \{\alpha\} \\ \Leftrightarrow\ & \mathsf{PD}(\alpha^R) && = \pi(\alpha^R) \cup \{\alpha^R\}\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow\quad (\overleftarrow{\mathsf{PD}}(\alpha))^R &&= (\overleftarrow{\pi}(\alpha))^R \cup \{\alpha^R\} \text{ by Corollary 5.20 and Proposition 5.24}\\
&\Leftrightarrow\quad \overleftarrow{\mathsf{PD}}(\alpha) &&= \overleftarrow{\pi}(\alpha) \cup \{\alpha\}.
\end{aligned}$$

□

The solution of the system of equations also allows to inductively define the transition function. Let $\overleftarrow{\varphi}(\alpha) = \{(\gamma,\sigma) \mid \gamma \in \overleftarrow{\partial}_\sigma(\alpha), \sigma \in \Sigma\}$ and $\overleftarrow{\lambda}(\alpha) = \{\alpha' \mid \alpha' \in \overleftarrow{\pi}(\alpha),\ \varepsilon(\alpha') = \varepsilon\}$, where both sets can be inductively defined as follows:

$$\begin{aligned}
&\overleftarrow{\varphi}(\emptyset) = \emptyset, && \overleftarrow{\varphi}(\alpha+\beta) = \overleftarrow{\varphi}(\alpha) \cup \overleftarrow{\varphi}(\beta),\\
&\overleftarrow{\varphi}(\varepsilon) = \emptyset, && \overleftarrow{\varphi}(\alpha\beta) = \alpha\overleftarrow{\varphi}(\beta) \cup \varepsilon(\beta)\overleftarrow{\varphi}(\alpha),\\
&\overleftarrow{\varphi}(\sigma) = \{(\varepsilon,\sigma)\},\ \sigma \in \Sigma, && \overleftarrow{\varphi}(\alpha^\star) = \alpha^\star\overleftarrow{\varphi}(\alpha);
\end{aligned} \tag{5.9}$$

$$\begin{aligned}
&\overleftarrow{\lambda}(\emptyset) = \emptyset, && \overleftarrow{\lambda}(\alpha+\beta) = \overleftarrow{\lambda}(\alpha) \cup \overleftarrow{\lambda}(\beta),\\
&\overleftarrow{\lambda}(\varepsilon) = \emptyset, && \overleftarrow{\lambda}(\alpha\beta) = \varepsilon(\alpha)\alpha\overleftarrow{\lambda}(\beta) \cup \overleftarrow{\lambda}(\alpha),\\
&\overleftarrow{\lambda}(\sigma) = \{\varepsilon\},\ \sigma \in \Sigma, && \overleftarrow{\lambda}(\alpha^\star) = \alpha^\star\overleftarrow{\lambda}(\alpha).
\end{aligned}$$

The set of transitions is $\overleftarrow{\varphi}(\alpha) \times \{\alpha\} \cup \overleftarrow{F}(\alpha)$ where the set $\overleftarrow{F}$ is defined inductively by:

$$\begin{aligned}
\overleftarrow{F}(\emptyset) = \overleftarrow{F}(\varepsilon) = \overleftarrow{F}(\sigma) = \emptyset,\ \sigma \in \Sigma,\\
\overleftarrow{F}(\alpha+\beta) = \overleftarrow{F}(\alpha) \cup \overleftarrow{F}(\beta),\\
\overleftarrow{F}(\alpha\beta) = \alpha\overleftarrow{F}(\beta) \cup \overleftarrow{F}(\alpha) \cup \varphi(\alpha) \times (\alpha\overleftarrow{\lambda}(\beta)),\\
\overleftarrow{F}(\alpha^\star) = \alpha^\star\overleftarrow{F}(\alpha) \cup \alpha^\star(\overleftarrow{\varphi}(\alpha) \times \overleftarrow{\lambda}(\alpha)).
\end{aligned} \tag{5.10}$$

Note that the concatenation of a regular expression γ with a transition (α,σ,β) is defined by $\gamma(\alpha,\sigma,\beta) = (\gamma\alpha,\sigma,\gamma\beta)$, if $\gamma \notin \{\emptyset,\varepsilon\}$, $\emptyset(\alpha,\sigma,\beta) = \emptyset$ and $\varepsilon(\alpha,\sigma,\beta) = (\alpha,\sigma,\beta)$.

Proposition 5.26. *For all $\alpha \in RE$, we can also define the right-partial derivative*

automaton of α *as*

$$\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha) = \langle \overleftarrow{\pi}(\alpha) \cup \{\alpha\}, \Sigma, \overleftarrow{\varphi}(\alpha) \times \{\alpha\} \cup \overleftarrow{F}(\alpha), \overleftarrow{\lambda}(\alpha) \cup \varepsilon(\alpha)\{\alpha\}, \alpha \rangle.$$

Proof. Similar to the proof of Proposition 5.2. □

As we already mentioned, in Fig. 5.13 is represented the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(a^\star b + a^\star ba + a^\star)^\star b)$.

Let us define that $\forall \alpha \in RE, \sigma \in \Sigma, \{(\sigma, \alpha)\}^R = \{(\alpha^R, \sigma)\}$. The following results establish a relationship between the functions λ and $\overleftarrow{\lambda}$, φ and $\overleftarrow{\varphi}$, and F and $\overleftarrow{F}$.

Lemma 5.27. *Let* α *be a regular expression,* $(\lambda(\alpha^R))^R = \overleftarrow{\lambda}(\alpha)$, $(\varphi(\alpha^R))^R = \overleftarrow{\varphi}(\alpha)$ *and* $(F(\alpha^R))^R = \overleftarrow{F}(\alpha)$.

Note that, while $\lambda(\alpha)$ defines the set of final states of $\mathcal{A}_{\mathrm{PD}}(\alpha)$, $\overleftarrow{\lambda}(\alpha)$ defines the set of initial states of $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$.

Using the previous results we can relate $\mathcal{A}_{\mathrm{PD}}$ with $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$.

Proposition 5.28. *Let* α *be a regular expression. Then* $(\mathcal{A}_{\mathrm{PD}}(\alpha^R))^R \simeq \overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$.

Proof. Follows from the Proposition 5.24 and Lemma 5.27. □

Using the above result is not difficult to prove that:

Proposition 5.29. *Let* α *be a regular expression. Then* $L(\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)) = L(\alpha)$.

Proof. We know that $L(\alpha) = L(\mathcal{A}_{\mathrm{PD}}(\alpha))$. Thus,

$$\begin{aligned} L(\alpha) = L(\mathcal{A}_{\mathrm{PD}}(\alpha)) &\Leftrightarrow L(\alpha^R) = L(\mathcal{A}_{\mathrm{PD}}(\alpha^R)) \\ &\Leftrightarrow L((\alpha^R)^R) = L((\mathcal{A}_{\mathrm{PD}}(\alpha^R))^R) \\ &\Leftrightarrow L(\alpha) = L(\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)). \end{aligned}$$

□

As we know that $\mathcal{A}_{\mathrm{PD}}(\alpha) \simeq {}^{\mathcal{A}_{\mathrm{Pos}}(\alpha)}\!/_{\equiv_c}$ and $\mathcal{A}_{\mathrm{Prev}}(\alpha) \simeq (\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$ we can conclude that:

Corollary 5.30. *For any $\alpha \in RE$, $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha) \simeq \mathcal{A}_{\mathrm{Prev}}(\alpha)/_{\equiv_c}$.*

It is also not difficult to see that:

Proposition 5.31. *For any $\alpha \in RE$ the following hold:*

$$|\overleftarrow{\pi}(\alpha)| \leq |\alpha|_\Sigma, \tag{5.11}$$

$$|\overleftarrow{\mathsf{PD}}(\alpha)| \leq |\alpha|_\Sigma + 1. \tag{5.12}$$

Proof. Since $\overleftarrow{\mathsf{PD}}(\alpha) = \overleftarrow{\pi}(\alpha) \cup \{\alpha\}$, the first inequality implies the second one, thus we only need to prove (5.11). We proceed by induction on α. The base cases are obvious. Let us suppose that the inequality (5.11) holds for some $\alpha_1,\ \alpha_2 \in RE$ and consider three subcases. First, consider $\alpha \equiv \alpha_1 + \alpha_2$. Then, we have:

$$|\overleftarrow{\pi}(\alpha_1 + \alpha_2)| = |\overleftarrow{\pi}(\alpha_1) \cup \overleftarrow{\pi}(\alpha_2)| = |\overleftarrow{\pi}(\alpha_1)| + |\overleftarrow{\pi}(\alpha_2)| \leq |\alpha_1 + \alpha_2|_\Sigma.$$

For the second case, consider $\alpha \equiv \alpha_1\alpha_2$, then

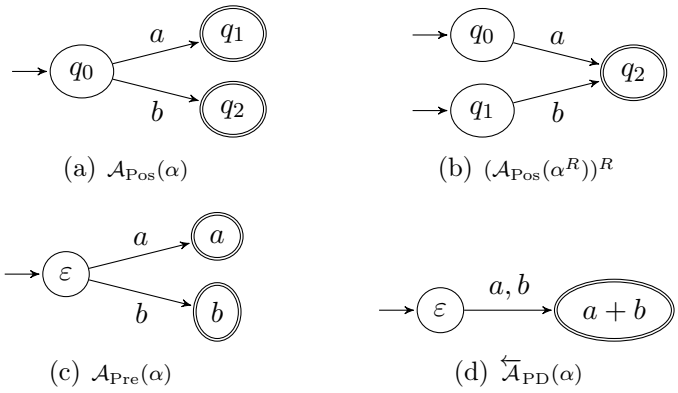
$$\begin{aligned} |\overleftarrow{\pi}(\alpha_1\alpha_2)| = |\alpha_1 \overleftarrow{\pi}(\alpha_2) \cup \overleftarrow{\pi}(\alpha_1)| &= |\alpha_1 \overleftarrow{\pi}(\alpha_2)| + |\overleftarrow{\pi}(\alpha_1)| \\ &\leq |\alpha_2|_\Sigma + |\alpha_1|_\Sigma = |\alpha_1\alpha_2|_\Sigma. \end{aligned}$$

Finally, consider $\alpha \equiv \alpha_1^\star$, thus we have that

$$|\overleftarrow{\pi}(\alpha_1^\star)| = |\alpha_1^\star \overleftarrow{\pi}(\alpha_1)| \leq |\alpha_1|_\Sigma = |\alpha_1^\star|_\Sigma. \qquad \square$$

5.4 Prefix Automaton ($\mathcal{A}_{\mathrm{Pre}}$)

Yamamoto [Yam14] presented an algorithm for converting a regular expression into an equivalent NFA A_Y. First, a labeled version of the usual Thompson NFA ($\mathcal{A}_{\mathsf{T}}$), $M = \langle Q, \Sigma, \delta, q_0, \{f\}, LP, LS\rangle$, is obtained, where each state $q \in Q$ is labeled with two regular expressions, one that corresponds to its left language, $LP(q)$, and the



(a) $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ (b) $(\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$

(c) $\mathcal{A}_{\mathrm{Pre}}(\alpha)$ (d) $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$

Figure 5.14: $\alpha = a + b$.

other to its right language, $LS(q)$. The states for which the in-transitions are labeled with a letter are called *sym-states*. The equivalence relations $\equiv_{pre}$ and $\equiv_{suf}$ are defined on the set of sym-states: for two states p, q, $p \equiv_{pre} q$ if and only if $LP(p) = LP(q)$; and $p \equiv_{suf} q$ if and only if $LS(p) = LS(q)$. The *prefix automaton* $\mathcal{A}_{\mathrm{Pre}}$ and the *suffix automaton* $\mathcal{A}_{\mathrm{Suf}}$ are the quotient automata of $\mathcal{A}_{\mathsf{T}}$ by these relations. The final automaton A_Y is a combination of these two. The author also shows that $\mathcal{A}_{\mathrm{Suf}}$ automaton coincides with $\mathcal{A}_{\mathrm{PD}}$. This is no surprise, since it is known that the result of the elimination of all ε-transitions of $\mathcal{A}_{\mathsf{T}}$ is the $\mathcal{A}_{\mathrm{Pos}}$.

In what follows we construct the $\mathcal{A}_{\mathrm{Pre}}$ automaton directly from the regular expression without the need to use the $\mathcal{A}_{\mathsf{T}}$ automaton. The relation between $\mathcal{A}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Suf}}$ could lead us to think that $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ coincides with $\mathcal{A}_{\mathrm{Pre}}$, but this is not the case. For instance, for $\alpha = a + b$, the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$ has 2 states and the $\mathcal{A}_{\mathrm{Pre}}(\alpha)$ has 3 states (see Fig. 5.14). Note that both automata are obtained from another automaton by merging the states with the same left language: while the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$ is obtained from $(\mathcal{A}_{\mathrm{Pos}}(\alpha^R))^R$, we are going to see that the $\mathcal{A}_{\mathrm{Pre}}(\alpha)$ is obtained from $\mathcal{A}_{\mathrm{Pos}}(\alpha)$.

Consider a system of left equations $\alpha_i = \alpha_{i1}\sigma_1 + \cdots + \alpha_{ik}\sigma_k$, $i \in [1, n]$, where $\alpha = \sum_{i \in I \subseteq [0,n]} \alpha_i$, $\alpha_{ij} \equiv \sum_{l \in I_{ij} \subseteq [0,n]} \alpha_l$ and $\alpha_0 \equiv \varepsilon$. Note that α_0 in α_{ik} for some $i \in [1, n]$, but α_0 is not in the solution set of the system of equations.

Proposition 5.32. *The set* $\mathsf{Pre}(\alpha)$ *inductively defined as:*

$$\begin{aligned} \mathsf{Pre}(\emptyset) &= \emptyset, & \mathsf{Pre}(\alpha+\beta) &= \mathsf{Pre}(\alpha) \cup \mathsf{Pre}(\beta), \\ \mathsf{Pre}(\varepsilon) &= \emptyset, & \mathsf{Pre}(\alpha\beta) &= \alpha\mathsf{Pre}(\beta) \cup \mathsf{Pre}(\alpha), \\ \mathsf{Pre}(\sigma) &= \{\sigma\}, & \mathsf{Pre}(\alpha^\star) &= \alpha^\star\mathsf{Pre}(\alpha), \end{aligned} \tag{5.13}$$

is a solution (left support) of the system of left equations defined above.

Proof. For $\alpha \equiv \emptyset$ or $\alpha \equiv \varepsilon$ is obvious that the solution is $\emptyset$. For $\alpha \equiv \sigma$,

$$\begin{aligned} \alpha &= \alpha_1, \\ \alpha_1 &= \alpha_0\sigma, \\ \alpha_0 &\equiv \varepsilon. \end{aligned}$$

Thus $\mathsf{Pre}(\alpha) = \{\sigma\}$.

Let us suppose that

$$\begin{aligned} \beta &= \sum_{i \in I \subseteq [0,n]} \beta_i, \\ \beta_i &= \beta_{i1}\sigma_1 + \cdots + \beta_{ik}\sigma_k, \end{aligned}$$

with $\mathsf{Pre}(\beta) = \{\beta_1, \ldots, \beta_n\}$ and

$$\begin{aligned} \gamma &= \sum_{i \in I' \subseteq [0,m]} \gamma_i, \\ \gamma_i &= \gamma_{i1}\sigma_1 + \cdots + \gamma_{ik}\sigma_k, \end{aligned}$$

with $\mathsf{Pre}(\gamma) = \{\gamma_1, \ldots, \gamma_m\}$.

Consider $\alpha \equiv \beta + \gamma$, then

$$\beta + \gamma = \sum_{i \in I \subseteq [1,n]} \beta_i + \sum_{i \in I' \subseteq [1,m]} \gamma_i.$$

As we need all $\beta_i,\ i \in [1,n]$ to define β, and all $\gamma_i,\ i \in [1,m]$ to define γ, $\mathsf{Pre}(\alpha) = \{\beta_1, \ldots, \beta_n\} \cup \{\gamma_1, \ldots, \gamma_m\}$. Consider $\alpha \equiv \beta\gamma$ then

$$\begin{aligned} \beta\gamma &= \beta(\sum_{i \in I \subseteq [0,m]} \gamma_i), \\ &= \begin{cases} \beta(\sum_{i \in I' \subseteq [1,m]} \gamma_i) & \text{If } 0 \notin I', \\ \beta(\sum_{i \in I' \subseteq [1,m]} \gamma_i) + \sum_{i \in I \subseteq [0,n]} \beta_i & \text{If } 0 \in I', \end{cases} \end{aligned}$$

and $\beta\gamma_i = \beta(\gamma_{i1}\sigma_1 + \ldots + \gamma_{ik}\sigma_k)$. As we know that $\gamma_0 \equiv \varepsilon$ occurs in γ_{ik} for some $i \in [0,m]$, the solution set is $\mathsf{Pre}(\alpha) = \{\beta\gamma_1, \ldots, \beta\gamma_m\} \cup \{\beta_1, \ldots, \beta_n\}$.

Consider $\alpha \equiv \beta^\star$ then

$$\begin{aligned} \beta^\star &= \beta^\star\beta + \varepsilon, \\ &= \beta^\star(\sum_{i \in I \subseteq [1,n]} \beta_i) + \varepsilon. \end{aligned}$$

Thus, $\mathsf{Pre}(\alpha) = \{\beta^\star\beta_1, \ldots, \beta^\star\beta_n\}$. □

The definition of Pre can be extended to sets of regular expressions: $\mathsf{Pre}(S) = \bigcup_{\alpha \in S} \mathsf{Pre}(\alpha)$ for $S \subseteq RE$.

The LP labelling scheme proposed by Yamamoto corresponds to the set Pre, i.e., considering that $\langle Q, \Sigma, \delta, q_0, \{f\}, LP, LS\rangle$ is the labeled version of $\mathcal{A}_{\mathsf{T}}(\alpha)$, $\bigcup_{q \in Q} LP(q) = \mathsf{Pre}(\alpha)$.

Remark 2. *For any $\alpha \in RE$, either $\mathsf{Pre}(\alpha)$ is $\emptyset$ or its elements are always of the form $\alpha'\sigma$, where α' is a subexpression of α, a concatenation of subexpressions of α or ε, and $\sigma \in \Sigma_\alpha$.*

Using the above system of equations, we can define the prefix automaton of a regular expression α as

$$\mathcal{A}_{\mathrm{Pre}}(\alpha) = \langle \mathsf{Pre}_0(\alpha), \Sigma, \{\varepsilon\} \times \psi(\alpha) \cup \mathsf{T}(\alpha), \varepsilon, \mathsf{Pr}'(\alpha) \cup \varepsilon(\alpha) \rangle,$$

where $\mathsf{Pre}_0(\alpha) = \mathsf{Pre}(\alpha) \cup \{\varepsilon\}$, $\mathsf{Pr}'(\alpha) = \{\alpha_i \mid i \in I \subseteq [0,n]\}$, $\psi(\alpha) = \{(\sigma_j, \alpha_i) \mid \varepsilon \in \alpha_{ij}, j \in [i,k]\}$, and $\mathsf{T}(\alpha) = \{(\alpha_i, \sigma_l, \alpha_j) \mid \alpha_i \in \alpha_{jl}, l \in [1,k]\}$.

The sets $\mathsf{Pr}'(\alpha)$, $\psi(\alpha)$ and $\mathsf{T}(\alpha)$ can also be inductively defined, respectively, as follows

$$\begin{aligned} &\mathsf{Pr}'(\emptyset) = \emptyset, && \mathsf{Pr}'(\alpha\beta) = \alpha\mathsf{Pr}'(\beta) \cup \varepsilon(\beta)\mathsf{Pr}'(\alpha), \\ &\mathsf{Pr}'(\varepsilon) = \varepsilon, && \mathsf{Pr}'(\alpha+\beta) = \mathsf{Pr}'(\alpha) \cup \mathsf{Pr}'(\beta), \\ &\mathsf{Pr}'(\sigma) = \{\sigma\}, && \mathsf{Pr}'(\alpha^\star) = \alpha^\star\mathsf{Pr}'(\alpha); \end{aligned} \tag{5.14}$$

$$\begin{aligned} &\psi(\emptyset) = \emptyset, && \psi(\alpha+\beta) = \psi(\alpha) \cup \psi(\beta), \\ &\psi(\varepsilon) = \emptyset, && \psi(\alpha\beta) = \psi(\alpha) \cup \varepsilon(\alpha)\ \alpha\ \psi(\beta), \\ &\psi(\sigma) = \{(\sigma,\sigma)\}, && \psi(\alpha^\star) = \alpha^\star\psi(\alpha); \end{aligned} \tag{5.15}$$

$$\begin{aligned} \mathsf{T}(\emptyset) = \mathsf{T}(\varepsilon) = \mathsf{T}(\sigma) &= \emptyset,\ \sigma \in \Sigma, \\ \mathsf{T}(\alpha+\beta) &= \mathsf{T}(\alpha) \cup \mathsf{T}(\beta), \\ \mathsf{T}(\alpha\beta) &= \mathsf{T}(\alpha) \cup \alpha\mathsf{T}(\beta) \cup \mathsf{Pr}'(\alpha) \times (\alpha\psi(\beta)), \\ \mathsf{T}(\alpha^\star) &= \alpha^\star\mathsf{T}(\alpha) \cup \alpha^\star(\mathsf{Pr}'(\alpha) \times \psi(\alpha)). \end{aligned} \tag{5.16}$$

Similarly to what happens for Pre, Pr' can be extended to sets of regular expressions: $\mathsf{Pr}'(S) = \bigcup_{\alpha \in S} \mathsf{Pr}'(\alpha)$ for $S \subseteq RE$. Note that $\mathcal{L}(\alpha) = \mathcal{L}(\mathsf{Pr}'(\alpha)) \cup \varepsilon(\alpha)$. In Figure 5.15 we can see the $\mathcal{A}_{\mathrm{Pre}}((a^\star b + a^\star ba + a^\star)^\star b)$.

By Remark 2, we know that the state labels of $\mathcal{A}_{\mathrm{Pre}}$ automaton have always the form $\alpha\sigma$, σ or ε, which correspond to the left language of each state, by the construction of $\mathcal{A}_{\mathrm{Pre}}$. Thus, it is obvious that given a state α and a symbol σ the following function

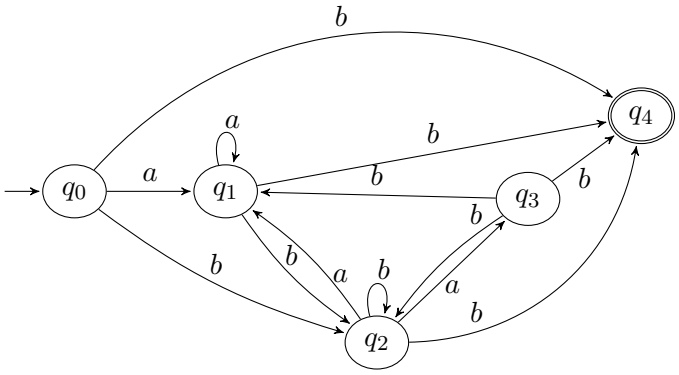


Figure 5.15: $\mathcal{A}_{\mathrm{Pre}}((a^\star b + a^\star ba + a^\star)^\star b) : q_0 = \varepsilon, q_1 = (a^\star b + a^\star ba + a^\star)^\star (a^\star a), q_2 = (a^\star b + a^\star ba + a^\star)^\star (a^\star b), q_3 = (a^\star b + a^\star ba + a^\star)^\star ((a^\star b)a), q_4 = (a^\star b + a^\star ba + a^\star)^\star b.$

calculates the predecessors of α by σ:

$$\begin{aligned} \mathsf{Pr}_\sigma(\emptyset) &= \mathsf{Pr}_\sigma(\varepsilon) = \emptyset, \\ \mathsf{Pr}_\sigma(\sigma') &= \begin{cases} \{\varepsilon\}, & \text{if } \sigma' = \sigma, \\ \emptyset, & \text{otherwise,} \end{cases} \end{aligned} \qquad \mathsf{Pr}_\sigma(\alpha'\sigma') = \begin{cases} \mathsf{Pr}'(\alpha') \cup \varepsilon(\alpha'), & \text{if } \sigma' = \sigma, \\ \emptyset, & \text{otherwise.} \end{cases} \tag{5.17}$$

The definition of Pr_σ can be naturally extended to sets of regular expressions, words, and languages. Given $\alpha \in RE$ and $\sigma \in \Sigma$, $\mathsf{Pr}_\sigma(S) = \bigcup_{\alpha \in S} \mathsf{Pr}_\sigma(\alpha)$ for $S \subseteq RE$, $\mathsf{Pr}_\varepsilon(\alpha) = \mathsf{Pr}'(\alpha)$ and $\mathsf{Pr}_{\sigma w}(\alpha) = \mathsf{Pr}_\sigma(\mathsf{Pr}_w(\alpha))$, for any $w \in \Sigma^\star, \sigma \in \Sigma$. Therefore, is not difficult to conclude that the automaton $\mathcal{A}_{\mathrm{Pre}}$ can also be inductively defined by

$$\mathcal{A}_{\mathrm{Pre}}(\alpha) = \langle \mathsf{Pre}_0(\alpha), \Sigma, \delta_{pre}, \varepsilon, \mathsf{Pr}'(\alpha) \cup \varepsilon(\alpha) \rangle,$$

where $\delta_{pre} = \{(s', \sigma, s) \mid s \in \mathsf{Pre}(\alpha),\ s' \in \mathsf{Pr}_\sigma(s),\ \sigma \in \Sigma\}$.

5.4.1 $\mathcal{A}_{\mathrm{Pre}}$ as $\mathcal{A}_{\mathrm{Pos}}$ Quotient

In the following we show that the $\mathcal{A}_{\mathrm{Pre}}(\alpha)$ is a quotient of $\mathcal{A}_{\mathrm{Pos}}(\alpha)$. If α is a linear regular expression, $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ is deterministic and thus all its states have distinct left languages. Therefore, in this case, $\mathcal{A}_{\mathrm{Pre}}(\alpha)$ coincides with $\mathcal{A}_{\mathrm{Pos}}(\alpha)$.

Proposition 5.33. *For any linear regular expression α, $|\mathsf{Pre}(\alpha)| = |\alpha|_\Sigma$.*

Proof. The proof proceeds by induction on α. For the base cases the result is obviously true. Assuming that the result holds for $\alpha_1, \alpha_2 \in \mathsf{PO}$, we prove it for the operations. Note that $\Sigma_{\alpha_1} \cap \Sigma_{\alpha_2} = \emptyset$, and because of that $\mathsf{Pre}(\alpha_1) \cap \mathsf{Pre}(\alpha_2) = \emptyset$. If $\alpha \equiv \alpha_1 + \alpha_2$, then $|\mathsf{Pre}(\alpha_1 + \alpha_2)| = |\mathsf{Pre}(\alpha_1) \cup \mathsf{Pre}(\alpha_2)|$. As $\mathsf{Pre}(\alpha_1) \cap \mathsf{Pre}(\alpha_2) = \emptyset$, $|\mathsf{Pre}(\alpha_1) \cup \mathsf{Pre}(\alpha_2)| = |\alpha_1|_\Sigma + |\alpha_2|_\Sigma = |\alpha_1 + \alpha_2|_\Sigma$. Considering $\alpha \equiv \alpha_1\alpha_2$ we have that $|\mathsf{Pre}(\alpha_1\alpha_2)| = |\alpha_1\mathsf{Pre}(\alpha_2) \cup \mathsf{Pre}(\alpha_1)|$. By the same reason of the previous case, $|\alpha_1\mathsf{Pre}(\alpha_2) \cup \mathsf{Pre}(\alpha_1)| = |\alpha_2|_\Sigma + |\alpha_1|_\Sigma = |\alpha_1\alpha_2|_\Sigma$. Finally, if $\alpha \equiv \alpha_1^\star$, then $|\mathsf{Pre}(\alpha_1^\star)| = |\alpha_1^\star\mathsf{Pre}(\alpha_1)| = |\alpha_1|_\Sigma = |\alpha_1^\star|_\Sigma$. □

Corollary 5.34. *For an arbitrary RE α, $\mathcal{A}_{\mathrm{Pre}}(\overline{\alpha}) \simeq \mathcal{A}_{\mathrm{Pos}}(\overline{\alpha})$.*

The following results show that the functions ψ, T and Pr' are related with the functions First, Follow and Last, respectively.

Proposition 5.35. *For any linear regular expression α, $\bigcup_{(\sigma,\alpha') \in \psi(\alpha)} \mathsf{Last}(\alpha') = \mathsf{First}(\alpha)$.*

Proof. Let us prove this result proceeding by induction on the structure of α. For $\alpha \equiv \emptyset$ and $\alpha \equiv \varepsilon$ the equality is obvious.

Considering $\alpha \equiv \sigma_i$, $\psi(\alpha) = \{(\sigma_i, \sigma_i)\}$. Thus, $\bigcup_{(\sigma,\alpha') \in \psi(\sigma_i)} \mathsf{Last}(\alpha') = \mathsf{Last}(\sigma_i) = \mathsf{First}(\sigma_i)$.

If $\alpha \equiv \alpha_1 + \alpha_2$, $\psi(\alpha_1 + \alpha_2) = \psi(\alpha_1) \cup \psi(\alpha_2)$. Then

$$\begin{aligned}\bigcup_{(\sigma,\alpha') \in \psi(\alpha_1+\alpha_2)} \mathsf{Last}(\alpha') &= \bigcup_{(\sigma,\alpha') \in \psi(\alpha_1)} \mathsf{Last}(\alpha') \cup \bigcup_{(\sigma,\alpha') \in \psi(\alpha_2)} \mathsf{Last}(\alpha') \\ &= \mathsf{First}(\alpha_1) \cup \mathsf{First}(\alpha_2) = \mathsf{First}(\alpha_1 + \alpha_2).\end{aligned}$$

For $\alpha \equiv \alpha_1\alpha_2$, $\psi(\alpha_2\alpha_2) = \psi(\alpha_1) \cup \varepsilon(\alpha_1)\alpha_1\psi(\alpha_2)$. Thus,

$$\bigcup_{(\sigma,\alpha')\in\psi(\alpha_1\alpha_2)} \mathsf{Last}(\alpha') = \bigcup_{(\sigma,\alpha')\in\psi(\alpha_1)} \mathsf{Last}(\alpha') \cup \varepsilon(\alpha_1) \bigcup_{(\sigma,\alpha')\in\alpha_1\psi(\alpha_2)} \mathsf{Last}(\alpha')$$

as by definition $\forall(\sigma,\beta') \in \psi(\alpha_2)\ \varepsilon(\beta') \neq \varepsilon$

$$= \mathsf{First}(\alpha_1) \cup \varepsilon(\alpha_1) \bigcup_{(\sigma,\alpha')\in\psi(\alpha_2)} \mathsf{Last}(\alpha')$$

$$= \mathsf{First}(\alpha_1) \cup \varepsilon(\alpha_1)\mathsf{First}(\alpha_2) = \mathsf{First}(\alpha_1\alpha_2).$$

If $\alpha \equiv \alpha_1^\star$, $\psi(\alpha_1^\star) = \alpha_1^\star\psi(\alpha_1)$. Then

$$\bigcup_{(\sigma,\alpha')\in\psi(\alpha_1^\star)} \mathsf{Last}(\alpha') = \bigcup_{(\sigma,\alpha')\in\alpha_1^\star\psi(\alpha_1)} \mathsf{Last}(\alpha')$$

$$= \bigcup_{(\sigma,\alpha')\in\psi(\alpha_1)} \mathsf{Last}(\alpha') = \mathsf{First}(\alpha_1) = \mathsf{First}(\alpha_1^\star).$$

□

Proposition 5.36. *For any linear regular expression α,* $\bigcup_{\alpha'\in\mathsf{Pr}'(\alpha)} \mathsf{Last}(\alpha') = \mathsf{Last}(\alpha)$.

Proof. The proof proceed by induction on the structure of α. For $\alpha \equiv \emptyset$ and $\alpha \equiv \varepsilon$ the equality is obvious. Considering $\alpha \equiv \sigma_i$, $\mathsf{Pr}'(\alpha) = \{\sigma_i\}$. Thus, $\bigcup_{\alpha'\in\mathsf{Pr}'(\sigma_i)} \mathsf{Last}(\alpha') = \mathsf{Last}(\sigma_i) = \mathsf{Last}(\sigma_i)$.

If $\alpha \equiv \alpha_1 + \alpha_2$, $\mathsf{Pr}'(\alpha_1 + \alpha_2) = \mathsf{Pr}'(\alpha_1) \cup \mathsf{Pr}'(\alpha_2)$. Then

$$\bigcup_{\alpha'\in\mathsf{Pr}'(\alpha_1+\alpha_2)} \mathsf{Last}(\alpha') = \bigcup_{\alpha'\in\mathsf{Pr}'(\alpha_1)} \mathsf{Last}(\alpha') \cup \bigcup_{\alpha'\in\mathsf{Pr}'(\alpha_2)} \mathsf{Last}(\alpha')$$

$$= \mathsf{Last}(\alpha_1) \cup \mathsf{Last}(\alpha_2) = \mathsf{Last}(\alpha_1 + \alpha_2).$$

For $\alpha \equiv \alpha_1\alpha_2$, $\mathsf{Pr}'(\alpha_2\alpha_2) = \alpha_1\mathsf{Pr}'(\alpha_2) \cup \varepsilon(\alpha_2)\mathsf{Pr}'(\alpha_1)$. Thus,

$$\bigcup_{\alpha'\in\mathsf{Pr}'(\alpha_1\alpha_2)} \mathsf{Last}(\alpha') = \bigcup_{\alpha'\in\alpha_1\mathsf{Pr}'(\alpha_2)} \mathsf{Last}(\alpha') \cup \varepsilon(\alpha_2) \bigcup_{\alpha'\in\mathsf{Pr}'(\alpha_1)} \mathsf{Last}(\alpha')$$

as by definition, if $\varepsilon \in \mathsf{Pr}'(\alpha_2)$ then $\varepsilon(\alpha_2) = \varepsilon$

$$= \bigcup_{\alpha' \in \mathsf{Pr}'(\alpha_2)} \mathsf{Last}(\alpha') \cup \varepsilon(\alpha_2) \bigcup_{\alpha' \in \mathsf{Pr}'(\alpha_1)} \mathsf{Last}(\alpha')$$
$$= \mathsf{Last}(\alpha_2) \cup \varepsilon(\alpha_2)\mathsf{Last}(\alpha_1) = \mathsf{Last}(\alpha_1\alpha_2).$$

If $\alpha \equiv \alpha_1^\star$, $\mathsf{Pr}'(\alpha_1^\star) = \alpha_1^\star \mathsf{Pr}'(\alpha_1)$. Then

$$\bigcup_{(\alpha' \in \mathsf{Pr}'(\alpha_1^\star)} \mathsf{Last}(\alpha') = \bigcup_{\alpha' \in \alpha_1^\star \mathsf{Pr}'(\alpha_1)} \mathsf{Last}(\alpha')$$
$$= \bigcup_{\alpha' \in \mathsf{Pr}'(\alpha_1)} \mathsf{Last}(\alpha') \cup \varepsilon(\alpha_1) \bigcup_{\alpha' \in \mathsf{Pr}'(\alpha_1^\star)} \mathsf{Last}(\alpha')$$
$$= \mathsf{Last}(\alpha_1) \cup \varepsilon(\alpha_1)\mathsf{Last}(\alpha_1^\star)$$
$$= \mathsf{Last}(\alpha_1) = \mathsf{Last}(\alpha_1^\star).$$

Thus the equality holds. □

Proposition 5.37. *For any linear regular expression α, and $\alpha_i, \alpha_j \in \mathsf{Pre}(\alpha)$,*

$$(\alpha_i, \sigma, \alpha_j) \in \mathsf{T}(\alpha) \Leftrightarrow (\mathsf{Last}(\alpha_i), \mathsf{Last}(\alpha_j)) \in \mathsf{Follow}(\alpha).$$

Proof. Note that as $\alpha_i, \alpha_j \in \mathsf{Pre}(\alpha)$, by Remark 2, $|\mathsf{Last}(\alpha_i)| = |\mathsf{Last}(\alpha_j)| = 1$. Let us define the function Follow (see (2.20) in page 27) in a different way:

$$\begin{aligned}
\mathsf{Follow}(\emptyset) &= \mathsf{Follow}(\varepsilon) = \mathsf{Follow}(\sigma_j) = \emptyset \\
\mathsf{Follow}(\alpha + \beta) &= \mathsf{Follow}(\alpha) \cup \mathsf{Follow}(\beta) \\
\mathsf{Follow}(\alpha\beta) &= \mathsf{Follow}(\alpha) \cup \mathsf{Follow}(\beta) \cup \mathsf{Last}(\alpha) \times \mathsf{First}(\beta) \\
\mathsf{Follow}(\alpha^\star) &= \mathsf{Follow}(\alpha) \cup \mathsf{Last}(\alpha) \times \mathsf{First}(\alpha).
\end{aligned}$$

Notice that the difference between this definition from [BMMR11] and the one in (2.20) is in the type of the functions ($RE \to \Sigma \times \mathsf{pos}_0$, in this case, and $RE \times \Sigma \to \mathsf{pos}_0$, in the other one). Using this Follow definition, the position automaton for α is $\mathcal{A}_{\mathrm{Pos}}(\alpha) = \langle \mathsf{pos}_0(\alpha), \Sigma, \delta_{pos}, 0, \mathsf{Last}_0(\alpha) \rangle$, with $\delta_{pos} = \{(0, \overline{\sigma_j}, j) \mid j \in \mathsf{First}(\alpha)\} \cup \{(i, \overline{\sigma_j}, j) \mid (i, j) \in \mathsf{Follow}(\alpha)\}$.

The result follows directly from this definition of Follow and the definition of T, and from the two previous propositions. □

Let $\overline{\mathcal{A}_{\text{Pre}}(\overline{\alpha})}$ be equal to $\mathcal{A}_{\text{Pre}}(\overline{\alpha})$, but with the letters in the transitions unmarked, then the following result holds.

Proposition 5.38. *Let α be a regular expression. Then $\overline{\mathcal{A}_{\text{Pre}}(\overline{\alpha})} \simeq \mathcal{A}_{\text{Pos}}(\alpha)$.*

Proof. To prove that these automata are isomorphic it is sufficient to consider the bijection $\kappa : \mathsf{Pre}_0 \to \mathsf{pos}_0$ such that for any $\gamma \in \mathsf{Pre}_0(\overline{\alpha})$, $\kappa(\varepsilon) = 0$, and $\kappa(\gamma) = \mathsf{Last}(\gamma)$, if $\gamma \not\equiv \varepsilon$. Note that by Remark 2 we can conclude that $|\mathsf{Last}(\gamma)| \leq 1$, for any $\gamma \in \mathsf{Pre}_0(\overline{\alpha})$. For initial and final states the isomorphism is obvious. Considering the transitions the isomorphism also holds by the Proposition 5.35 and Proposition 5.37. □

Let us define the equivalence relation $\equiv_l$ such that for any regular expression α, $\forall s_1, s_2 \in \mathsf{Pre}(\overline{\alpha}),\ s_1 \equiv_l s_2 \Leftrightarrow \overline{s_1} \equiv \overline{s_2}$.

Lemma 5.39. *The relation $\equiv_l$ is left-invariant.*

Proof. Follows directly from the construction of $\mathcal{A}_{\text{Pre}}$ automaton. □

From the previous results it is not difficult to conclude that $\mathcal{A}_{\text{Pre}}$ automaton is a quotient of $\mathcal{A}_{\text{Pos}}$.

Theorem 5.40. *Let α be a regular expression. Then $\mathcal{A}_{\text{Pre}}(\alpha) \simeq \overline{\mathcal{A}_{\text{Pre}}(\overline{\alpha})}/_{\equiv_l}$, and because of this $\mathcal{A}_{\text{Pre}}(\alpha) \simeq \mathcal{A}_{\text{Pos}}(\alpha)/_{\equiv_l}$.*

Proof. The first isomorphism is obvious by the system of equations. The second one is evident by Proposition 5.38. □

By construction, $\mathcal{A}_{\text{Pos}}$ is homogeneous, i.e. the transitions reaching each state are all labelled by the same letter. By Theorem 5.40 this also holds for $\mathcal{A}_{\text{Pre}}$.

Table 5.1: Experimental results for uniform random generated regular expressions: conversion methods.

k	$\lvert\alpha\rvert$	$\lvert\mathsf{pos}_0\rvert$	$\lvert\delta_{pos}\rvert$	$\lvert\mathsf{PD}\rvert$	$\lvert\delta_\pi\rvert$	$\frac{\lvert\pi\rvert}{\lvert\mathsf{pos}\rvert}$	$\lvert\overleftarrow{\mathsf{PD}}\rvert$	$\lvert\delta_{\overleftarrow{\pi}}\rvert$	$\frac{\lvert\overleftarrow{\pi}\rvert}{\lvert\mathsf{pos}\rvert}$	$\lvert\mathsf{Pre}_0\rvert$	$\lvert\delta_{pre}\rvert$	$\frac{\lvert\mathsf{Pre}\rvert}{\lvert\mathsf{pos}\rvert}$	$1-\eta_k$
2	100	28.9	167.5	15.7	56.0	0.55	15.9	56.4	0.55	20.1	73.7	0.71	0.90
	500	139.9	1486.5	71.6	389.8	0.51	71.5	393.1	0.51	91.9	530.8	0.66	
10	100	42.5	159.4	23.8	73.7	0.56	23.8	72.9	0.56	38.5	130.4	0.91	0.99
	500	207.1	1019.1	113.2	423.8	0.55	112.4	425.6	0.54	186	807.1	0.90	
	1000	412.1	2182.1	223.7	884.1	0.54	223.1	884.5	0.54	369.5	1717.6	0.90	

5.5 $\mathcal{A}_{\mathrm{Pos}}$, $\mathcal{A}_{\mathrm{PD}}$, $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Pre}}$ Automata: an Average-case Analysis

We conducted some experimental tests in order to compare the sizes of $\mathcal{A}_{\mathrm{Pos}}$, $\mathcal{A}_{\mathrm{PD}}$, $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Pre}}$ automata in practice. We used the FAdo library that includes implementations of those NFA conversions, and several tools for uniformly random generate regular expressions. In order to obtain regular expressions uniformly generated in the size of the syntactic tree, we used a prefix notation version of the grammar. For each alphabet size, k, and $|\alpha|$, samples of 10 000 REs were generated, which is sufficient to ensure a 95% confidence level within a 1% error margin. Table 5.1 presents the average values obtained for $|\alpha| \in \{100, 500, 1000\}$ and $k \in \{2, 10\}$.

These experiments suggest that, on average, the $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and the $\mathcal{A}_{\mathrm{PD}}$ have the same size and the $\mathcal{A}_{\mathrm{Pre}}$ is not significantly smaller than the $\mathcal{A}_{\mathrm{Pos}}$.

Broda et al. [BMMR11] studied the average size of $\mathcal{A}_{\mathrm{PD}}$ and concluded that, on average and asymptotically, the $\mathcal{A}_{\mathrm{PD}}$ has at most half the number of transitions of the $\mathcal{A}_{\mathrm{Pos}}$. By Proposition 5.28, $|\alpha^R|_\Sigma = |\alpha|_\Sigma$ and by the fact that $\varepsilon \in \pi(\alpha)$ if and only if $\varepsilon \in \overleftarrow{\pi}(\alpha)$, this analysis of the average size of $\mathcal{A}_{\mathrm{PD}}(\alpha)$ still holds for $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$. Thus the average sizes of $\mathcal{A}_{\mathrm{PD}}$ and $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ are asymptotically the same. However, $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}(\alpha)$ has only one final state and its number of initial states is the number of final states of $\mathcal{A}_{\mathrm{PD}}(\alpha^R)$.

Again following the ideas in Broda et al., we estimate the number of mergings of states

that arise when computing $\mathcal{A}_{\mathrm{Pre}}$ from $\mathcal{A}_{\mathrm{Pos}}$. The $\mathcal{A}_{\mathrm{Pre}}$ has at most $|\alpha|_\Sigma + 1$ states and this only occurs when all unions in $\mathsf{Pre}(\alpha)$ are disjoint. However for some cases this does not happen. For instance, when $\sigma \in \mathsf{Pre}(\beta) \cap \mathsf{Pre}(\gamma)$,

$$\begin{aligned} |\mathsf{Pre}(\beta+\gamma)| &= |\mathsf{Pre}(\beta) \cup \mathsf{Pre}(\gamma)| \leq |\mathsf{Pre}(\beta)| + |\mathsf{Pre}(\gamma)| - 1, \\ |\mathsf{Pre}(\beta^\star\gamma)| &= |\beta^\star\mathsf{Pre}(\gamma) \cup \beta^\star\mathsf{Pre}(\beta)| \leq |\mathsf{Pre}(\beta)| + |\mathsf{Pre}(\gamma)| - 1. \end{aligned} \tag{5.18}$$

In what follows, we estimate the number of these non-disjoint unions, which corresponds to a lower bound for the number of states merged in the $\mathcal{A}_{\mathrm{Pos}}$ automaton. This is done by the use of the methods of analytic combinatorics that was introduced in Section 3.2.1.

The regular expressions α_σ for which $\sigma \in \mathsf{Pre}(\alpha_\sigma)$, $\sigma \in \Sigma$ are generated by following grammar

$$\alpha_\sigma := \sigma \mid \alpha_\sigma + \alpha \mid \alpha_{\overline{\sigma}} + \alpha_\sigma \mid \alpha_\sigma \cdot \alpha \mid \varepsilon \cdot \alpha_\sigma.$$

The regular expressions that are not generated by α_σ are denoted by $\alpha_{\overline{\sigma}}$. The generating function for α_σ, $R_{\sigma,k}(z)$, satisfies

$$R_{\sigma,k}(z) = z + zR_{\sigma,k}(z)R_k(z) + z(R_k(z) - R_{\sigma,k}(z))R_{\sigma,k}(z) + zR_{\sigma,k}(z)R_k(z) + z^2R_{\sigma,k}(z)$$

that is equivalent to

$$zR_{\sigma,k}(z)^2 - (3zR_k(z) + z^2 - 1)R_{\sigma,k}(z) - z = 0. \tag{5.19}$$

From this one gets

$$R_{\sigma,k}(z) = \frac{(z^2 + 3zR_k(z) - 1) + \sqrt{(z^2 + 3zR_k(z) - 1)^2 + 4z^2}}{2z}. \tag{5.20}$$

As we know that $R_k(z) = \frac{1-z-\sqrt{\Delta_k(z)}}{4z}$, which is the generating function for REs given

by grammar (2.1) (omitting the $\emptyset$), one has

$$8zR_{\sigma,k}(z) = -b(z) - 3\sqrt{\Delta_k(z)} + \sqrt{a(z) + 6b(z)\sqrt{\Delta_k(z)} + 9\Delta_k(z)}, \tag{5.21}$$

where $a(z) = 16z^4 - 24z^3 + 65z^2 + 6z + 1$, $b(z) = -4z^2 + 3z + 1$, and $\Delta_k(z) = 1 - 2z - (7 + 8k)z^2$. Using the binomial theorem, we know that

$$\sqrt{a(z) + 6b(z)\sqrt{\Delta_k(z)} + 9\Delta_k(z)} = \sqrt{a(z)} + 3\frac{b(z)}{\sqrt{a(z)}}\sqrt{\Delta_k(z)} + o(\Delta_k(z)^{\frac{1}{2}}).$$

Thus,

$$8zR_{\sigma,k}(z) = -b(z) + \sqrt{a(z)} + 3\left(\frac{b(z)}{\sqrt{a(z)}} - 1\right)\sqrt{\Delta_k(z)} + o(\Delta_k(z)^{\frac{1}{2}}). \tag{5.22}$$

As we know that the following equalities are true:

$$\sqrt{\Delta_k(z)} = \sqrt{(7 + 8k)\rho_k(z - \overline{\rho}_k)}\sqrt{1 - z/\rho_k},$$

$$\sqrt{(7 + 8k)\rho_k(\rho_k - \overline{\rho}_k)} = \sqrt{2 - 2\rho_k},$$

and using the Proposition 3.2 and Lemma 3.3 (Section 3.2.1),

$$[z^n]R_{\sigma,k}(z) \sim \frac{3}{16\sqrt{\pi}}\left(1 - \frac{b(\rho_k)}{\sqrt{a(\rho_k)}}\right)\sqrt{2(1 - \rho_k)}\rho_k^{-(n+1)}n^{-\frac{3}{2}}. \tag{5.23}$$

Thus, the asymptotic ratio of regular expressions with $\sigma \in \mathsf{Pre}(\alpha)$ is:

$$\frac{[z^n]R_{\sigma,k}(z)}{[z^n]R_k(z)} \sim \frac{3}{2}\left(1 - \frac{b(\rho_k)}{\sqrt{a(\rho_k)}}\right). \tag{5.24}$$

As $\lim_{k\to\infty} \rho_k = 0$, $\lim_{k\to\infty} a(\rho_k) = 1$, and $\lim_{k\to\infty} b(\rho_k) = 1$, the asymptotic ratio of regular

expressions with $\sigma \in \mathsf{Pre}$ approaches 0 when $k \to \infty$.

Let $i(\alpha)$ be the number of non-disjoint unions appearing during the computation of $\mathsf{Pre}(\alpha),\ \alpha \in RE$ originated by the two cases described in (5.18). Then $i(\alpha)$ verifies the following equations

$$\begin{aligned}
i(\varepsilon) &= i(\sigma) = 0, & i(\alpha^\star_\sigma\alpha_\sigma) &= i(\alpha^\star_\sigma) + i(\alpha_\sigma) + 1,\\
i(\alpha_\sigma + \alpha_\sigma) &= i(\alpha_\sigma) + i(\alpha_\sigma) + 1, & i(\overline{\alpha^\star_\sigma}\alpha_\sigma) &= i(\overline{\alpha^\star_\sigma}) + i(\alpha_\sigma),\\
i(\alpha_\sigma + \alpha_{\overline{\sigma}}) &= i(\alpha_\sigma) + i(\alpha_{\overline{\sigma}}), & i(\alpha\alpha_{\overline{\sigma}}) &= i(\alpha) + i(\alpha_{\overline{\sigma}}),\\
i(\alpha_{\overline{\sigma}} + \alpha) &= i(\alpha_{\overline{\sigma}}) + i(\alpha), & i(\alpha^\star) &= i(\alpha).
\end{aligned}$$

From these equations we can obtain the cost generating function of the mergings, $I_a(z)$, by adding the contributions of each one of them. For example, the contribution of the regular expressions of the form $\alpha_\sigma + \alpha_\sigma$ can be computed as follows:

$$\begin{aligned}
\sum_{\alpha_\sigma+\alpha_\sigma} i(\alpha_\sigma + \alpha_\sigma) z^{|(\alpha_\sigma+\alpha_\sigma)|} &= z \sum_{\alpha_\sigma}\sum_{\alpha_\sigma} (i(\alpha_\sigma) + i(\alpha_\sigma) + 1) z^{|\alpha_\sigma|} z^{|\alpha_\sigma|}\\
&= z \sum_{\alpha_\sigma}\sum_{\alpha_\sigma} (i(\alpha_\sigma) + i(\alpha_\sigma)) z^{|\alpha_\sigma|} z^{|\alpha_\sigma|} + z \sum_{\alpha_\sigma}\sum_{\alpha_\sigma} z^{|\alpha_\sigma|} z^{|\alpha_\sigma|}\\
&= 2z I_{\alpha_\sigma,k}(z) R_{\sigma,k}(z) + z R_{\sigma,k}(z)^2
\end{aligned}$$

where $I_{\alpha_\sigma,k}(z)$ is the generating function for the mergings coming from α_σ. Applying this technique to the remaining cases, we obtain

$$I_a(z) = \frac{(z+z^2) R_{\sigma,k}(z)^2}{\sqrt{\Delta_k(z)}}. \tag{5.25}$$

Using again the same Proposition 3.2, we conclude that:

$$[z^n] I_a(z) \sim \frac{1+\rho_k}{64} \frac{\left(a(\rho_k) + b(\rho_k)^2 - 2b(\rho_k)\sqrt{a(\rho_k)}\right)}{\sqrt{\pi}\sqrt{2-2\rho_k}} \rho_k^{-(n+1)} n^{-\frac{1}{2}}. \tag{5.26}$$

Recall that the number of states of $\mathcal{A}_{\mathrm{Pos}}(\alpha)$ is equal to the number of letters in α. Thus, in order to obtain a lower bound for the reduction in the number of states of

the $\mathcal{A}_{\text{Pre}}$ automaton, as compared to the ones of the $\mathcal{A}_{\text{Pos}}$ automaton, it is enough to compare the number of mergings for an expression α, with the number of letters in α. Therefore, the asymptotic estimate for the average number of mergings is given by:

$$\frac{[z^n]I_\sigma(z)}{[z^n]L_k(z)} \sim \frac{1-\rho_k}{4\rho_k^2}\lambda_k = \eta_k, \tag{5.27}$$

where $\lambda_k = \frac{(1+\rho_k)}{16(1-\rho_k)}\left(a(\rho_k) + b(\rho_k)^2 - 2b(\rho_k)\sqrt{a(\rho_k)}\right)$. It is not difficult to conclude that $\lim_{k\to\infty} \lambda_k = 0$, therefore $\lim_{k\to\infty} \eta_k = 0$. In other words, the average number of states of the $\mathcal{A}_{\text{Pre}}$ automaton is equal to the number of states of the $\mathcal{A}_{\text{Pos}}$ automaton.

As it is evident from the last two columns of Table 5.1, for small values of k, the lower bound η_k does not capture all the mergings that occur in $\mathcal{A}_{\text{Pre}}$. However, it seems that for larger values of k, the average number of states of the $\mathcal{A}_{\text{Pre}}$ automaton approaches the number of states of the $\mathcal{A}_{\text{Pos}}$ automaton.

5.6 $\mathcal{A}_{\text{Pos}}$, $\mathcal{A}_{\text{PD}}$, $\mathcal{A}_{\text{Prev}}$ and $\overleftarrow{\mathcal{A}}_{\text{PD}}$ Determinization

Despite the fact that the DFAs obtained from regular expressions can be exponentially larger in size, sometimes we want to avoid the nondeterminism. We performed some experimental tests in order to compare the sizes of the DFAs obtained from the determinization of $\mathcal{A}_{\text{Pos}}$, $\mathcal{A}_{\text{PD}}$, $\mathcal{A}_{\text{Prev}}$ and $\overleftarrow{\mathcal{A}}_{\text{PD}}$ automata in practice. Recall that the determinization of $\mathcal{A}_{\text{Pos}}$ is the McNaughton & Yamada automaton (Section 2.3.1.2), and the determinization of $\mathcal{A}_{\text{Prev}}$ is the $\mathcal{A}_{dPrev}$ automaton (Section 2.3.1.3).

As in the previous section, we used the FAdo library, and for each alphabet size, k, and $|\alpha|$, samples of 10 000 uniformly random REs were generated. Table 5.2 presents the average values obtained for the number of states with $|\alpha| \in \{300, 500\}$ and $k \in \{2, 5, 10\}$. The measures $|Q_{d\mathcal{A}_{\text{Pos}}}|$, $|Q_{\mathcal{A}_{dPrev}}|$, $|Q_{d\mathsf{PD}}|$ and $|Q_{d\overleftarrow{\mathsf{PD}}}|$ represent the number of states of $D(\mathcal{A}_{\text{Pos}})$, $D(\mathcal{A}_{\text{Prev}})$, $D(\mathcal{A}_{\text{PD}})$ and $D(\overleftarrow{\mathcal{A}}_{\text{PD}})$, respectively.

The results suggest that $\mathcal{A}_{\text{Prev}}$ and $\mathcal{A}_{\text{Pos}}$ automata have the same size on average, as

Table 5.2: Experimental results for uniform random generated regular expressions: determinizations.

k	$\lvert\alpha\rvert$	$\lvert\mathsf{pos}_0\rvert$	$\lvert Q_{\mathrm{Prev}}\rvert$	$\lvert\mathsf{PD}\rvert$	$\lvert\overleftarrow{\mathsf{PD}}\rvert$	$\lvert Q_{d\mathcal{A}_{\mathrm{Pos}}}\rvert$	$\lvert Q_{\mathcal{A}_{dPrev}}\rvert$	$\lvert Q_{d\mathsf{PD}}\rvert$	$\lvert Q_{d\overleftarrow{\mathsf{PD}}}\rvert$
2	300	84.41	84.41	43.74	43.75	85.10	61.27	72.26	60.88
	500	139.82	139.82	71.60	71.61	202.91	146.58	172.33	146.21
5	300	110.75	110.75	60.49	60.49	312.35	244.38	276.86	243.43
10	300	124.70	124.70	68.20	68.20	162.98	120.55	127.59	119.55
	500	206.76	206.76	112.70	112.70	330.58	253.57	270.24	252.58

we expected because of Proposition 2.7. We can also observe that $D(\mathcal{A}_{\mathrm{Pos}})$ is greater than $D(\mathcal{A}_{\mathrm{Prev}})$. The automaton which results from the determinization of partial derivative automaton $D(\mathcal{A}_{\mathrm{PD}})$ is smaller than $D(\mathcal{A}_{\mathrm{Pos}})$, but greater than $D(\mathcal{A}_{\mathrm{Prev}})$. The automaton $D(\overleftarrow{\mathcal{A}}_{\mathrm{PD}})$ is the smallest one.

It is importante to note that there exist regular expressions for which all these DFAs have exponential size. For example, the family $(a^\star(ab^\star)^{l-1}a)^\star$, where $|\alpha|_\Sigma = 2l$, presented by Ellul et al. [EKSW04] as the worst-case lower bound from the conversion from RE to equivalent DFA, is a witness of that exponential growing.

Chapter 6

Conclusion

Descriptional complexity focus on the succinctness of the model representations. Over the last two decades, the study of the descriptional complexity of regular languages has become a major topic of research. In this work, we studied the descriptional complexity of some operations and simulations of regular models.

First of all, we presented tight upper bounds for the incomplete state and transition complexities for union, concatenation, Kleene star, complement and reversal on general and finite regular languages. Transition complexity bounds were expressed as functions of several more fine-grained measures of the operands, such as the number of final states, the number of undefined transitions or the number of transitions that leave the initial state. Table 4.1 summarises the results for incomplete transition complexity, using the witnesses parameters. Tables 4.2 and 4.3 summarise some of the results on state complexity and transition complexity of basic operations on general regular languages, respectively. In Table 4.2 we present the state complexity (sc), based on complete DFAs [YZS94]; the incomplete state complexity (isc), the new results here presented, and the ones from Gao et al. [GSY11]; and also, the results for state complexity for NFAs (nsc) [HK03]. The upper bound for the nondeterministic transition complexity of the complement is not tight, and thus, in Table 4.3, we inscribe the corresponding lower and upper bounds. Table 4.5 and Table 4.6 have the formulae

for the upper bounds of state and transition complexity for all the studied operations on finite regular languages.

The experimental results for both cases show that the upper bounds for state and transition complexities are much higher than the observed number of states and transitions of the DFAs resulting from the operations, with uniform random generated operands. Thus, although the study of the descriptional complexities considering the worst-case analysis is fundamental, in order to have good estimates of the amount of resources required to manipulate representations of a given language in practical applications, average-case complexity results need to be considered.

Posteriorly, we studied several methods of simulation of regular expressions by finite automata. Some of them were already known ($\mathcal{A}_{\mathrm{PD}}$, $\mathcal{A}_{\mathrm{Pos}}$, $\mathcal{A}_{MY}$ automata), other were introduced by us ($\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$, $\mathcal{A}_{\mathrm{Pre}}$, $\mathcal{A}_{\mathrm{Prev}}$ automata). We wanted to characterise direct constructions of small finite automaton from regular expressions. For that, we started to obtain a better characterisation of the $\mathcal{A}_{\mathrm{PD}}$ automaton, which is a quotient of the $\mathcal{A}_{\mathrm{Pos}}$ automaton. Considering finite languages, we presented a sufficient condition that specify the NFAs that are the partial derivative automaton of some finite regular expression. The $\mathcal{A}_{\mathrm{Pos}}$ bisimilarity is always not larger than all other quotients. We proved that, for regular expressions without Kleene star and under certain conditions, the $\mathcal{A}_{\mathrm{PD}}$ is an optimal conversion method, since the it is isomorphic to the position bisimilarity automaton.

The right-partial derivative automaton ($\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$) was introduced using the notion of right-partial derivatives, and we studied its relation with $\mathcal{A}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Pos}}$. We also presented a new construction of the $\mathcal{A}_{\mathrm{Pre}}$ automaton directly from the regular expression, without the use of an intermediary automaton. We showed that this automaton is also a quotient of $\mathcal{A}_{\mathrm{Pos}}$. The size of $\mathcal{A}_{\mathrm{Pos}}$ and $\mathcal{A}_{\mathrm{PD}}$ automata have already been studied, in both approaches, worst and average-case. As $\mathcal{A}_{\mathrm{PD}}$, $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{Pre}}$ are quotients of $\mathcal{A}_{\mathrm{Pos}}$ we know that, in the worst case, they have the same size of $\mathcal{A}_{\mathrm{Pos}}$. We showed that the average sizes of $\overleftarrow{\mathcal{A}}_{\mathrm{PD}}$ and $\mathcal{A}_{\mathrm{PD}}$ automata are asymptotically the same. We also showed that the average number of states of $\mathcal{A}_{\mathrm{Pre}}$ automaton approaches the number

of states of the $\mathcal{A}_{\text{Pos}}$ automaton. Thus, it seems that $\mathcal{A}_{\text{PD}}$ and $\overleftarrow{\mathcal{A}}_{\text{PD}}$ are, on average, the better methods of conversion from REs to NFAs w.r.t. the size of the resulting NFA. However, if we determinize the resulting NFA, $D(\overleftarrow{\mathcal{A}}_{\text{PD}})$ seems to be the smallest automaton obtained from the referred conversion methods.

There are several methods of conversion from REs to DFAs, for instance the Brzozowski automaton. As future work, it would be important to compare the size of the DFAs resulting from these methods, with the ones resulting from the determinization of $\mathcal{A}_{\text{Pos}}$, $\mathcal{A}_{\text{Prev}}$, $\mathcal{A}_{\text{PD}}$ and $\overleftarrow{\mathcal{A}}_{\text{PD}}$, in order to analyse the better method of conversion from REs to DFAs w.r.t. the size of the resulting automaton. An interesting approach would be to study the average-case complexity of these conversion methods using the analytic combinatorics framework.

Bibliography

[AMR07] M. Almeida, N. Moreira, and R. Reis. Enumeration and generation with a string automata representation. *Theor. Comput. Sci.*, 387(2):93–102, 2007.

[Ant96] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.

[BCG07] J. C. M. Baeten, F. Corradini, and C. A. Grabmayer. A characterization of regular expressions under bisimulation. *J. ACM*, 54(2), 2007.

[BGN10] F. Bassino, L. Giambruno, and C. Nicaud. The average state complexity of rational operations on finite languages. *Int. J. Found. Comput. Sci.*, 21(4):495–516, 2010.

[BHK09] H. Bordihn, M. Holzer, and M. Kutrib. Determination of finite automata accepting subregular languages. *Theor. Comput. Sci.*, 410(35):3209–3222, 2009.

[BK93] A. Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 48:197–213, 1993.

[BKW97] A. Brüggemann-Klein and D. Wood. The validation of SGML content models. *Mathematical and Computer Modelling*, 25(4):73 – 84, 1997.

[BMMR11] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the average state complexity of partial derivative automata. *Int. J. Found. Comput. Sci.*, 22(7):1593–1606, 2011.

[BMMR12] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the average size of Glushkov and partial derivative automata. *Int. J. Found. Comput. Sci.*, 23(5):969–984, 2012.

[BMMR14] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. A hitchhiker's guide to descriptional complexity through analytic combinatorics. *Theor. Comput. Sci.*, 528:85–100, 2014.

[Brz64] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964.

[BS86] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(1):117–126, December 1986.

[BT14] J. A. Brzozowski and H. Tamm. Theory of átomata. *Theor. Comput. Sci.*, 539:13–27, 2014.

[CCSY01] C. Câmpeanu, K. Culik II, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In O. Boldt and H. Jürgensen, editors, *Proceedings of 4th WIA*, volume 2214 of *LNCS*, pages 60–70. Springer, 2001.

[CDJM13] J. Champarnaud, J. Dubernard, H. Jeanne, and L. Mignot. Two-sided derivatives for regular expressions and for Hairpin expressions. In A. H. Dediu, C. Martín-Vide, and B. Truthe, editors, *Proceedings of 7th LATA*, volume 7810 of *LNCS*, pages 202–213. Springer, 2013.

[CL06] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems.* Springer, 2006.

[COZ04] J. Champarnaud, F. Ouardi, and D. Ziadi. Follow automaton versus equation automaton. In L. Ilie and D. Wotschke, editors, *Proceedings of 6th DCFS*, volume Report No. 619, pages 145–153, 2004.

[CZ00] P. Caron and D. Ziadi. Characterization of Glushkov automata. *Theor. Comput. Sci.*, 233(1-2):75–90, 2000.

[CZ01] J. Champarnaud and D. Ziadi. From Mirkin's prebases to Antimirov's word partial derivatives. *Fundam. Inform.*, 45(3):195–205, 2001.

[CZ02] J. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theor. Comput. Sci.*, 289(1):137–163, 2002.

[DoCS] Aarhus University Department of Computer Science. MONA. http://www.brics.dk/mona/index.html.

[DS07] M. Domaratzki and K. Salomaa. Transition complexity of language operations. *Theor. Comput. Sci.*, 387(2):147–154, 2007.

[DW11] J. Daciuk and D. Weiss. Smaller representation of finite state automata. In B. Bouchou-Markhoff, P. Caron, J. Champarnaud, and D. Maurel, editors, *Proceedings of 16th CIAA*, volume 6807 of *LNCS*, pages 118–129. Springer, 2011.

[EKSW04] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *J. Autom. Lang. Comb.*, 9(2-3):233–256, September 2004.

[Ell02] K. Ellul. Descriptional complexity measures of regular languages, master thesis. *University of Waterloo, Ont., Canada*, 2002.

[Emi] B. Emir. AMoRE: Automata, monoids, and regular expressions. http://amore.sourceforge.net/.

[FN13] S. De Felice and C. Nicaud. Brzozowski algorithm is generically super-polynomial for deterministic automata. In M. Béal and O. Carton, editors, *Proceedings of 17th DLT*, volume 7907 of *LNCS*, pages 179–190. Springer, 2013.

[FN14] S. Felice and C. Nicaud. On the average complexity of brzozowski's algorithm for deterministic automata with a small number of final states. In A. M. Shur and M. V. Volkov, editors, *Proceedings of 18th DLT*, volume 8633 of *LNCS*, pages 25–36. Springer, 2014.

[FS08] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. CUP, 2008.

[GH07] H. Gruber and M. Holzer. On the average state and transition complexity of finite languages. *Theor. Comput. Sci.*, 387(2):155–166, 2007.

[GLRA11] P. García, D. López, J. Ruiz, and G. I. Alvarez. From regular expressions to smaller nfas. *Theor. Comput. Sci.*, 412(41):5802–5807, 2011.

[Glu61] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1–53, 1961.

[GMR10] H. Gouveia, N. Moreira, and R. Reis. Small nfas from regular expressions: Some experimental results. *CoRR*, abs/1009.3599, 2010.

[Gro] The GAP Group. GAP. http://www.gap-system.org/gap.html.

[GSY11] Y. Gao, K. Salomaa, and S. Yu. Transition complexity of incomplete DFAs. *Fundam. Inform.*, 110(1-4):143–158, 2011.

[Gul13] S. Gulan. Series parallel digraphs with loops - graphs encoded by regular expression. *Theory Comput. Syst.*, 53(2):126–158, 2013.

[HK02] M. Holzer and M. Kutrib. Unary language operations and their nondeterministic state complexity. In M. Ito and M. Toyama, editors, *Proceedings of 6th DLT*, volume 2450 of *LNCS*, pages 162–172. Springer, 2002.

[HK03] M. Holzer and M. Kutrib. State complexity of basic operations on nondeterministic finite automata. In J. Champarnaud and D. Maurel, editors, *Proceedings of 7th CIAA*, volume 2608 of *LNCS*, pages 148–157. Springer, 2003.

[HK09a] M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata. In A. H. Dediu, A. Ionescu, and C. Martín-Vide, editors, *Proceedings of 3rd LATA*, volume 5457 of *LNCS*, pages 23–42. Springer, 2009.

[HK09b] M. Holzer and M. Kutrib. Nondeterministic finite automata - recent results on the descriptional and computational complexity. *Int. J. Found. Comput. Sci.*, 20(4):563–580, 2009.

[HK11] M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata—a survey. *Inf. Comput.*, 209(3):456 – 470, 2011.

[HS08] Y. Han and K. Salomaa. State complexity of union and intersection of finite languages. *Int. J. Found. Comput. Sci.*, 19(3):581–595, 2008.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[IY03a] L. Ilie and S. Yu. Follow automata. *Inf. Comput.*, 186(1):140–162, 2003.

[IY03b] L. Ilie and S. Yu. Reducing nfas by invariant equivalences. *Theor. Comput. Sci.*, 306(1-3):373–390, 2003.

[Jir05] G. Jirásková. State complexity of some operations on binary regular languages. *Theor. Comput. Sci.*, 330(2):287–298, 2005.

[JS11] G. Jirásková and J. Sebej. Note on reversal of binary regular languages. In M. Holzer, M. Kutrib, and G. Pighizzini, editors, *Proceedings of 13th DCFS*, volume 6808 of *LNCS*, pages 212–221. Springer, 2011.

[Kam] S. Kampakis. AGL: Artificial grammar learning. http://sourceforge.net/projects/aglsuite/.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[Koz97] Dexter Kozen. *Automata and computability.* Undergraduate texts in computer science. Springer, 1997.

[Lup66] O. Lupanov. A comparison of two types of finite sources. *Problemy Kibernetiki 9, 321–326, In Russian. German translation: Über den Vergleich zweier Typen endlicher Quellen. Probleme der Kybernetik*, 6:328–335, 1966.

[lV] University Paris-Est Marne la Vallée. Unitex. http://www-igm.univ-mlv.fr/ unitex/.

[Mas70] A. Maslov. Estimates of the number of states of finite automata. *Dokllady Akademii Nauk SSSR 194, 1266–1268, In Russian. English trans- lation in Soviet Mathematics Doklady*, 11:1373–1375, 1970.

[Mir66] B. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5:110–116, 1966.

[MMR13a] E. Maia, N. Moreira, and R. Reis. Incomplete transition complexity of basic operations on finite languages. In S. Konstantinidis, editor, *Proceedings of 18th CIAA*, volume 7982 of *LNCS*, pages 349–356. Springer, 2013.

[MMR13b] E. Maia, N. Moreira, and R. Reis. Incomplete transition complexity of some basic operations. In P. van Emde et al., editor, *Proceedings of 39th SOFSEM*, volume 7741 of *LNCS*, pages 319–331. Springer, 2013.

[MMR14] E. Maia, N. Moreira, and R. Reis. Partial derivative and position bisimilarity automata. In M. Holzer and M. Kutrib, editors, *Proceedings of 19th CIAA*, volume 8587 of *LNCS*, pages 264–277. Springer, 2014.

[MMR15a] E. Maia, N. Moreira, and R. Reis. Incomplete operational transition complexity of regular languages. *Inf. Comput.*, 244:1 – 22, 2015.

[MMR15b] E. Maia, N. Moreira, and R. Reis. Prefix and right-partial derivative automata. In A. Beckmann, V. Mitrana, and M. Soskova, editors, *Proceedings of CiE 2015*, volume 9136 of *LNCS*, pages 258–267. Springer, 2015.

[Moo71] F. R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic and two-way finite automata by deterministic automata. *IEEE Trans. Computers*, 20, 1971.

[MR] N. Moreira and R. Reis. FAdo: Tools for formal languages manipulation. http://fado.dcc.fc.up.pt/.

[MR09] N. Moreira and R. Reis. Series-parallel automata and short regular expressions. *Fundam. Inform.*, 91(3-4):611–629, 2009.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of 13th Swat*, pages 125–129, Washington, DC, USA, 1972. IEEE Computer Society.

[MY60] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–47, 1960.

[Nic99] C. Nicaud. Average state complexity of operations on unary automata. In M. Kutylowski, L. Pacholski, and T. Wierzbicki, editors, *Proceedings of 24th MFCS*, volume 1672 of *LNCS*, pages 231–240. Springer, 1999.

[Nic09] C. Nicaud. On the average size of Glushkov's automata. In A. H. Dediu, A. Ionescu, and C. Martín-Vide, editors, *Proceedings of 3rd LATA*, volume 5457 of *LNCS*, pages 626–637, 2009.

[oPEI] University of Prince Edward Island. Grail+. http://www.csit.upei.ca/ ccampeanu/grail/.

[ORT09] S. Owens, J. H. Reppy, and A. Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, 2009.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[RFL] S. Rodger, T. Finley, and P. Linz. JFLAP: Java formal languages and automata package. http://www.jflap.org/.

[RS59] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Research Development*, 3(2):114–169, 1959.

[Sak09] J. Sakarovitch. *Elements of Automata Theory.* CUP, 2009.

[Sal66] A. Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, January 1966.

[Sal07] K. Salomaa. Descriptional complexity of nondeterministic finite automata. In T. Harju, J. Karhumäki, and A. Lepistö, editors, *Proceedings of 11th DLT*, volume 4588 of *LNCS*, pages 31–35. Springer, 2007.

[Seb10] J. Sebej. Reversal of regular languages and state complexity. In D. Pardubská, editor, *Proceedings of ITAT*, volume 683 of *CEUR Workshop Proceedings*, pages 47–54. CEUR-WS.org, 2010.

[Sen92] H. Sengoku. Minimization of nondeterministic finite automata. Master's thesis, Kyoto University, 1992.

[Sha08] J. Shallit. *A Second Course in Formal Languages and Automata Theory.* CUP, 2008.

[SL] J. Sakarovitch and S. Lombardy. VAUCANSON. http://www.vaucanson-project.org/.

[Tho68] K. Thompson. Regular expression search algorithm. *Com. ACM*, 11(6):410–422, 1968.

[Yam14] H. Yamamoto. A new finite automaton construction for regular expressions. In S. Bensch, R. Freund, and F. Otto, editors, *Proceedings of 6th NCMA*, volume 304 of *books@ocg.at*, pages 249–264. ÖCG, 2014.

[YG11] S. Yu and Y. Gao. State complexity research and approximation. In G. Mauri and A. Leporati, editors, *Proceedings of 15th DLT*, volume 6795 of *LNCS*, pages 46–57. Springer, 2011.

[Yu97] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.

[Yu01] S. Yu. State complexity of regular languages. *J. of Autom., Lang. and Comb.*, 6(2):221–234, 2001.

[Yu05] S. Yu. State complexity: Recent results and open problems. *Fundam. Inform.*, 64(1-4):471–480, 2005.

[Yu06] S. Yu. On the state complexity of combined operations. In O. H. Ibarra and H. Yen, editors, *Proceedings of 11th CIAA*, volume 4094 of *LNCS*, pages 11–22. Springer, 2006.

[YZS94] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2):315–328, 1994.

Alphabetical Index