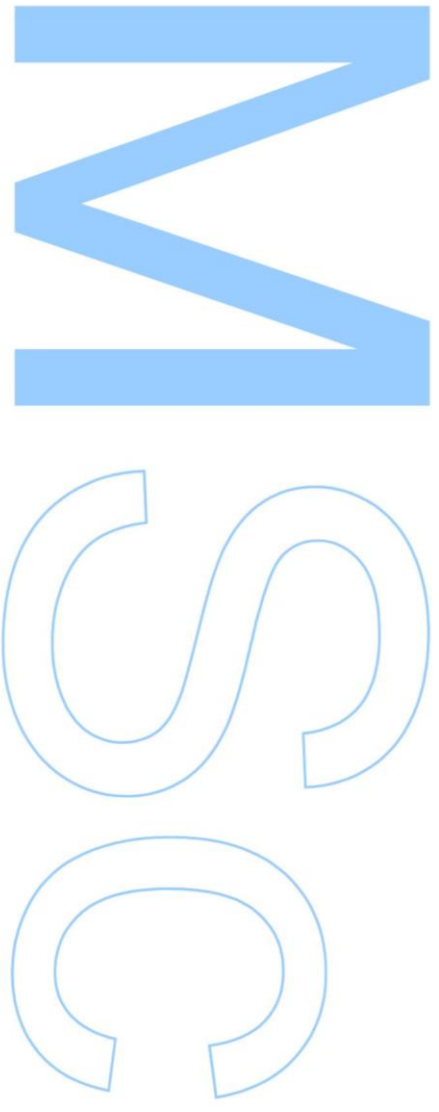
Manipulation of Extended Regular Expressions with Derivatives

Rafaela Carolina Ribeiro Bastos
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2015

Orientador
Nelma Resende Araújo Moreira, Professora Auxiliar,
Faculdade de Ciências da Universidade do Porto

Coorientador
Rogério Ventura Lages dos Santos Reis, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Para os meus pais

Agradecimentos

Inicialmente, os meus sinceros agradecimentos à professora Nelma Moreira e ao professor Rogério Reis, que tão dedicada e pacientemente me orientaram ao longo dos últimos anos. Agradeço também à professora Sabine Broda, pelas suas correções e contribuições.

Sou especialmente grata aos meus familiares e amigos, que carinhosamente me acompanharam nesta jornada. Sobretudo, à minha mãe e ao meu pai, que tudo tornaram possível, acompanhando, apoiando e dando todo o suporte. Aos meus irmãos, Paulinho e Helena. E ao Carlos pela sua quase infinita paciência, ajuda e carinho.

Gostaria também de agradecer aos projetos de investigação CANTE (PTDC/EIA-CCO/101904/2008) e PEst-C/MAT/UI0144/2013.

Abstract

The use of derivatives for efficiently deciding equivalence and membership problems in regular languages has been a major topic of recent research. Following these studies, in this work, we generalize the notions of partial derivative and support to extended regular expressions (regular expressions enriched with the intersection and complement operators) and prove that they lead to different automata constructions, unlike what happens with simple regular expressions.

For a regular expression with intersection, α, we show that $2^{|\alpha|_\Sigma - |\alpha|_\cap - 1}$ is a tight worst-case upper bound for the number of states of partial derivative's automaton, where $|\alpha|_\Sigma$ and $|\alpha|_\cap$ are the number of occurrences of alphabetic symbols and the number of occurrences of the intersection operator in α, respectively. We also conducted an experimental study that suggests that the average-state complexity of partial derivative's automaton may even be polynomial w.r.t. the size for these expressions.

Furthermore, adding the complement operator, we prove that the construction of the set of partial derivatives of an extended regular expression is ensured to be finite only if extended regular expressions are considered modulo commutativity, idempotence and identity of intersection.

We also present a special representation for extended regular expressions, which ensures the termination of the construction of Brzozowski's and partial derivative's automaton. These regular expressions and associated methods were implemented in the FAdo system.

Keywords: regular languages, finite automata, extended regular expressions, derivatives.

Resumo

O uso de derivadas para decidir os problemas da pertença e da equivalência em expressões regulares tem sido um importante tópico de investigação. No seguimento destes estudos, generalizamos as noções de derivada parcial e de suporte a expressões regulares estendidas (expressões regulares com interseção e complemento). Provamos também que estas noções conduzem a diferentes construções de autómatos, ao contrário do que se verifica para expressões regulares simples.

Para uma expressão regular com interseção, α, demonstramos que $2^{|\alpha|_\Sigma - |\alpha|_\cap - 1}$ é um majorante antigível do número de estados do autómato das derivadas parciais, para o qual $|\alpha|_\Sigma$ e $|\alpha|_\cap$ são o número de ocorrências de símbolos alfabéticos e o número de ocorrências do operador interseção, respetivamente. Conduzimos também um estudo experimental que sugere que, no caso médio, o número de estados do autómato das derivadas parciais pode ser polinomial relativamente ao tamanho destas expressões.

Para além disso, para expressões regulares estendidas, provamos que a construção do conjunto das derivadas parciais é finita se considerarmos estas expressões módulo comutatividade, idempotência e identidade do operador interseção.

Apresentamos também uma representação especial para expressões regulares estendidas que assegura a conclusão em tempo finito da construção do autómato de Brzozowski e do autómato das derivadas parciais. Estas expressões e os seus respetivos métodos foram implementados no sistema FAdo.

Palavras-chave: linguagens regulares, autómatos finitos, expressões regulares estendidas, derivadas.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Regular languages, in spite of their apparently limited expressive power, have applications in almost all areas of Computer Science. Regular expressions are the most readable and compact representation for regular languages and can be efficiently transformed into equivalent nondeterministic automata (NFA). In many applications, however, regular expressions with additional operators, such as intersection ($\cap$) and complement ($\neg$), are considered. Although these operators do not increase the expressive power of the associated language, they lead to gains in the succinctness of the representation. In fact, regular expressions with intersection and complement (extended regular expressions) are double exponentially more succinct [16].

To verify if two regular expressions are equivalent, or if a word belongs to the language described by a regular expression, an useful approach is to use finite automata. This is made by converting the regular expression into an equivalent finite automaton, which should be small to be efficiently manipulated. For simple regular expressions it is possible to build equivalent NFAs and deterministic finite automata (DFA) with linear and exponential number of states, respectively, with respect to the size of the regular expression. The conversion of an extended regular expression into an equivalent DFA and NFA, however, involves, in the worst case, an exponential and double exponential blow up [16], respectively.

The conversion of an extended regular expression into an equivalent DFA can be done by using Brzozowski's derivatives [10]. To ensure termination of this conversion, regular expressions must be considered modulo some algebraic properties such as

associativity, commutativity, and idempotence of disjunction (ACI_+). The Antimirov's construction [2] converts a simple regular expressions into an NFA, by using partial derivatives. This construction is equivalent to the resolution of Mirkin's systems of equations [24]. The average complexity of both conversions were recently studied using the framework of analytic combinatorics [7, 8].

Caron et al. [11] generalized Antimirov's partial derivatives to extended regular expressions. These differ from Antimirov's partial derivatives, since they are sets of sets of regular expressions instead of a set of regular expressions. These sets of sets can be seen as a kind of a disjunctive normal form for extended regular expressions.

One of the goals of this work was to continue this line of research. In particular, to implement extended regular expressions and several derivative based methods within the FAdo system and test their feasibility. Furthermore, we wanted to generalize Mirkin's method to extended regular expressions, as this method provides inductive definitions that are more adequate for the study of average case complexity using the analytic framework.

Towards these goals, we gradually added the operators to simple regular expressions. We began our study with regular expressions with intersection. Then we made an identical study for extended regular expressions. We concluded our work by introducing extended regular expression modulo some properties as ACI of disjunction and intersection.

The structure of this thesis is as follows. The next two chapters contain the essential theoretical background. Chapter 2 gives an introduction to some concepts of language theory, to finite automata and to the FAdo system. In Chapter 3, we introduce the notion of regular expression and its conversion methods to finite automata.

In Chapter 4, we extend simple regular expression with the intersection operator. We give several notions of derivatives and conversion methods to finite automata. Then, we compare Antimirov's construction and Mirkin's construction and also present a tight worst-case upper bound for the number of states of both. We also present some implementations in the FAdo system, as well as some experimental results.

In Chapter 5, extended regular expressions are introduced. Again we extend the notions of partial derivatives and of solution of a system of expression equations.

Comparative studies of Antimirov's construction and Mirkin's construction for these regular expressions are also presented.

Finally, in Chapter 6, we present a special representation for extended regular expressions, which ensures the termination of Brzozowski's and Antimirov's constructions. We also expose how we implemented them in the FAdo system.

Chapter 7 ends this thesis with some conclusions and lines of future research.

Chapter 2

Preliminaries

In the context of formal languages, an *alphabet*, denoted by Σ, is a finite nonempty set of symbols. A *word* over Σ is a finite sequence of symbols taken from Σ. The word consisting in zero symbols is the *empty word* and it is denoted by ε. For example, a and b are symbols of the alphabet $\Sigma = \{a, b\}$ and ε, a and $aaba$ are words over Σ. The length of a word w, denoted by $|w|$, is the number of symbols in w. The words ε, a and $aaba$, for example, have length 0, 1 and 4, respectively. The length of a word $w \in \Sigma^*$ is defined recursively in the following way:

$$\begin{aligned} &|\varepsilon| = 0, \\ &|wa| = |w| + 1, \text{ where } a \in \Sigma. \end{aligned}$$

The set of all words over Σ is denoted by Σ^*. For example, $\Sigma = \{a, b\}$, then

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}.$$

The concatenation of two words w and u over Σ, denoted $w \cdot u$ or wu, is the word obtained by juxtaposing the word u at the end of w. For example, the concatenation of aa and ba is the word $aaba$. The concatenation of $w, w' \in \Sigma^*$ is defined recursively by the following:

$$\begin{aligned} &w \cdot \varepsilon = w, \\ &w \cdot (w'a) = (w \cdot w')\, a, \text{ where } a \in \Sigma. \end{aligned}$$

Furthermore, the concatenation is associative and the empty word is the identity element, that is $w \cdot \varepsilon = \varepsilon \cdot w = w$. The set Σ^* is thus a monoid with the concatenation as the operator and the empty word as the identity.

Given a word $w = uvz$, where $u, v, z \in \Sigma^*$, we say that u is a *prefix* of w, v is a *infix* of w and z is a *suffix* of w. The words u, v and z are *subwords* of w.

The reversal of a word $w = a_1 \dots a_n$, denoted by w^r, is the word $a_n \dots a_1$. It is defined recursively in the following way:

$$\begin{aligned} &\varepsilon^r = \varepsilon, \\ &w^r = u^r a, \quad \text{where } w = au,\ a \in \Sigma \text{ and } u \in \Sigma^*. \end{aligned}$$

2.1 Languages

A *language* over an alphabet Σ is a subset of Σ^*. The empty set, $\emptyset$, denotes the *empty language*. The set Σ^* denotes the *universal language*. The *cardinality* of a language $\mathcal{L}$ is denoted by $|\mathcal{L}|$.

Languages are closed under the standard set operations, such as union, intersection and difference. The *complement* of a language $\mathcal{L}$ over Σ, denoted by $\overline{\mathcal{L}}$, is the set $\Sigma^* \backslash \mathcal{L}$. The *concatenation* of $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_1\mathcal{L}_2$, is the language

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

The ith power of a language $\mathcal{L}$, denoted $\mathcal{L}^i$, is defined recursively as $\mathcal{L}^0 = \{\varepsilon\}$ and $\mathcal{L}^i = \mathcal{L}\mathcal{L}^{i-1}$ for an integer $i \geq 1$. The *Kleene closure* of $\mathcal{L}$, denoted $\mathcal{L}^*$ is the set

$$\mathcal{L}^* = \bigcup_{i=0}^{\infty} \mathcal{L}^i$$

and the *positive closure*, denoted $\mathcal{L}^+$, represents the language $\mathcal{L}\mathcal{L}^*$ or equivalently

$$\mathcal{L}^+ = \bigcup_{i=1}^{\infty} \mathcal{L}^i.$$

The class of *regular languages* over an alphabet Σ is the smallest class containing the empty set $\emptyset$ and the singletons $\{a\}$, where $a \in \Sigma$, and closed under union,

concatenation and Kleene star. The class of regular languages is also closed under complement and intersection.

For a regular language $\mathcal{L}$, the function $\varepsilon(\mathcal{L})$ is defined by $\varepsilon(\mathcal{L}) = \varepsilon$, if $\varepsilon \in \mathcal{L}$, and $\varepsilon(\mathcal{L}) = \emptyset$, otherwise.

Given a language $\mathcal{L}$ over Σ, the *left-quotient* of $\mathcal{L}$ w.r.t. a symbol $a \in \Sigma$ is the language $a^{-1}\mathcal{L} = \{w \mid aw \in \mathcal{L}\}$. This can be defined recursively as follows:

$$
\begin{array}{ll}
a^{-1}\emptyset = \emptyset, & a^{-1}(\mathcal{L}_1 \cap \mathcal{L}_2) = a^{-1}\mathcal{L}_1 \cap a^{-1}\mathcal{L}_2, \\
a^{-1}\{\varepsilon\} = \emptyset, & a^{-1}(\mathcal{L}_1\mathcal{L}_2) = (a^{-1}\mathcal{L}_1)\mathcal{L}_2 \cup a^{-1}\mathcal{L}_2, \text{ if } \varepsilon \in \mathcal{L}_1, \\
a^{-1}\{a\} = \{\varepsilon\}, & a^{-1}(\mathcal{L}_1\mathcal{L}_2) = (a^{-1}\mathcal{L}_1)\mathcal{L}_2, \text{ if } \varepsilon \notin \mathcal{L}_1, \\
a^{-1}\{b\} = \emptyset, \ b \in \Sigma \text{ and } b \neq a, & a^{-1}(\mathcal{L}^*) = (a^{-1}\mathcal{L})\mathcal{L}^*, \\
a^{-1}(\mathcal{L}_1 \cup \mathcal{L}_2) = a^{-1}\mathcal{L}_1 \cup a^{-1}\mathcal{L}_2, & a^{-1}(\overline{\mathcal{L}}) = \Sigma^* \backslash (a^{-1}\mathcal{L}),
\end{array}
$$

where $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ are languages over Σ. The definition of left-quotient of a language $\mathcal{L}$ over Σ is extensible to a word $w \in \Sigma^*$ and it is the language $w^{-1}(\mathcal{L}) = \{w' \mid ww' \in \mathcal{L}\}$. Moreover, $w^{-1}\mathcal{L}$ can be defined inductively by:

$$
\begin{aligned}
&\varepsilon^{-1}\mathcal{L} = \mathcal{L}, \\
&(wa)^{-1}\mathcal{L} = a^{-1}(w^{-1}\mathcal{L}), \ a \in \Sigma,
\end{aligned}
$$

where $a \in \Sigma$. Observe that it means that a word w belongs to the language $\mathcal{L}$ if and only if $\varepsilon \in w^{-1}(\mathcal{L})$.

The left-quotient of a regular language is also a regular language. Furthermore, the set $\bigcup_{w \in \Sigma^*} w^{-1}\mathcal{L}$ of all left-quotients of a regular language $\mathcal{L}$ over Σ w.r.t. the words $w \in \Sigma^*$ is finite.

2.2 Finite Automata

In the context of formal languages, languages are defined by models. The models that define, and also accept, the regular languages are the finite automata. In this section we describe two types of finite automata: deterministic finite automata and nondeterministic finite automata.

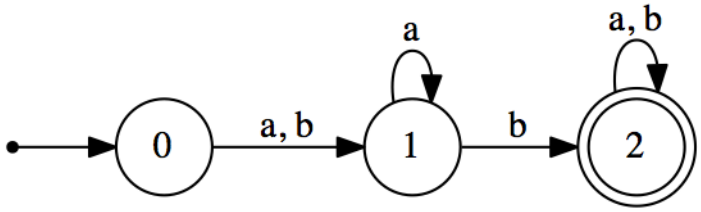


Figure 2.1: Diagram of the deterministic finite automaton $\mathcal{D}_1$.

2.2.1 Deterministic Finite Automata

Definition 2.1. ***(Deterministic Finite Automaton)*** *A* deterministic finite automaton *(DFA)* $\mathcal{D}$ *is a quintuple* $\langle Q, \Sigma, q_0, \delta, F \rangle$ *where:*

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

A *transition diagram* is a graphical representation for a DFA. The transition diagram is a directed graph. The vertices of the graph correspond to the states Q of the DFA. Each transition from state q to state p with input a, $\delta(q, a) = p$, corresponds to an arc from state p to state q labeled by the symbol a. The initial state q_0 is marked with an arrow. The final states are represented by a double circle.

Example 2.2. *Let* $\mathcal{D}_1 = \langle Q, \Sigma, q_0, \delta, F \rangle$ *be the deterministic finite automaton described by the transition diagram in Figure 2.1, where the set of states is* $Q = \{0, 1, 2\}$*, the alphabet is* $\Sigma = \{a, b\}$*, the initial state is* $q_0 = 0$*, the set of final states is* $F = \{2\}$ *and the transition function is defined as follows:*

$$\delta(0, a) = 1, \qquad \delta(1, b) = 2,$$

$$\delta(0, b) = 1, \qquad \delta(2, a) = 2,$$
$$\delta(1, a) = 1, \qquad \delta(2, b) = 2.$$

A DFA $\mathcal{D} = \langle Q, \Sigma, q_0, \delta, F \rangle$ is called *complete* if the transition function δ is total, i.e., if δ is defined for every state $q \in Q$ and symbol $a \in \Sigma$.

Let $\delta : Q \times \Sigma^* \to Q$ be a extension of the transition function δ to words. The extended δ is defined recursively by the following:

$$\delta(q, \varepsilon) = q,$$
$$\delta(q, wa) = \delta(\delta(q, w), a), \quad w \in \Sigma^* \text{ and } a \in \Sigma.$$

A word w is accepted by a DFA $\mathcal{D} = \langle Q, \Sigma, q_0, \delta, F \rangle$, if $\delta(q_0, w)$ is defined and the image of the last symbol of w is a final state. The language accepted by a DFA $\mathcal{D}$ is the set of words that are accepted by $\mathcal{D}$ and it is defined by:

$$\mathcal{L}(\mathcal{D}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}.$$

Two automata are said to be *equivalent* if they accept exactly the same language. A DFA is *minimal* if there is no equivalent automaton with fewer states. Furthermore, there is a unique minimal DFA for every language.

2.2.2 Nondeterministic Finite Automata

A *nondeterministic finite automaton* (NFA) is an extension of the deterministic finite automaton. This is obtained by basically allowing the image of the transition function of the DFA to be a set of states.

Definition 2.3. ***(Nondeterministic Finite Automaton)*** *A nondeterministic finite automaton (NFA) $\mathcal{N}$ is a quintuple $\langle Q, \Sigma, Q_0, \delta, F \rangle$, where Q, Σ and F represent exactly the same as for a DFA, i.e., the set of states, the alphabet and the set of final states, respectively, being different only in the initial state and in the transition function definition. The set $Q_0 \subseteq Q$ is the set of initial states. The transition function is defined as $\delta : Q \times \Sigma \to 2^Q$.*

Note that we often denote a singleton state, i.e., the set containing exactly one state, by the state itself when no confusion will be caused.

Example 2.4. *Let $\mathcal{N}_2 = \langle Q, \Sigma, q_0, \delta, F\rangle$ be a nondeterministic finite automaton, where the set of states is $Q = \{0, 1, 2\}$, the alphabet is $\Sigma = \{a, b\}$, the initial state is $q_0 = 0$, the set of final states is $F = \{2\}$ and the transition function is defined as follows:*

$$\begin{aligned} &\delta(0, a) = \{0, 1\}, && \delta(1, b) = \{0, 2\},\\ &\delta(0, b) = \{0\}, && \delta(2, a) = \{2\},\\ &\delta(1, a) = \{1\}, && \delta(2, b) = \{1\}. \end{aligned}$$

The transition diagram of $\mathcal{N}_2$ is shown in Figure 2.2,

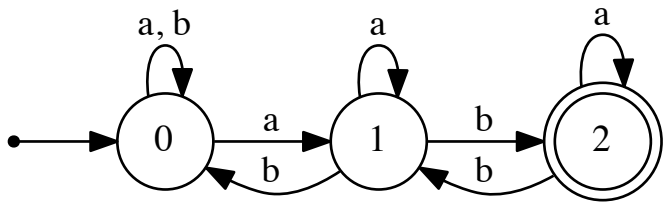


Figure 2.2: Diagram of the nondeterministic finite automaton $\mathcal{N}_2$.

The transition function δ, as in DFAs, can be extended to $\delta : Q \times \Sigma^* \to 2^Q$. This function is defined as follows:

$$\begin{aligned} &\delta(q, \varepsilon) = \{q\},\\ &\delta(q, wa) = \bigcup_{p \in \delta(q,w)} \delta(p, a), \quad w \in \Sigma^* \text{ and } a \in \Sigma. \end{aligned}$$

It may also be necessary to use the transition function to map several states, extending δ to $2^Q \times \Sigma^* \to 2^Q$ by

$$\delta(P, w) = \bigcup_{q \in P} \delta(q, w),$$

where $P \subseteq Q$.

An NFA $\mathcal{N} = \langle Q, \Sigma, Q_0, \delta, F\rangle$ accepts a word $w \in \Sigma^*$ if there is a state $q_f \in F$ such that $q_f \in \delta(q_0, w)$, for some $q_0 \in Q_0$. Therefore, the language accepted by an NFA $\mathcal{N}$ is

$$\mathcal{L}(\mathcal{N}) = \{w \in \Sigma^* \mid \delta(Q_0, w) \cap F \neq \emptyset\}.$$

Two finite automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if and only if they accept exactly the same language, i.e., $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. An NFA is said *minimal* if there is no equivalent NFA with fewer states. The number of minimal NFAs for a given language can be greater than one.

The *left language* of a state q of $\mathcal{N}$ is defined as $\mathcal{L}_L(\mathcal{N}, q) = \{w \in \Sigma^* \mid q \in \delta(Q_0, w)\}$ and the *right language* of q as $\mathcal{L}_R(\mathcal{N}, q) = \{w \in \Sigma^* \mid \delta(q, w) \cap F \neq \emptyset\}$). Two states of an NFA are said *equivalent states* if they have the same right language.

The automaton that accepts the reversal of an input sequence is called the *reversal automaton*. The automaton is next defined.

Definition 2.5. ***(Reversal Automaton)*** *For a nondeterministic finite automaton $\mathcal{A} = \langle Q, \Sigma, Q_0, \delta, F\rangle$, the reversal automaton of $\mathcal{A}$ is the finite automaton $\mathcal{A}^R = \langle Q, \Sigma, F, \delta^R, Q_0\rangle$, where*

$$\forall a \in \Sigma, \forall q_i, q_j \in Q \ (q_i \in \delta^R(q_j, a) \Leftrightarrow q_j \in \delta(q_i, a)).$$

We say that an automaton is *trimmed* if there are no states not reachable from the initial state. So, we obtain a trimmed automaton by pruning these unreachable states.

An ε-NFA is an extension of NFA that allows transitions with the empty input ε. We define the ε-NFA as the quintuple $\langle Q, \Sigma, q_0, \delta, F\rangle$, having Q, Σ, Q_0 and F the same meaning as for an NFA and being δ the transition function that maps $Q \times \{\varepsilon\} \cup \Sigma$ to 2^Q. The ε-NFAs have the same descriptive power of NFAs, i.e., for every ε-NFA there is an equivalent NFA.

2.2.3 The equivalence of DFAs and NFAs

Since deterministic finite automata are a special case of nondeterministic finite automata, it is clear that every class of languages accepted by DFAs are also accepted by NFAs. Moreover, every NFA can be transformed into an equivalent DFA. So the languages accepted by NFAs are also accepted by DFAs and thus NFAs and DFAs accept exactly the same class of languages.

The class of languages accepted by NFAs and DFAs is exactly the class of regular languages. The proof that the class of languages accepted by finite automata contains

the finite languages and that is closed under union, concatenation and Kleene star can be found in several books, such as [22] and [18].

The transformation of an NFA into a DFA is achieved by the *subset construction*, which is described bellow.

Definition 2.6. ***(Subset Construction)*** *Given an NFA $\mathcal{N} = \langle Q, \Sigma, Q_0, \delta, F\rangle$, a DFA equivalent to $\mathcal{N}$ is defined as $Det(\mathcal{N}) = \langle Q^d, \Sigma, q_0, \delta^d, F^d\rangle$, where:*

- $Q^d = \{\delta(Q_0, w) \mid w \in \Sigma^*\} = \{P_1, \ldots, P_n\} \subseteq 2^Q$;
- $\delta^d(P_i, a) = \delta(P_i, a)$;
- $q_0 = Q_0$;
- $F^d = \{P_i \in Q^d \mid P_i \cap F \neq \emptyset\}$.

Note that all states of $\mathcal{N}$ with the same left language were merged into one state in $Det(\mathcal{N})$.

In the worst case, the obtained $Det(\mathcal{N})$ can have $2^{|Q|}$ states, being exponential larger than $\mathcal{N}$. The determinisation of an NFA is an EXPTIME-hard problem.

The *co-determinisation* of an NFA $\mathcal{N}$ corresponds to construct the reversal automaton of $\mathcal{N}$, determinise it and to construct the reversal automaton of the latter automaton, i.e., $Codet(\mathcal{N}) = Det(\mathcal{N}^R)^R$. By co-determinising $\mathcal{N}$, we have all states with the same right language merged into one state. Thereby, $Codet(\mathcal{N})$ has only a final state.

We obtain the *minimal DFA* equivalent to an NFA $\mathcal{N}$, $Min(\mathcal{N})$, if all two states with the same left language were merged too, i.e., if there are no states with the same right and left languages. Therefore, in such a way to obtain the minimal DFA equivalent to $\mathcal{N}$ it is enough to determinise $Codet(\mathcal{N})$, i.e., $Min(\mathcal{N}) = Det(Codet(\mathcal{N}))$.

However, for a nondeterministic finite automaton, merging states with the same right and left language is not enough to obtain the minimal NFA.

2.2.4 Systems of Equations

Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F\rangle$ be a finite automaton with the states $Q = \{0, 1, \ldots, n\}$, the initial state $q_0 = 0$ and the alphabet $\Sigma = \{a_1, \ldots, a_k\}$. Denote by $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_n$ the

right languages of each state $i \in Q$ and define the language $\mathcal{L}_{li}$, for each $l = 1, \ldots, k$ and $i = 1, \ldots, n$, as

$$\mathcal{L}_{li} = \bigcup_{j \in \delta(i,a_l)} \mathcal{L}_j,$$

if $\mathcal{A}$ is an NFA, or $\mathcal{L}_{li} = \mathcal{L}_{\delta(i,a_l)}$, if $\mathcal{A}$ is a DFA. Therefore, the following *system of equations* holds:

$$\begin{aligned}
\mathcal{L}_0 &= a_1\mathcal{L}_{10} \cup \ldots \cup a_k\mathcal{L}_{k0} \cup \varepsilon(\mathcal{L}_0)\\
\mathcal{L}_1 &= a_1\mathcal{L}_{11} \cup \ldots \cup a_k\mathcal{L}_{k1} \cup \varepsilon(\mathcal{L}_1)\\
&\vdots\\
\mathcal{L}_n &= a_1\mathcal{L}_{1n} \cup \ldots \cup a_k\mathcal{L}_{kn} \cup \varepsilon(\mathcal{L}_n).
\end{aligned}$$

This system of equations is denoted by $S_{\mathcal{L}_0}$.

2.3 FAdo System

FAdo [14, 1] is an open source system written in Python that provides tools for formal languages manipulation. At present, most of the standard operations for the manipulation of regular languages are implemented in this system.

One of the modules provided by FAdo is the module `fa`. The module `fa` defines the classes representing finite automata. The class `FA` is the main class of this module and it defines the basic structure of determininstic and nondeterministic finite automata. The attributes of `FA` are:

- *Sigma*: a set of alphabet symbols;
- *States*: a list of states (each state is referred as its index in the list);
- *Initial*: an index of the initial state or a set of indexes of the initial states (NFA);
- *Final*: a set of indexes of the final states;
- *delta*: a dictionary containing the transition function.

For each subclass of `FA` there are special methods to add, to delete and to modify alphabet symbols, states and transitions.

The class `DFA` provides methods to manipulate deterministic finite automata and it inherits from `FA`. The following code demonstrates how to proceed to build the `DFA` of Example 2.2:

```
from FAdo.fa import *
dfa = DFA()
dfa.setSigma({'a', 'b'})
dfa.addState(0)
dfa.addState(1)
dfa.addState(2)
dfa.setInitial(0)
dfa.addFinal(2)
dfa.addTransition(0, 'a', 1)
dfa.addTransition(0, 'b', 1)
dfa.addTransition(1, 'a', 1)
dfa.addTransition(1, 'b', 2)
dfa.addTransition(2, 'a', 2)
dfa.addTransition(2, 'b', 2)
```

To verify if a word w is recognized by `dfa`, we use the function `evalWordP("w")`:

```
>>> dfa.evalWordP("aba")
True
>>> dfa.evalWordP("abb")
False
```

We can use the operator == to test if two DFAs are equivalent (`dfa == other_dfa`). It is also possible to minimize and to create the reversal automaton of `dfa`:

```
>>> minimal_dfa = dfa.minimal()
>>> reversal_dfa = dfa.reversal()
```

The class `NFA` provides methods to manipulate nondeterministic finite automata and, in the same way as the class `DFA`, it inherites from `FA`. The above methods are also available for NFAs. In FAdo, every NFA can have ε-transitions.

The full documentation of the available methods and classes can be found in the FAdo webpage [14].

Chapter 3

Regular Expressions

Just as finite automata, *regular expressions* define the regular languages. In contrast with the finite automata, regular expressions provide a more succinct and also an human-readable notation.

Definition 3.1. ***(Regular Expression)*** *Let Σ be an alphabet. A regular expression over Σ with intersection is given by the following grammar:*

$$\alpha := \emptyset \mid \varepsilon \mid a \in \Sigma \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^*.$$

The language $\mathcal{L}(\alpha)$ associated with α is defined inductively as follows:

$$\begin{aligned} &\mathcal{L}(\emptyset) = \emptyset, && \mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta), \\ &\mathcal{L}(\varepsilon) = \{\varepsilon\}, && \mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta), \\ &\mathcal{L}(a) = \{a\}, && \mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*. \end{aligned}$$

The set of all regular expressions is denoted by RE.

The regular expressions in RE will also be called *simple regular expressions*. We assume that $*$ has higher precedence than $\cdot$ and $+$, and that $\cdot$ has higher precedence than $+$. Assuming this precedence, parentheses can be omitted in regular expressions. We also usually omit the operator $\cdot$.

Given a set $S \subseteq \mathsf{RE}$, the language of S is defined as being $\mathcal{L}(S) = \bigcup_{\alpha \in S} \mathcal{L}(\alpha)$. Observe that a set of regular expressions denotes the same language that the disjunction of

the regular expressions within that set. That is, given $\alpha_1, \ldots, \alpha_n \in \mathsf{RE}$, $\mathcal{L}(\{\alpha_1, \ldots, \alpha_n\}) = \bigcup_{\alpha_i \in S} \mathcal{L}(\alpha_i) = \mathcal{L}(\alpha_1 + \cdots + \alpha_n)$.

Two regular expressions α and β over Σ are *equivalent*, we denote by $\alpha = \beta$, if they represent the same language, i.e., $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$. If α and β are syntactically identical, we write $\alpha \equiv \beta$.

We define the *size* of α, written as $|\alpha|$, as the number of symbols in α, not counting parentheses. The *alphabetic size* of α, written $|\alpha|_\Sigma$, is the number of alphabet symbols in α. For example, $\alpha = (aa + b^*)a$ has size $|\alpha| = 8$ and alphabetic size $|\alpha|_\Sigma = 4$.

A regular expression α possesses the *empty word property* (e.w.p.), if the language of α contains the empty word. The e.w.p. can be coded by the function $\varepsilon : \mathsf{RE} \to \{\varepsilon, \emptyset\}$, where $\varepsilon(\alpha) = \varepsilon$ if $\varepsilon \in \mathcal{L}(\alpha)$ and $\varepsilon(\alpha) = \emptyset$ otherwise. For example, $\varepsilon(a^* + b) = \varepsilon$ and $\varepsilon(a + b) = \emptyset$. Being α and β regular expressions over Σ, a recursive definition for ε is given by the following:

$$
\begin{array}{ll}
\varepsilon(\emptyset) = \emptyset, & \varepsilon(\alpha + \beta) = \varepsilon, \text{ if } \varepsilon(\alpha) = \varepsilon \text{ or } \varepsilon(\beta) = \varepsilon, \\
\varepsilon(\varepsilon) = \varepsilon, & \varepsilon(\alpha\beta) = \emptyset, \text{ if } \varepsilon(\alpha) = \emptyset \text{ or } \varepsilon(\beta) = \emptyset, \\
\varepsilon(a) = \emptyset, \ a \in \Sigma, & \varepsilon(\alpha\beta) = \varepsilon, \text{ if } \varepsilon(\alpha) = \varepsilon(\beta) = \varepsilon, \\
\varepsilon(\alpha + \beta) = \emptyset, \text{ if } \varepsilon(\alpha) = \varepsilon(\beta) = \emptyset, & \varepsilon(\alpha^*) = \varepsilon.
\end{array}
$$

3.1 Kleene Algebra

A *Kleene algebra* is an algebraic structure $(A, +, \cdot, *, 0, 1)$, where $(A, +, \cdot, 0, 1)$ is an idempotent semiring and $*$ is a unary operator satisfying a set of axioms, that are specified below. The set of regular expressions RE over Σ forms a Kleene Algebra, $(\mathsf{RE}, +, \cdot, *, \emptyset, \varepsilon)$.

There are several axiomatizations for the Kleene algebra [25, 19, 13]. However, only the axiomatic system suggested by Salomaa [25] was extended to other operations such as intersection and complement. That is why this will be the chosen axiomatic system.

Salomaa [25] presented the system F_1 as an axiomatization for the Kleene algebra, and showed its completeness and soundness. There are 11 axioms in the system F_1,

these axioms are (A_1) - (A_{11}):

$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma, \tag{A_1}$$
$$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma), \tag{A_2}$$
$$\alpha + \beta = \beta + \alpha, \tag{A_3}$$
$$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma, \tag{A_4}$$
$$(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma, \tag{A_5}$$
$$\alpha + \alpha = \alpha, \tag{A_6}$$
$$\alpha \cdot \varepsilon = \alpha, \tag{A_7}$$
$$\alpha.\emptyset = \emptyset, \tag{A_8}$$
$$\alpha + \emptyset = \alpha, \tag{A_9}$$
$$\alpha^* = \varepsilon + \alpha \cdot \alpha^*, \tag{A_{10}}$$
$$\alpha^* = (\varepsilon + \alpha)^*, \tag{A_{11}}$$

where $\alpha, \beta, \gamma \in \mathsf{RE}$. There are also two inference rules in the system F_1, which are:

1. Rule of *substitution*: If $\alpha = \beta$ and γ' is the result of replacing an occurrence of α by β in γ, then we may infer $\gamma' = \gamma$.
2. Rule of *solution of equations*: If $\alpha = \alpha\beta + \gamma$ and $\varepsilon(\beta) = \emptyset$, then we may infer $\alpha = \gamma\beta^*$.

In the context of the Kleene algebras, the axioms of $*$ are (A_{10}) and (A_{11}). The idempotent semiring $(\mathsf{RE}, +, \cdot, \emptyset, \varepsilon)$ satisfies the axioms (A_1) to (A_9).

We say that two regular expressions are *E-similar* if one can be transformed to the other by using the set of axioms E. Two regular expressions are *E-dissimilar* if and only if they are not E-similar.

We denote by ACI_+ the set of axioms that corresponds to the associativity, commutativity and idempotence of the operator $+$. The set ACI_+ is constituted by the axioms A_1, A_3 and A_6.

From (A_6) and (A_9), it follows that $\varepsilon + \varepsilon = \varepsilon$ and $\varepsilon + \emptyset = \varepsilon$. On the other hand, from (A_7) and (A_8), we have that $\varepsilon \cdot \varepsilon = \varepsilon$ and $\varepsilon \cdot \emptyset = \emptyset$. Thus, it is possible to redefine $\varepsilon : \mathsf{RE} \to \{\emptyset, \varepsilon\}$ as follows:

$$\varepsilon(\emptyset) = \emptyset, \qquad \varepsilon(\alpha + \beta) = \varepsilon(\alpha) + \varepsilon(\beta),$$

$$\varepsilon(\varepsilon) = \varepsilon, \qquad \varepsilon(\alpha\beta) = \varepsilon(\alpha)\varepsilon(\beta),$$
$$\varepsilon(a) = \emptyset, \ \ a \in \Sigma, \qquad \varepsilon(\alpha^*) = \varepsilon.$$

3.2 Derivatives

Brzozowski [10] presented the definition of the derivative of a regular expression, which extends the notion of a left-quotient of a language to regular expressions. In the same article, it is suggested an algorithm for converting a regular expression into a deterministic finite automaton, where the derivatives of the regular expressions represent the states of this automaton. The construction of this automaton will be described in the section 3.6.

Definition 3.2. ***(Derivative)*** *Let $\alpha \in RE$ be a regular expression over Σ. The derivative of α w.r.t. a symbol $a \in \Sigma$, written $d_a(\alpha)$, is recursively defined as follows:*

$$d_a(\emptyset) = \emptyset, \qquad d_a(\alpha + \beta) = d_a(\alpha) + d_a(\beta),$$
$$d_a(\varepsilon) = \emptyset, \qquad d_a(\alpha\beta) = d_a(\alpha)\beta + d_a(\beta), \ \ \textit{if } \varepsilon(\alpha) = \varepsilon,$$
$$d_a(a) = \varepsilon, \qquad d_a(\alpha\beta) = d_a(\alpha)\beta, \ \ \textit{if } \varepsilon(\alpha) = \emptyset,$$
$$d_a(b) = \emptyset, \ \ b \in \Sigma \textit{ and } b \neq a, \qquad d_a(\alpha^*) = d_a(\alpha)\alpha^*.$$

Remark 3.3. *In the original recursively definition of derivative [10], the derivative was defined for all boolean operators:*

$$d_a(f(\alpha, \beta)) = f(d_a(\alpha), d_a(\beta)),$$

where f is a boolean function. Since, simple regular expressions only have the disjunction operator, only this was presented in Definition 3.2.

Example 3.4. *Consider the regular expression $\alpha \equiv (ab + b)^*ab$ over $\{a, b\}$. The derivative of α w.r.t. a is*

$$\begin{aligned} d_a((ab + b)^*ab) &= d_a((ab + b)^*)ab + d_a(ab) \\ &= d_a(ab + b)(ab + b)^*ab + d_a(a)b \\ &= (d_a(ab) + d_a(b))(ab + b)^*ab + d_a(a)b \end{aligned}$$

$$= (d_a(a)b + d_a(b))(ab + b)^*ab + d_a(a)b$$
$$= (b + \emptyset)(ab + b)^*ab + b.$$

The definition of derivative of a regular expression α can be extended to a word $w \in \Sigma^*$ in the following way:

$$d_\varepsilon(\alpha) = \alpha,$$
$$d_{wa}(\alpha) = d_a(d_w(\alpha)), \;\; a \in \Sigma.$$

The left-quotient of the language of α w.r.t. w is the language of $d_w(\alpha)$, i.e., $w^{-1}\mathcal{L}(\alpha) = \mathcal{L}(d_w(\alpha))$ [10, Theorem 3.1]. A word w belongs to $\mathcal{L}(\alpha)$ if and only if $\varepsilon(d_w(\alpha)) = \varepsilon$.

Furthermore, for every regular expression α over Σ the following equivalence holds [10, Theorem 4.4]:

$$\alpha = \sum_{a \in \Sigma} a d_a(\alpha) + \varepsilon(\alpha), \tag{3.1}$$

where the terms in the sum are disjoint.

Brzozowski also proved that every regular expressions has only a finite number of ACI_+-dissimilar derivatives [10, Theorem 5.2].

Remark 3.5. *In the original definition of derivative [10], we have the following equation for the concatenation of two regular expressions α, β:*

$$d_a(\alpha\beta) = d_a(\alpha)\beta + \varepsilon(\alpha)d_a(\beta),$$

instead of having [Definition 3.2]

$$d_a(\alpha\beta) = \begin{cases} d_a(\alpha)\beta + d_a(\beta), & \text{if } \varepsilon(\alpha) = \varepsilon \\ d_a(\alpha)\beta, & \text{otherwise.} \end{cases}$$

This modification was proposed by Antimirov [2], since, without this modification, the set of all ACI_+-dissimilar derivatives is not finite. Take as example the regular expression a^*,

$$d_a(a^*) = \varepsilon a^*, \qquad d_{aa}(a^*) = \varepsilon\varepsilon a^*, \qquad d_{aaa}(a^*) = \varepsilon\varepsilon\varepsilon a^*, \qquad \ldots$$

It is clear to note that the set of all derivatives is infinite, even being ACI_+-dissimilars. With this modification, the set of all ACI_+-dissimilar derivatives is finite.

The set $D_{ACI_+}(\alpha)$ over Σ is the set of all ACI_+-dissimilar derivatives of a regular expression α. Namely, the set of representatives of the equivalence class modulo ACI_+ of the regular expressions $d_w(\alpha)$, for all $w \in \Sigma^*$.

3.3 Partial Derivatives

Antimirov [2] proposed the notion of *partial derivative*, which is a nondeterministic version of the Brzozowski derivative. Instead of a deterministic finite automaton, the partial derivative leads to a construction of a nondeterministic finite automaton.

Let the operation $\cdot : 2^{\mathsf{RE}} \times \mathsf{RE} \to 2^{\mathsf{RE}}$ be an extension of concatenation for sets of regular expressions. The operation is recursively defined as follows:

$$
\begin{aligned}
&S \cdot \emptyset = \emptyset,\\
&S \cdot \varepsilon = S,\\
&\emptyset \cdot \beta = \emptyset,\\
&\{\varepsilon\} \cdot \beta = \{\beta\},\\
&\{\alpha\} \cdot \beta = \{\alpha\beta\},\\
&(S \cup S') \cdot \beta = (S \cdot \beta) \cup (S' \cdot \beta),
\end{aligned}
$$

where $S, S' \subseteq 2^{\mathsf{RE}}$, $\alpha \in \mathsf{RE} \setminus \{\emptyset\}$ and $\beta \in \mathsf{RE} \setminus \{\emptyset, \varepsilon\}$.

Definition 3.6. ***(Partial Derivatives)*** *Let $\alpha \in \mathsf{RE}$ be a regular expression over Σ. The set of partial derivatives of α w.r.t. a symbol $a \in \Sigma$, written $\partial_a(\alpha)$, is recursively defined as follows:*

$$
\begin{aligned}
&\partial_a(\emptyset) = \emptyset, && \partial_a(\alpha + \beta) = \partial_a(\alpha) \cup \partial_a(\beta),\\
&\partial_a(\varepsilon) = \emptyset, && \partial_a(\alpha\beta) = \partial_a(\alpha)\beta \cup \partial_a(\beta), \ \textit{if } \varepsilon(\alpha) = \varepsilon,\\
&\partial_a(a) = \{\varepsilon\}, && \partial_a(\alpha\beta) = \partial_a(\alpha)\beta, \ \textit{if } \varepsilon(\alpha) = \emptyset,\\
&\partial_a(b) = \emptyset, \ b \in \Sigma \textit{ and } b \neq a, && \partial_a(\alpha^*) = \partial_a(\alpha)\alpha^*.
\end{aligned}
$$

Example 3.7. *For the regular expression $\alpha \equiv (ab + b)^*ab$ over $\Sigma = \{a, b\}$, the set of partial derivatives of α w.r.t. $a \in \Sigma$ is*

$$\partial_a((ab + b)^*ab) = \partial_a((ab + b)^*)ab \cup \partial_a(ab)$$

$$
\begin{aligned}
&= \partial_a((ab+b))(ab+b)^*ab \cup \partial_a(a)b \\
&= (\partial_a(ab) \cup \partial_a(b))(ab+b)^*ab \cup \partial_a(a)b \\
&= (\partial_a(a)b \cup \partial_a(b))(ab+b)^*ab \cup \partial_a(a)b \\
&= (\{\varepsilon\}b \cup \emptyset)(ab+b)^*ab \cup \{\varepsilon\}b \\
&= \{b\}(ab+b)^*ab \cup \{b\} \\
&= \{b(ab+b)^*ab\} \cup \{b\} \\
&= \{b(ab+b)^*ab, b\}.
\end{aligned}
$$

The definition of the set partial derivatives is extended to a word $w \in \Sigma^*$, to a set of words $W \subseteq \Sigma^*$ and to a set of regular expressions $R \subseteq \mathsf{RE}$ in the following way:

$$\partial_\varepsilon(\alpha) = \{\alpha\}, \tag{3.2}$$

$$\partial_{wa}(\alpha) = \partial_a(\partial_w(\alpha)), \;\; a \in \Sigma, \tag{3.3}$$

$$\partial_w(R) = \bigcup_{\alpha_i \in R} \partial_w(\alpha_i), \tag{3.4}$$

$$\partial_W(\alpha) = \bigcup_{w \in W} \partial_w(\alpha). \tag{3.5}$$

For any regular expression α and word $w \in \Sigma$, the left-quotient of the language $\mathcal{L}(\alpha)$ w.r.t. w is equivalent to the language of $\partial_w(\alpha)$, i.e. $w^{-1}\mathcal{L}(\alpha) = \mathcal{L}(\partial_w(\alpha))$, such as for Brzozowski derivatives.

Proposition 3.8. *Given a regular expression $\alpha \in \mathsf{RE}$ over Σ and a word $w \in \Sigma$, $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$.*

Proof. Let $\alpha \in \mathsf{RE}$ and $w \in \Sigma^*$. The proof proceed by induction on the length of w. For $w = \varepsilon$, we find that $\mathcal{L}(\partial_\varepsilon(\alpha)) = \mathcal{L}(\{\alpha\}) = \varepsilon^{-1}\mathcal{L}(\alpha)$. Let us assume that $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$ holds for some $w \in \Sigma^+$, we will prove the claim for $w' = wa$, where $a \in \Sigma$,

$$
\begin{aligned}
\mathcal{L}(\partial_{wa}(\alpha)) &= \mathcal{L}(\partial_a(\partial_w(\alpha))) = a^{-1}\mathcal{L}(\partial_w(\alpha)) \\
&= a^{-1}w^{-1}\mathcal{L}(\alpha) \;=\; (wa)^{-1}\mathcal{L}(\alpha).
\end{aligned}
$$

Observe that this follows from Lemma 4.7 and from the fact that $\mathcal{L}(\partial_a(S)) = a^{-1}\mathcal{L}(\alpha)$, which has a straightforward proof. □

Let $Suf(w)$ be the set of all non-empty suffixes of w, being defined as $Suf(w) = \{v \in \Sigma^+ \mid \exists u \in \Sigma^* : uv = w\}$. The following properties of the function ∂_w for every regular expressions $\alpha, \beta \in \mathsf{RE}$ and word $w \in \Sigma^+$ were proved by Antimirov [2, Lemma 3.3]:

$$\partial_w(\alpha + \beta) = \partial_w(\alpha) \cup \partial_w(\beta), \tag{3.6}$$

$$\partial_w(\alpha\beta) \subseteq \partial_w(\alpha)\beta \cup \bigcup_{v \in Suf(w)} \partial_v(\beta), \tag{3.7}$$

$$\partial_w(\alpha^*) \subseteq \bigcup_{v \in Suf(w)} \partial_v(\alpha)\alpha^*. \tag{3.8}$$

We denote by $\mathcal{PD}(\alpha)$ the set of all (syntactical different) partial derivatives of α, i.e., $\mathcal{PD}(\alpha) = \bigcup_{w \in \Sigma^*} \partial_w(\alpha)$. By $\partial^+(\alpha)$, we denote the set of all partial derivatives excluding the derivative by the empty word, $\partial^+(\alpha) = \bigcup_{w \in \Sigma^+} \partial_w(\alpha)$.

Example 3.9. *For the regular expression $(ab + b)^*ab$ over $\Sigma = \{a, b\}$, the set of all partial derivatives of α is:*

$$\mathcal{PD}((ab + b)^*ab) = \partial^+((ab + b)^*ab) = \{b(ab + b)^*ab, (ab + b)^*ab, b, \varepsilon\}.$$

From the equations (3.6), (3.7) and (3.8), Antimirov proved that the following inequalities hold [2, Theorem 3.4]:

$$|\partial^+(\alpha)| \leq |\alpha|_\Sigma, \tag{3.9}$$

$$|\mathcal{PD}(\alpha)| \leq |\alpha|_\Sigma + 1. \tag{3.10}$$

In this way, the set of all (syntactical different) partial derivatives of a regular expression is finite.

3.4 Linear Form

All partial derivatives w.r.t. every alphabet symbol of a regular expression can be computed on a single pass over the regular expression. With this aim, Antimirov [2] defined the function *linear form*, which will be treated in this section.

For a regular expression α and an alphabetic symbol a, the pair (a, α) is called a *monomial* with *head* a and a *tail* α. We denote by $Mon = \Sigma \times \mathsf{RE}$, the set of all

monomials for a given alphabet Σ. The language associated with a monomial and the language associated with a set of monomials are defined, respectively, as follows:

$$\mathcal{L}((a,\alpha)) = \mathcal{L}(a \cdot \alpha) = \{a\} \cdot \alpha, \tag{3.11}$$

$$\mathcal{L}(S) = \bigcup_{(a,\alpha)\in S} \mathcal{L}((a,\alpha)) = \bigcup_{(a,\alpha)\in S} \{a\} \cdot \mathcal{L}(\alpha), \tag{3.12}$$

where $\alpha \in \mathsf{RE}$ over Σ, $a \in \Sigma$ and $S \subseteq 2^{Mon}$.

Let the binary operator $\odot : 2^{Mon} \times \mathsf{RE} \to 2^{\mathsf{RE}}$ be the extension of concatenation for sets of monomials:

$$\begin{aligned}
&S \odot \emptyset = \emptyset,\\
&S \odot \varepsilon = S,\\
&\{(a,\emptyset)\} \odot \beta = \{(a,\emptyset)\},\\
&\{(a,\varepsilon)\} \odot \beta = \{(a,\beta)\},\\
&\{(a,\alpha)\} \odot \beta = \{(a,\alpha\beta)\},\\
&(S \cup S') \odot \beta = (S \odot \beta) \cup (S' \odot \beta),
\end{aligned}$$

where $a \in \Sigma$, $\alpha \in \mathsf{RE} \setminus \{\emptyset\}$, $\beta \in \mathsf{RE} \setminus \{\emptyset, \varepsilon\}$ and $S, S' \subseteq 2^{\mathsf{RE}}$.

Definition 3.10. *(**Linear Form**) The function $lf : RE \to 2^{Mon}$ returns the linear form of a regular expression and it is recursively defined as follows:*

$$\begin{aligned}
&lf(\emptyset) = \emptyset, && lf(\alpha\beta) = lf(\alpha) \odot \beta \cup lf(\beta), \ \ \textit{if } \varepsilon(\alpha) = \varepsilon,\\
&lf(\varepsilon) = \emptyset, && lf(\alpha\beta) = lf(\alpha) \odot \beta, \ \ \textit{if } \varepsilon(\alpha) = \emptyset,\\
&lf(a) = \{(a,\varepsilon)\}, && lf(\alpha^*) = lf(\alpha) \odot \alpha^*,\\
&lf(\alpha + \beta) = lf(\alpha) \cup lf(\beta). &&
\end{aligned}$$

where $\alpha, \beta \in RE$ over Σ and $a \in \Sigma$.

Example 3.11. *The linear form of the regular expression $\alpha \equiv (ab+b)^*ab$ is*

$$\begin{aligned}
lf((ab+b)^*ab) &= lf((ab+b)^*) \odot ab \cup lf(ab)\\
&= lf((ab+b)) \odot (ab+b)^*ab \cup lf(a) \odot b\\
&= (lf(ab) \cup lf(b)) \odot (ab+b)^*ab \cup lf(a) \odot b\\
&= (lf(a) \odot b \cup lf(b)) \odot (ab+b)^*ab \cup lf(a) \odot b
\end{aligned}$$

$$
\begin{aligned}
&= (\{(a,\varepsilon)\} \odot b \cup \{(b,\varepsilon)\}) \odot (ab+b)^*ab \cup \{(a,\varepsilon)\} \odot b \\
&= (\{(a,b),(b,\varepsilon)\}) \odot (ab+b)^*ab \cup \{(a,b)\} \\
&= \{(a, b(ab+b)^*ab), (b,(ab+b)^*ab)\} \cup \{(a,b)\} \\
&= \{(a, b(ab+b)^*ab), (b,(ab+b)^*ab), (a,b)\}.
\end{aligned}
$$

Moreover, for every regular expression α over Σ the following equivalence holds [2, Proposition 2.5]:

$$\alpha = \sum_{(a,\alpha') \in lf(\alpha)} a \cdot \alpha' + \varepsilon(\alpha).$$

The set of partial derivatives of α w.r.t. $a \in \Sigma$ can also be defined as [2, Definition 2.8]:

$$\partial_a(\alpha) = \{\alpha' \in \mathsf{RE} \setminus \{\emptyset\} \mid (a,\alpha') \in lf(\alpha)\}. \tag{3.13}$$

3.5 Systems of Equations

We can have a *system of equations* of regular expressions, in the same way that we did a system of equations of languages. Let $\mathcal{L}_0$ be the language denoted by the regular expression α_0. The system of equations is obtained from $S_{\mathcal{L}_0}$ by replacing each language $\mathcal{L}_i$ and $\mathcal{L}_{li}$ by the regular expressions α_i and α_{li}, respectively, where $\mathcal{L}_i = \mathcal{L}(\alpha_i)$ and $\mathcal{L}_{li} = \mathcal{L}(\alpha_{li})$.

Given $\alpha_0, \ldots, \alpha_n \in \mathsf{RE}$ over $\Sigma = \{a_1, \ldots, a_k\}$, a *system of equations* is:

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i) \quad \text{for all } i \in \{0, \ldots, n\},$$

where α_{li}, with $l = 1, \ldots, k$, is a, possibly empty, sum of elements of $\{\alpha_0, \ldots, \alpha_n\}$, i.e., $\alpha_{li} = \sum_{j \in I_{li}} \alpha_j$, where $I_{li} \subseteq \{\alpha_0, \ldots, \alpha_n\}$. This system is denoted by S_{α_0}.

Example 3.12. *Given the regular expressions $\alpha_0 \equiv a^*b + ab^*$, $\alpha_1 \equiv a^*b$, $\alpha_2 \equiv b^*$ and $\alpha_3 \equiv \varepsilon$ over $\{a,b\}$, a corresponding system of equations is*

$$
\begin{aligned}
a^*b + ab^* &= a \cdot (a^*b + b^*) + b \cdot \varepsilon + \emptyset \\
a^*b &= a \cdot a^*b + b \cdot \varepsilon + \emptyset \\
b^* &= a \cdot \emptyset + b \cdot b^* + \varepsilon \\
\varepsilon &= a \cdot \emptyset + b \cdot \emptyset + \varepsilon.
\end{aligned}
$$

Given a regular expression α_0, the main problem is to find a set of regular expressions $\alpha_1, \ldots, \alpha_n$, for which the system S_{α_0} holds. As previously seen in the section 3.2, for the deterministic case, the set of ACI_+-dissimilar derivatives $D_{ACI_+}(\alpha_0)$ is a solution for this problem. The Mirkin's prebases [24] and the partial derivatives are solutions for the nondeterministic case, as proved by Champarnaud and Ziadi [12]. Moreover, they proved that the Mirkin's prebases and the partial derivatives lead to an identical solution for the system of equations and thus to an identical nondeterministic finite automaton.

The construction of Mirkin's automaton is based on the notion of support and prebase of a regular expression, which are defined bellow.

Definition 3.13. *(**Support**) Consider $\alpha_0 \in \mathsf{RE}$ a regular expression over $\Sigma = \{a_1, \ldots, a_k\}$. A support of α_0 is a set $\{\alpha_1, \ldots, \alpha_n\}$, which for each $\alpha_i \in \{\alpha_0\} \cup \{\alpha_1, \ldots, \alpha_n\}$, the following equation holds:*

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i), \tag{3.14}$$

where each α_{li}, with $l = 1, \ldots, k$, is a, possibly empty, sum of elements of $\{\alpha_1, \ldots, \alpha_n\}$.

If the set π is a support of the regular expression α, then the set $\pi \cup \{\alpha\}$ is a *prebase* of α.

The following proposition gives a recursive definition of a support of a regular expression. This proposition was proved by Mirkin [24] and Champarnaud [12]. Here, it is presented a more detailed proof.

Proposition 3.14. *Let $\alpha \in \mathsf{RE}$ be a regular expression over Σ. Then the set π, recursively defined as follows, is a support of α:*

$$\begin{aligned} &\pi(\emptyset) = \emptyset, && \pi(\alpha + \beta) = \pi(\alpha) \cup \pi(\beta), \\ &\pi(\varepsilon) = \emptyset, && \pi(\alpha\beta) = \pi(\alpha)\beta \cup \pi(\beta), \\ &\pi(a) = \{\varepsilon\}, && \pi(\alpha^*) = \pi(\alpha)\alpha^*. \end{aligned}$$

Proof. Let $\gamma_0 \in \mathsf{RE}$ be a regular expression over $\Sigma = \{a_1, \ldots, a_k\}$. In order to prove that a given set π is a support of γ_0, we need to prove that the equation (3.14) holds for γ_0 and for each element of π. We will proceed by induction on the structure of γ_0. If $\gamma_0 \equiv \varepsilon$ or $\gamma_0 \equiv \emptyset$, then we have

$$\gamma_0 = a_1\emptyset + \cdots + a_k\emptyset + \varepsilon(\gamma),$$

and we can conclude that the empty set $\emptyset$ is a support of $\emptyset$ and of ε. In the case that $\gamma_0 \equiv a_i \in \Sigma$, we have

$$a_i = a_1\emptyset + \cdots + a_i\varepsilon + \cdots + a_k\emptyset + \emptyset$$
$$\varepsilon = a_1\emptyset + \cdots + a_k\emptyset + \varepsilon,$$

and, for each $a_i \in \Sigma$, $\{\varepsilon\}$ is a support of it.

Given $\alpha_0, \beta_0 \in \mathsf{RE}$, suppose, by induction hypothesis, that $\pi(\alpha_0) = \{\alpha_1, \ldots, \alpha_n\}$ and $\pi(\beta_0) = \{\beta_1, \ldots, \beta_m\}$ are supports of α_0 and β_0, respectively. Therefore, from the definition of support, we have

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i), \quad \text{for all } i \in \{0, \ldots, n\},$$

and

$$\beta_j = a_1\beta_{1j} + \cdots + a_k\beta_{kj} + \varepsilon(\beta_j), \quad \text{for all } j \in \{0, \ldots, m\},$$

where, for all $l \in \{1, \ldots, k\}$, α_{li} and β_{lj} are sums of elements of $\pi(\alpha_0)$ and $\pi(\beta_0)$, respectively.

1. If $\gamma_0 \equiv \alpha_0 + \beta_0$, from the definition of the function π, then $\pi(\gamma_0) = \pi(\alpha_0) \cup \pi(\beta_0) = \{\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m\}$. Since $\pi(\alpha_0), \pi(\beta_0) \subseteq \pi(\gamma_0)$, it is obvious, by induction hypothesis, that for each α_i and β_j, where $i = 1, \ldots n$ and $j = 1, \ldots m$, the equation (3.14) holds. Thus, we just need to prove this for γ_0. We have the following equation

$$\begin{aligned}\gamma_0 = \alpha_0 + \beta_0 &= (a_1\alpha_{10} + \cdots + a_k\alpha_{k0} + \varepsilon(\alpha_0)) + (a_1\beta_{10} + \cdots + a_k\beta_{k0} + \varepsilon(\beta_0)) \\ &= a_1(\alpha_{10} + \beta_{10}) + \cdots + a_k(\alpha_{k0} + \beta_{k0}) + \varepsilon(\alpha_0 + \beta_0).\end{aligned}$$

   Since α_{l0} and β_{l0} $(l = 1, \ldots, k)$ are a sum of elements of $\pi(\alpha_0)$ and $\pi(\beta_0)$, respectively, then $\alpha_{l0} + \beta_{l0}$ is a sum of elements of $\pi(\alpha_0) \cup \pi(\beta_0) = \pi(\gamma_0)$. Thereby, the set $\pi(\alpha_0) \cup \pi(\beta_0)$ is a support of $\gamma_0 = \alpha_0 + \beta_0$.

2. If $\gamma_0 \equiv \alpha_0\beta_0$, then $\pi(\gamma_0) = \pi(\alpha_0)\beta_0 \cup \pi(\beta_0) = \{\alpha_1\beta_0, \ldots, \alpha_n\beta_0, \beta_1, \ldots, \beta_m\}$. Since $\pi(\beta_0) \subseteq \pi(\gamma_0)$, it is obvious that the equation (3.14) holds for each β_j $(j = 1, \ldots, m)$. So, we just need to prove that the equation holds for every $\alpha_i\beta_0$, $i = 0, \ldots, n$. Note that, in this way, we are also proving it for γ_0. Given that the following equation holds

$$\alpha_i\beta_0 = a_1\alpha_{1i}\beta_0 + \cdots + a_k\alpha_{ki}\beta_0 + \varepsilon(\alpha_i)\beta_0,$$

there are two possible cases: $\varepsilon(\alpha_i) = \emptyset$ or $\varepsilon(\alpha_i) = \varepsilon$. If $\varepsilon(\alpha_i) = \emptyset$, we have that $\varepsilon(\alpha_i)\beta_0 = \emptyset \cdot \beta_0 = \emptyset \cdot \varepsilon(\beta_0) = \varepsilon(\alpha_i\beta_0)$ and thus

$$\alpha_i\beta_0 = a_1\alpha_{1i}\beta_0 + \cdots + a_k\alpha_{ki}\beta_0 + \varepsilon(\alpha_i\beta_0).$$

Since α_{li}, for $l = 1, \ldots, k$, is a sum of elements of $\pi(\alpha_0)$, then $\alpha_{li}\beta_0$ is a sum of elements of $\pi(\alpha_0)\beta_0 \subseteq \pi(\gamma_0)$. If $\varepsilon(\alpha_i) = \varepsilon$, the following equation holds:

$$\begin{aligned}\alpha_i\beta_0 &= a_1\alpha_{1i}\beta_0 + \cdots + a_k\alpha_{ki}\beta_0 + \varepsilon(\alpha_i)(a_1\beta_{10} + \cdots + a_k\beta_{k0} + \varepsilon(\beta_0)) \\ &= a_1(\alpha_{1i}\beta_0 + \beta_{10}) + \cdots + a_k(\alpha_{ki}\beta_0 + \beta_{k0}) + \varepsilon(\alpha_i\beta_0).\end{aligned}$$

Since $\alpha_{li}\beta_0$ and β_{l0} $(l = 1, \ldots, k)$ are a sum of elements of $\pi(\alpha_0)\beta_0$ and $\pi(\beta_0)$, respectively, then $\alpha_{li}\beta_0 + \beta_{l0}$ is a sum of elements of $\pi(\alpha_0)\beta_0 \cup \pi(\beta_0) = \pi(\gamma_0)$. Proved both cases, we can conclude that $\pi(\alpha_0)\beta_0 \cup \pi(\beta_0)$ is a support of $\alpha_0\beta_0$.

3. If $\gamma_0 \equiv \alpha_0^*$, then $\pi(\gamma_0) = \pi(\alpha_0)\alpha_0^* = \{\alpha_1\alpha_0^*, \ldots, \alpha_n\alpha_0^*\}$. First, we will prove that the equation (3.14) holds for γ_0. We have that

$$\begin{aligned}\alpha_0^* &= (a_1\alpha_{10} + \cdots + a_k\alpha_{k0} + \varepsilon(\alpha_0))^* \\ &= (a_1\alpha_{10} + \cdots + a_k\alpha_{k0})^* \\ &= (a_1\alpha_{10} + \cdots + a_k\alpha_{k0})\alpha_0^* + \varepsilon \\ &= (a_1\alpha_{10} + \cdots + a_k\alpha_{k0})\alpha_0^* + \varepsilon(\alpha_0^*) \\ &= a_1\alpha_{10}\alpha_0^* + \cdots + a_k\alpha_{k0}\alpha_0^* + \varepsilon(\alpha_0^*)\end{aligned}$$

Since α_{l0} $(l = 1, \ldots, k)$ is a sum of elements of $\pi(\alpha_0)$, then $\alpha_{l0}\alpha_0^*$ is a sum of elements of $\pi(\alpha_0)\alpha_0^*$. Therefore, we just need to prove that the same holds for each $\alpha_i\alpha_0^*$ $(i = 1, \ldots, n)$. We have that

$$\begin{aligned}\alpha_i\alpha_0^* &= (a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i))\alpha_0^* \\ &= a_1\alpha_{1i}\alpha_0^* + \cdots + a_k\alpha_{ki}\alpha_0^* + \varepsilon(\alpha_i)\alpha_0^*\end{aligned}$$

As in 2., there are two possible cases: $\varepsilon(\alpha_i) = \emptyset$ or $\varepsilon(\alpha_i) = \varepsilon$. If $\varepsilon(\alpha_i) = \emptyset$, we have that $\varepsilon(\alpha_i)\alpha_0^* = \emptyset \cdot \alpha_0^* = \emptyset \cdot \varepsilon(\alpha_0^*) = \varepsilon(\alpha_i\alpha_0^*)$ and thus the following holds:

$$\alpha_i\alpha_0^* = a_1\alpha_{1i}\alpha_0^* + \cdots + a_k\alpha_{ki}\alpha_0^* + \varepsilon(\alpha_i\alpha_0^*)$$

Since α_{li}, $l = 1, \ldots, k$, is a sum of elements of $\pi(\alpha_0)$, then $\alpha_{li}\alpha_0^*$ is a sum of elements of $\pi(\alpha_0)\alpha_0^* = \pi(\gamma_0)$. If $\varepsilon(\alpha_i) = \varepsilon$, the following holds:

$$\alpha_i\alpha_0^* = a_1\alpha_{1i}\alpha_0^* + \cdots + a_k\alpha_{ki}\alpha_0^* + \varepsilon(\alpha_i)(a_1\alpha_{10}\alpha_0^* + \cdots + a_k\alpha_{k0}\alpha_0^* + \varepsilon(\alpha_0^*))$$

$$= a_1(\alpha_{1i}\alpha_0^* + \alpha_{10}\alpha_0^*) + \cdots + a_k(\alpha_{ki}\alpha_0^* + \alpha_{k0}\alpha_0^*) + \varepsilon(\alpha_i\alpha_0^*)$$

Since $\alpha_{li}\alpha_0^*$ and $\alpha_{l0}\alpha_0^*$ $(l = 1, \ldots, k)$ are a sum of elements of $\pi(\alpha_0)\alpha_0^*$, then also is $\alpha_{li}\alpha_0^* + \alpha_{l0}\alpha_0^*$. Therefore, have been proved both cases, we conclude that the set $\pi(\alpha_0)\alpha_0^*$ is a support of α_0^*.

□

Example 3.15. *For the regular expression $\alpha \equiv (ab + b)^*ab$, the set $\pi(\alpha)$ is computed as follows:*

$$\begin{aligned}
\pi((ab + b)^*ab) &= \pi((ab + b)^*)ab \cup \pi(ab) \\
&= \pi(ab + b)(ab + b)^*ab \cup \pi(a)b \cup \pi(b) \\
&= (\pi(ab) \cup \pi(b))(ab + b)^*ab \cup \pi(a)b \cup \pi(b) \\
&= (\pi(a)b \cup \pi(b) \cup \pi(b))(ab + b)^*ab \cup \pi(a)b \cup \pi(b) \\
&= (\{\varepsilon\}b \cup \{\varepsilon\} \cup \{\varepsilon\})(ab + b)^*ab \cup \{\varepsilon\} \cup \{\varepsilon\}b \\
&= \{b, \varepsilon\}(ab + b)^*ab \cup \{b, \varepsilon\} \\
&= \{b(ab + b)^*ab, (ab + b)^*ab, b, \varepsilon\}.
\end{aligned}$$

Mirkin [24] also proved that the size of $\pi(\alpha)$ is finite and $|\pi(\alpha)| \leq |\alpha|_\Sigma + 1$.

3.6 Regular Expressions to Finite Automata

The methods of transforming regular expressions into an equivalent finite automata are usually divided in two classes depending on whether ε-transitions are allowed or not in the resulting finite automaton. The first method presented is the Thompson's construction, which converts a regular expression into an ε-NFA. The second method, due to Glushkov, constructs an NFA without ε-transitions. The last two methods, Brzozowski's and Antimirov's, build a DFA and an NFA, respectively, and are based on derivatives.

3.6.1 Thompson's Automata

The Thompson's automaton [27] transforms a regular expression into an ε-NFA. This construction can be found in several books on automata and language theory [18,

22]. Here, we will present the construction proposed by Sheng Yu [28], which differs from the Thompson's construction, in the way that the number of final states is not restricted to one.

Definition 3.16. *(**Thompson's Automaton**) Let α be a regular expression over the alphabet Σ. The Thompson's automaton, $\mathcal{N}_\varepsilon$, is constructed recursively as follows:*

1. *If $\alpha \equiv \emptyset$, then $\mathcal{N}_\varepsilon = \langle \{q\}, \Sigma, q, \delta, \emptyset \rangle$, where $\delta(q, a) = \emptyset$, for each $a \in \Sigma \cup \{\varepsilon\}$.*

2. *If $\alpha \equiv \varepsilon$, then $\mathcal{N}_\varepsilon = \langle \{q\}, \Sigma, q, \delta, \{q\} \rangle$, where $\delta(q, a) = \emptyset$, for each $a \in \Sigma \cup \{\varepsilon\}$.*

3. *If $\alpha \equiv a$, $a \in \Sigma$, then $\mathcal{N}_\varepsilon = \langle \{q, f\}, \Sigma, q, \delta, \{f\} \rangle$, where $\delta(q, a) = f$ and it is the only defined transition.*

4. *Let $\alpha_1, \alpha_2 \in \mathsf{RE}$ be regular expressions over Σ, the $\mathcal{N}_{\varepsilon_1} = \langle Q_1, \Sigma, q_1, \delta_1, F_1 \rangle$ and $\mathcal{N}_{\varepsilon_2} = \langle Q_2, \Sigma, q_2, \delta_2, F_2 \rangle$ are the ε-NFA constructed for α_1 and α_2, respectively.*

   (a) *If $\alpha \equiv \alpha_1 + \alpha_2$, then $\mathcal{N}_\varepsilon = \langle Q, \Sigma, q, \delta, F \rangle$, such that $\mathcal{L}(\mathcal{N}_\varepsilon) = \mathcal{L}(\mathcal{N}_{\varepsilon_1}) \cup \mathcal{L}(\mathcal{N}_{\varepsilon_2})$ and where $Q = Q_1 \cup Q_2 \cup \{q\}$, $q \notin Q_1 \cup Q_2$, $F = F_1 \cup F_2$ and*

$$\begin{aligned} &\delta(q, \varepsilon) = \{q_1, q_2\}, \\ &\delta(s, a) = \delta_1(s, a), \quad \text{if } s \in Q_1 \text{ and } a \in \Sigma \cup \{\varepsilon\}, \\ &\delta(s, a) = \delta_2(s, a), \quad \text{if } s \in Q_2 \text{ and } a \in \Sigma \cup \{\varepsilon\}. \end{aligned}$$

   (b) *If $\alpha \equiv \alpha_1 \alpha_2$, then $\mathcal{N}_\varepsilon = \langle Q, \Sigma, q, \delta, F \rangle$, such that $\mathcal{L}(\mathcal{N}_\varepsilon) = \mathcal{L}(\mathcal{N}_{\varepsilon_1}) \mathcal{L}(\mathcal{N}_{\varepsilon_2})$ and where $Q = Q_1 \cup Q_2$, $q = q_1$, $F = F_2$ and*

$$\begin{aligned} &\delta(s, a) = \delta_1(s, a), \quad \text{if } s \in Q_1 \text{ and } a \in \Sigma, \text{ or } s \in Q_1 \setminus F_1 \text{ and } a = \varepsilon, \\ &\delta(s, \varepsilon) = \delta_1(s, \varepsilon) \cup \{q_2\}, \quad \text{if } s \in F_1, \\ &\delta(s, a) = \delta_2(s, a), \quad \text{if } s \in Q_2 \text{ and } a \in \Sigma \cup \{\varepsilon\}. \end{aligned}$$

   (c) *If $\alpha \equiv \alpha_1^*$, then $\mathcal{N}_\varepsilon = \langle Q, \Sigma, q, \delta, F \rangle$, such that $\mathcal{L}(\mathcal{N}_\varepsilon) = \mathcal{L}(\mathcal{N}_{\varepsilon_1})^*$ and where $Q = Q_1 \cup \{q\}$, $q \notin Q_1$, $F = F_1 \cup \{q\}$ and*

$$\begin{aligned} &\delta(q, \varepsilon) = \{q_1\}, \\ &\delta(s, \varepsilon) = \delta_1(s, \varepsilon) \cup \{q_1\}, \quad \text{if } s \in F_1, \\ &\delta(s, a) = \delta_1(s, a), \quad \text{if } s \in Q_1 \text{ and } a \in \Sigma, \text{ or } s \in Q_1 \setminus F_1 \text{ and } a = \varepsilon. \end{aligned}$$

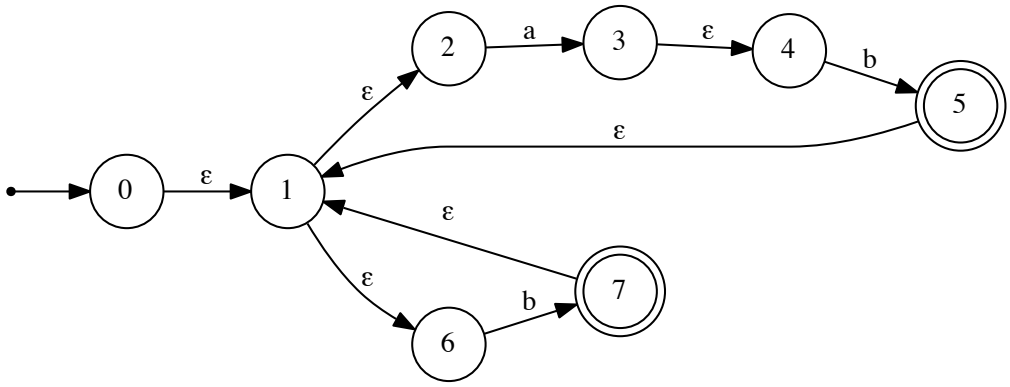


Figure 3.1: Diagram of Thompson's automaton for $(ab + b)^*$.

The automaton $\mathcal{N}_\varepsilon$ accepts the language $\mathcal{L}(\alpha)$.

In terms of the size of the regular expressions, Thompson's construction takes linear space and time.

The diagram corresponding to the Thompson's automaton of $(ab + b)^*$ over $\{a, b\}$ is shown in Figure 3.1.

3.6.2 Glushkov's automata

The *Glushkov's automaton* (or *position automaton*) was independently introduced by Glushkov [17] and McNaughton and Yamada [23]. While the previous construction builds an ε-NFA, the Glushkov's automaton is an NFA without ε-transitions.

The *marked regular expression* of a regular expression α, denoted by $\widetilde{\alpha}$, is the regular expression obtained by marking each alphabet symbol with its position in α. For example, given $\alpha \equiv (ab + b)^*ab$, the corresponding marked version is $\widetilde{\alpha} \equiv (a_1b_2 + b_3)^*a_4b_5$. The set $\widetilde{\Sigma}$ is the alphabet of the marked regular expression. For example, $\widetilde{\alpha} \equiv (a_1b_2 + b_3)^*a_4b_5$ has alphabet $\widetilde{\Sigma} = \{a_1, b_2, b_3, a_4, b_5\}$. The same notation is used to remove the markings, i.e., $\widetilde{\widetilde{\alpha}} \equiv \alpha$. The set of positions of α is defined as $pos(\alpha) = \{1, 2, \ldots, |\alpha_\Sigma|\}$ and $pos_0(\alpha) = pos(\alpha) \cup \{0\}$.

Given a regular expression α, the sets $\mathit{first}(\widetilde{\alpha})$, $\mathit{last}(\widetilde{\alpha})$ and $\mathit{follow}(\widetilde{\alpha})$ are defined as follows:

$$\begin{aligned}
&\mathit{first}(\widetilde{\alpha}) = \{i \mid a_i w \in \mathcal{L}(\widetilde{\alpha}), \text{ where } w \in \widetilde{\Sigma}^*\},\\
&\mathit{last}(\widetilde{\alpha}) = \{i \mid w a_i \in \mathcal{L}(\widetilde{\alpha}), \text{ where } w \in \widetilde{\Sigma}^*\},\\
&\mathit{follow}(\widetilde{\alpha}) = \{(i,j) \mid u a_i a_j v \in \mathcal{L}(\widetilde{\alpha}), \text{ where } u, v \in \widetilde{\Sigma}^*\}.
\end{aligned}$$

The set $\mathit{first}(\widetilde{\alpha})$ gives the position of the alphabetic symbols that occur at the beginning of the words in $\mathcal{L}(\alpha)$. This set is defined inductively as:

$$\begin{aligned}
&\mathit{first}(\emptyset) = \emptyset, && \mathit{first}(\alpha + \beta) = \mathit{first}(\alpha) \cup \mathit{first}(\beta),\\
&\mathit{first}(\varepsilon) = \emptyset, && \mathit{first}(\alpha\beta) = \mathit{first}(\alpha) \cup \mathit{first}(\beta), \text{ if } \varepsilon(\alpha) = \varepsilon,\\
&\mathit{first}(a_i) = \{i\}, && \mathit{first}(\alpha\beta) = \mathit{first}(\alpha), \text{ if } \varepsilon(\alpha) = \emptyset.\\
&\mathit{first}(\alpha^*) = \mathit{first}(\alpha),
\end{aligned}$$

By contrast, the set $\mathit{last}(\widetilde{\alpha})$ gives the position of the alphabetic symbols that occur at the end of the words in $\mathcal{L}(\alpha)$. This set is defined inductively as:

$$\begin{aligned}
&\mathit{last}(\emptyset) = \emptyset, && \mathit{last}(\alpha + \beta) = \mathit{last}(\alpha) \cup \mathit{last}(\beta),\\
&\mathit{last}(\varepsilon) = \emptyset, && \mathit{last}(\alpha\beta) = \mathit{last}(\alpha) \cup \mathit{last}(\beta), \text{ if } \varepsilon(\beta) = \varepsilon,\\
&\mathit{last}(a_i) = \{i\}, && \mathit{last}(\alpha\beta) = \mathit{last}(\alpha), \text{ if } \varepsilon(\beta) = \emptyset.\\
&\mathit{last}(\alpha^*) = \mathit{last}(\alpha),
\end{aligned}$$

The set $\mathit{follow}(\widetilde{\alpha})$ gives pairs of positions that are consecutive in words of $\mathcal{L}(\widetilde{\alpha})$. Given $(i,j) \in \mathit{follow}(\alpha)$, a_i precedes a_j in w. This set is defined inductively as follows:

$$\begin{aligned}
&\mathit{follow}(\emptyset) = \mathit{follow}(\varepsilon) = \mathit{follow}(a_i) = \emptyset,\\
&\mathit{follow}(\alpha + \beta) = \mathit{follow}(\alpha) \cup \mathit{follow}(\beta),\\
&\mathit{follow}(\alpha\beta) = \mathit{follow}(\alpha) \cup \mathit{follow}(\beta) \cup \mathit{last}(\alpha) \times \mathit{first}(\beta),\\
&\mathit{follow}(\alpha^*) = \mathit{follow}(\alpha) \cup \mathit{last}(\alpha) \times \mathit{first}(\alpha).
\end{aligned}$$

Definition 3.17. *(**Glushkov's automaton**) Let $\alpha \in \mathsf{RE}$ be a regular expression over Σ. The Glushkov's automaton for α is the NFA $\mathcal{N}_{pos} = \langle pos_0(\alpha), \Sigma, 0, \delta_{pos}, F\rangle$, where $\delta_{pos}(\alpha) = \{(0, \widetilde{a_j}, j) \mid j \in \mathrm{first}(\alpha)\} \cup \{(i, \widetilde{a_j}, j) \mid (i,j) \in \mathrm{follow}(\alpha)\}$ and, if $\varepsilon(\alpha) = \varepsilon$, the final states are $F = \mathrm{last}(\alpha) \cup \{0\}$, otherwise, $F = \mathrm{last}(\alpha)$. The automaton $\mathcal{N}_{pos}$ accepts the language $\mathcal{L}(\alpha)$.*

The number of states of $\mathcal{N}_{pos}$ is exactly $n+1$, where $n = |\alpha|_\Sigma$. The number of transitions is, in the worst case, n^2+2. Therefore, the time-complexity of this construction must be at least $O(n^2)$. A naive implementation of the Glushkov's automaton is $O(n^3)$. Brüggemann-Klein proposed a construction [9] of complexity $O(n^2+m)$, where $m = |\alpha|$. Broda et al. [6] presented an alternative recursive definition for $\mathit{follow}(\alpha)$, which allows simple implementations in time $O(n^2)$.

Example 3.18. *For $\alpha \equiv (ab+b)^*ab$ over $\Sigma = \{a,b\}$, the corresponding marked version of α is $\widetilde{\alpha} \equiv (a_1b_2+b_3)^*a_4b_5$ and $pos_0(\alpha) = \{0,1,2,3,4,5\}$. The sets $\mathit{first}(\widetilde{\alpha})$, $\mathit{last}(\widetilde{\alpha})$ and $\mathit{follow}(\widetilde{\alpha})$ are*

$$\begin{aligned}
&\mathit{first}(\widetilde{\alpha}) = \{1,3,4\},\\
&\mathit{last}(\widetilde{\alpha}) = \{5\},\\
&\mathit{follow}(\widetilde{\alpha}) = \{(1,2),(2,1),(2,3),(2,4),(3,1),(3,3),(3,4),(4,5)\}.
\end{aligned}$$

The Glushkov's automaton for α is the NFA $\mathcal{N}_{pos} = \langle pos_0(\alpha), \Sigma, 0, \delta_{pos}, F\rangle$, where the transition function is

$$\begin{aligned}
\delta_{pos} =& \{(0,\widetilde{a_j},j) \mid j \in \mathit{first}(\alpha)\} \cup \{(i,\widetilde{a_j},j) \mid (i,j) \in \mathit{follow}(\alpha)\}\\
=& \{(0,a,1),(0,b,3),(0,a,4)\} \cup \{(1,b,2),(2,a,1),(2,b,3),(2,a,4),\\
& (3,a,1),(3,b,3),(3,a,4),(4,b,5)\}
\end{aligned}$$

and, since $\varepsilon(\alpha) = \emptyset$, the set of final states is $F = \mathit{last}(\alpha) = \{5\}$. The diagram of $\mathcal{N}_{pos}$ is shown in Figure 3.2.

3.6.3 Brzozowski's Automata

Brzozowski [10] presented a construction of a deterministic finite automaton using derivatives. This construction builds an automaton where the states are derivatives of a given regular expression and the transitions of a state by a symbol is given by the derivative of that state w.r.t. the symbol. Note that, as seen in the section 3.2, a regular expression has a finite number of ACI_+-dissimilar derivatives. This dissimilarity will be used in the construction, in order to build an automaton with a finite number of states.

Definition 3.19. ***(Brzozowski's Automaton)*** *Let $\alpha \in \mathsf{RE}$ be a regular expression over Σ. The Brzozowski's automaton for α is the DFA $\mathcal{D} = \langle Q, \Sigma, q_0, \delta, F\rangle$, where the*

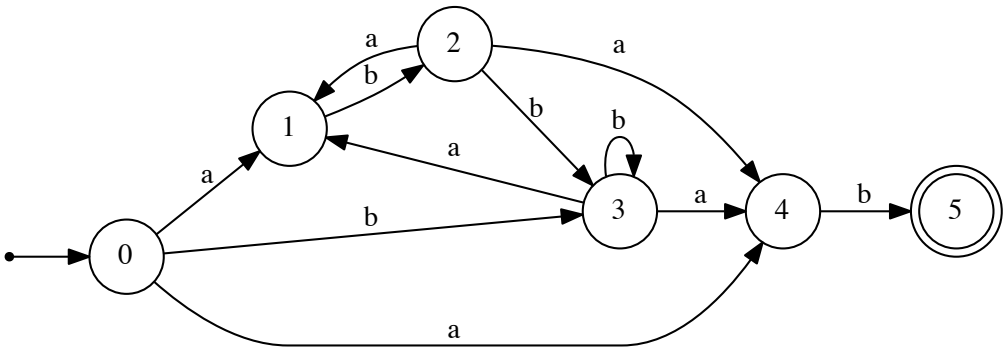


Figure 3.2: Diagram of the Glushkov's automaton for $(ab + b)^*ab$.

set of states is $Q = D_{ACI_+}(\alpha)$*, the initial state is* $q_0 = \alpha$*, the transition function is defined by* $\delta(q, a) = d_a(q)$*, for all* $a \in \Sigma$ *and* $q \in D_{ACI_+}(\alpha)$*, and the set of final states is* $F = \{q \in D_{ACI_+}(\alpha) \mid \varepsilon(q) = \varepsilon\}$*. The automaton* $\mathcal{D}$ *accepts the language* $\mathcal{L}(\alpha)$*.*

Example 3.20. *Let* $\alpha \equiv (ab + b)^*ab$ *over* $\Sigma = \{a, b\}$*. Let* $\mathcal{D} = \langle Q, \Sigma, q_0, \delta, F\rangle$ *be the Brzozowski's DFA of* α*. The initial state of* $\mathcal{D}$ *is* $q_0 = \alpha$*. The transition function* δ *is defined as follows:*

$$\delta((ab + b)^*ab, a) = d_a((ab + b)^*ab) = b(ab + b)^*ab + b,$$
$$\delta((ab + b)^*ab, b) = d_b((ab + b)^*ab) = (ab + b)^*ab,$$
$$\delta(b(ab + b)^*ab + b, a) = d_a(b(ab + b)^*ab + b) = \emptyset,$$
$$\delta(b(ab + b)^*ab + b, b) = d_b(b(ab + b)^*ab + b) = (ab + b)^*ab + \varepsilon,$$
$$\delta((ab + b)^*ab + \varepsilon, a) = d_a((ab + b)^*ab + \varepsilon) = b(ab + b)^*ab + b,$$
$$\delta((ab + b)^*ab + \varepsilon, b) = d_b((ab + b)^*ab + \varepsilon) = (ab + b)^*ab,$$
$$\delta(\emptyset, a) = d_a(\emptyset) = \emptyset,$$
$$\delta(\emptyset, b) = d_b(\emptyset) = \emptyset.$$

And thus, the set of states of $\mathcal{D}$ *is* $Q = D_{ACI_+}(\alpha) = \{b(ab + b)^*ab + b, (ab + b)^*ab, (ab + b)^*ab + \varepsilon\}$ *and the set of final states is* $F = \{(ab + b)^*ab + \varepsilon\}$*. The diagram of* $\mathcal{D}$ *is depicted in Figure 3.3.*

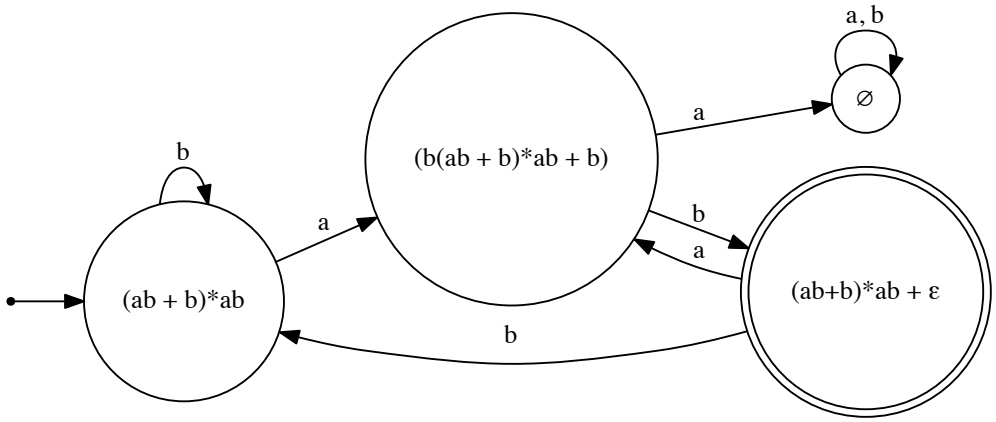


Figure 3.3: Diagram of the Brzozowski's automaton for $(ab + b)^*ab$.

3.6.4 Partial Derivative's Automata

Antimirov [2] presents a construction for an NFA based on partial derivatives. This automaton is the nondeterministic version of the Brzozowski's DFA. The construction of the automaton is given below.

Definition 3.21. *(**Partial Derivative's Automaton**) Let $\alpha \in \mathsf{RE}$ be a regular expression over Σ. The partial derivative's automaton for α is the NFA $\mathcal{N} = \langle Q, \Sigma, q_0, \delta, F\rangle$, where the set of states is $Q = \mathcal{PD}(\alpha)$, the initial state is $q_0 = \alpha$, the transition function is defined by $\delta(q, a) = \partial_a(q)$, for all $a \in \Sigma$ and $q \in \mathcal{PD}(\alpha)$, and the set of final states is $F = \{q \in \mathcal{PD}(\alpha) \mid \varepsilon(q) = \varepsilon\}$. The automaton $\mathcal{N}$ accepts the language $\mathcal{L}(\alpha)$.*

The partial derivative's automaton for a regular expression α has at most $|\alpha|_\Sigma + 1$ states, as proved by Antimirov [2].

As mentioned in the section 3.5, Champarnaud and Ziadi [12] proved that Mirkin's prebases and partial derivatives lead to identical nondeterministic finite automata. Since the set of states of partial derivative's automaton is $Q = \mathcal{PD}(\alpha)$ and the set of Mirkin's construction is the prebase of α, then $\mathcal{PD}(\alpha) = \pi(\alpha) \cup \{\alpha\}$.

Since the linear form is an efficient way to compute partial derivatives, the transition function of the partial derivative's automaton of a regular expression α can be computed using it:

$$\delta(\alpha, a) = \{\alpha' \mid (a, \alpha') \in lf(\alpha)\}, \quad \text{for all } \alpha' \in \pi(\alpha) \text{ and } a \in \Sigma,$$

where each $(a, \alpha') \in lf(\alpha)$ denotes a transition from the state α by the symbol a to the state α'.

Example 3.22. *Let α be the regular expression $(ab + b)^*ab$ over $\Sigma = \{a, b\}$. The corresponding partial derivative's automaton for α is the automaton $\mathcal{N} = \langle Q, \Sigma, q_0, \delta, F\rangle$, where: the set of states is $Q = \pi(\alpha) = \{(ab+b)^*ab, b(ab+b)^*ab, b, \varepsilon\}$, as computed in Example 3.15, the initial state is $q_0 = \alpha$. Since*

$$\begin{aligned}
&lf((ab + b)^*ab) = \{(a, b(ab + b)^*ab), (a, b), (b, (ab + b)^*ab)\},\\
&lf(b(ab + b)^*ab) = \{(b, (ab + b)^*ab)\},\\
&lf(b) = \{(b, \varepsilon)\},\\
&lf(\varepsilon) = \{\},
\end{aligned}$$

the transition function is defined as follows:

$$\begin{aligned}
&\delta((ab + b)^*ab, a) = \{b(ab + b)^*ab, a\}, && \delta(b, a) = \emptyset,\\
&\delta((ab + b)^*ab, b) = \{(ab + b)^*ab\}, && \delta(b, b) = \{\varepsilon\},\\
&\delta(b(ab + b)^*ab, a) = \emptyset, && \delta(\varepsilon, a) = \emptyset,\\
&\delta(b(ab + b)^*ab, b) = \{(ab + b)^*ab\}, && \delta(\varepsilon, b) = \emptyset.
\end{aligned}$$

The set of final states is $F = \{\varepsilon\}$. The diagram of the automaton is shown in Figure 3.4.

3.7 FAdo

The FAdo system provides tools to manipulate regular expressions. As for finite automata, there is a module defining regular expressions - the module `reex`. The main class implemented in this module is the class `regexp`.

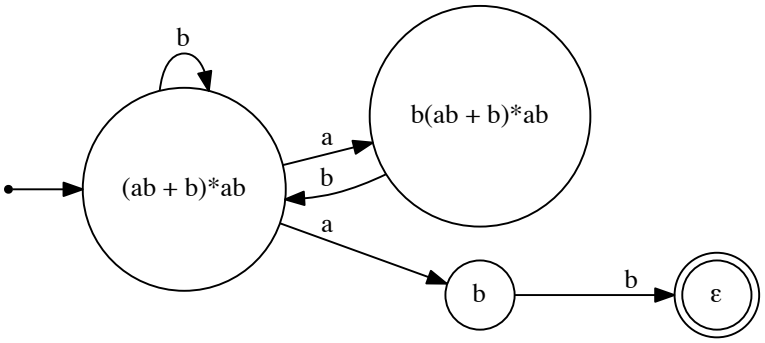


Figure 3.4: Diagram of the partial derivative's automaton for $(ab + b)^*ab$.

The classes `atom`, `epsilon`, `emptyset`, `concat`, `disj` and `star`, representing the alphabetic symbols, the empty word, the empty set, the concatenation, the disjunction and the Kleene star, respectively, inherit from `regexp`. The class diagram in Figure 3.5 shows this inheritance.

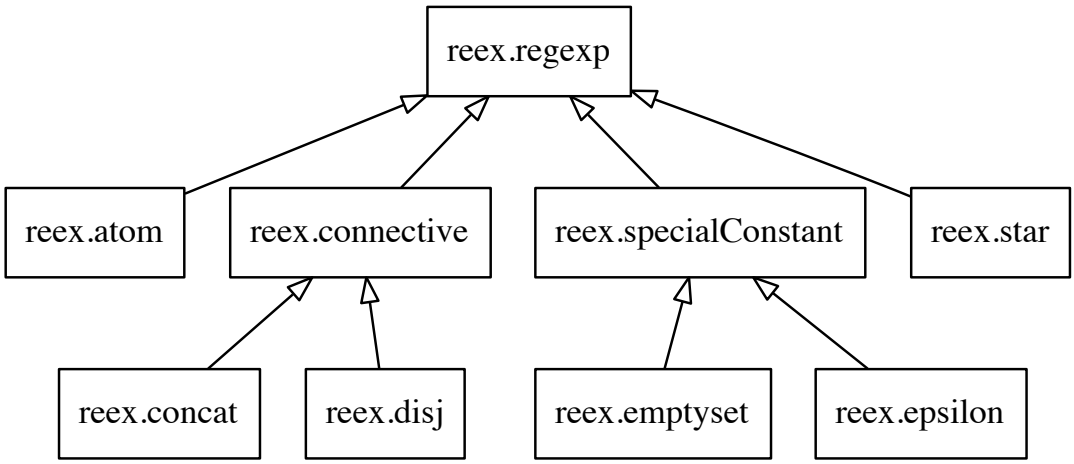


Figure 3.5: Class diagram of the FAdo module reex.

Using the function `reex.str2regexp(str)`, we can convert a string into a FAdo regular expression, where `str` is the string that we want to convert. The empty word, the empty set, the disjunction and the Kleene star are represented by the strings @epsilon, @emptyset, + and $*$, respectively. The string corresponding to $(a(ab+c)^*+\varepsilon)$ is a(ab+c)$*$+@epsilon.

```
>>> from FAdo.reex import *
>>> re = str2regexp("a(ab+c)* + @epsilon")
>>> re
disj(concat(atom(a),star(disj(concat(atom(a),atom(b)),atom(c))
   )),epsilon())
>>> print re
(a ((a b) + c)*) + @epsilon
```

The methods `regexp.treeLength()` and `regexp.alphabeticLength()` returns the size and alphabetic size of a regular expression, respectively. We can set the alphabet of a regular expression by using `regexp.setSigma(set)`.

We can test if a regular expression is equivalent to another (`regexp.compare(re)`) and if a regular expression possesses the empty word property (`regexp.ewp()`). To verify if two regular expressions are syntactically identical we use the operator ==.

```
>>> re.compare(str2regexp("@epsilon + a(ab+c)*")
True
>>> re == str2regexp("@epsilon + a(ab+c)*")
False
>>> re.ewp()
True
```

Furthermore, it is possible to evaluate the derivative and the set of partial derivatives w.r.t. a symbol of a regular expression (`regexp.derivative(symbol)` and `regexp.partialDerivatives(symbol)`, respectively), the linear form of a regular expression (`regexp.linearForm()`), the set of all partial derivatives (`regexp.PD()`) and the support (`regexp.support()`) of a regular expression.

```
>>> re = str2regexp("aa+b")
>>> re.derivative('a')
disj(concat(epsilon(),atom(a)),emptyset())
>>> re.partialDerivative('a')
set([atom(a)])
>>> re.linearForm('a')
{'a': set([atom(a)]), 'b': set([epsilon()])}
>>> re.support()
set([atom(a), epsilon()])
```

```
>>> re.support().union({re}) == re.PD()
True
```

There are several methods available to convert a regular expression into a finite automata. Among these, we can build the Thompson's (`regexp.nfaThompson()`), the Glushkov's (`regexp.nfaPosition()`) and the partial derivate's (`regexp.nfaPD()`) automaton. Since the operator == defined for the above classes is not ACI_+-dissimilar, the construction of the Brzozowski's automaton is not possible in FAdo.

Chapter 4

Regular Expressions with Intersection

Since the regular languages are closed under intersection, a regular expression can be enriched with the binary operator $\cap$, representing the intersection. The definition of a regular expression with intersection is given bellow.

Definition 4.1. ***(Regular Expression with Intersection)*** *Let Σ be an alphabet. A regular expression with intersection over Σ is given by the following grammar:*

$$\alpha := \emptyset \mid \varepsilon \mid a \in \Sigma \mid (\alpha + \alpha) \mid (\alpha \cap \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^*.$$

The language $\mathcal{L}(\alpha)$ associated with α is defined inductively as defined for RE (see page 15), adding the case $\mathcal{L}(\alpha \cap \beta) = \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$. The set of all regular expressions with intersection is denoted by $\mathsf{RE}_\cap$.

We assume that the operator $\cap$ has higher precedence than $+$ and lower precedence than $\cdot$. Hence, we have the following precedences: $* > \cdot > \cap > +$. The definitions of equivalence, size, alphabetic size and empty word property (e.w.p.) are defined in exactly the same way as for RE. However, given $\alpha, \beta \in \mathsf{RE}_\cap$, the function $\varepsilon : \mathsf{RE}_\cap \to \{\varepsilon, \emptyset\}$, defining the e.w.p., is defined recursively as for RE, adding the following cases:

$$\varepsilon(\alpha \cap \beta) = \varepsilon, \text{ if } \varepsilon(\alpha) = \varepsilon(\beta) = \varepsilon,$$
$$\varepsilon(\alpha \cap \beta) = \emptyset, \text{ if } \varepsilon(\alpha) = \emptyset \text{ or } \varepsilon(\beta) = \emptyset.$$

4.1 Algebra of $\mathsf{RE}_\cap$

The set of regular expression $\mathsf{RE}_\cap$ over Σ constitutes an algebraic structure $(\mathsf{RE}_\cap, +, \cdot, \cap, *, \emptyset, \varepsilon)$, where $(\mathsf{RE}_\cap, +, \cdot, *, \emptyset, \varepsilon)$ is a Kleene algebra and $\cap$ is a binary operator satisfying the axioms (A_{12}) to (A_{24}), which are presented below.

Antimirov [3] presents an axiomatization for the regular expressions with intersection $\mathsf{RE}_\cap$, the system AX, and also proved that AX is complete and sound. The axiomatic system AX is an extension of the system F_1, presented in the section 3.1. The axiomatic system AX is a set of 24 equational axioms, the axioms of the system F_1, (A_1)-(A_{11}), and the 13 axioms presented below:

$$(\varepsilon \cap \beta) = \emptyset \wedge (\alpha = \beta\alpha + \gamma) \Rightarrow \alpha = \beta^*\gamma, \tag{A_{12}}$$
$$\varepsilon \cap (\alpha \cdot \beta) = (\varepsilon \cap \alpha) \cap \beta, \tag{A_{13}}$$
$$\varepsilon \cap \alpha^* = \varepsilon, \tag{A_{14}}$$
$$\varepsilon \cap a = \emptyset, \tag{A_{15}}$$
$$\emptyset \cap \alpha = \emptyset, \tag{A_{16}}$$
$$\alpha \cap \alpha = \alpha, \tag{A_{17}}$$
$$\alpha \cap \beta = \beta \cap \alpha, \tag{A_{18}}$$
$$\alpha \cap (\beta \cap \gamma) = (\alpha \cap \beta) \cap \gamma, \tag{A_{19}}$$
$$\alpha \cap (\beta + \gamma) = (\alpha \cap \beta) + (\alpha \cap \gamma), \tag{A_{20}}$$
$$\alpha + (\alpha \cap \beta) = \alpha, \tag{A_{21}}$$
$$(a_1 \cdot \alpha) \cap (a_2 \cdot \beta) = (a_1 \cap a_2)(\alpha \cap \beta), \tag{A_{22}}$$
$$(\alpha \cdot a_1) \cap (\beta \cdot a_2) = (\alpha \cap \beta) \cdot (a_1 \cap a_2), \tag{A_{23}}$$
$$a_i \cap a_j = \emptyset, \qquad a_i, a_j \in \Sigma \text{ such that } a_i \neq a_j, \tag{A_{24}}$$

for all $a, a_1, a_2 \in \Sigma$ and all $\alpha, \beta, \gamma \in \mathsf{RE}_\cap$.

In RE, the empty word property cannot be interpreted as universal Horn formula. However in $\mathsf{RE}_\cap$ this problem disappears, since the e.w.p. can be expressed equationally by

$$\alpha \text{ possesses e.w.p.} \iff \varepsilon \cap \alpha = \varepsilon,$$
$$\alpha \text{ does not possess e.w.p.} \iff \varepsilon \cap \alpha = \emptyset.$$

In this way, the solution of equations rule can be expressed by an equational implication, which is given by the axiom (A_{12}).

Note that the operator $\cdot$ is not distributive over the operator $\cap$, that is, the following equations are, in general, not valid in $\mathsf{RE}_\cap$:

$$\alpha \cdot (\beta \cap \gamma) = (\alpha \cap \beta) \cdot (\alpha \cap \gamma),$$
$$(\alpha \cap \beta) \cdot \gamma = (\alpha \cap \gamma) \cdot (\beta \cap \gamma),$$

for all $\alpha, \beta, \gamma \in \mathsf{RE}_\cap$. It is only valid a restricted distributivity given by the axioms A_{22} and A_{23}.

Since, $\varepsilon \cap \emptyset = \emptyset \cap \varepsilon = \emptyset$, by (A_{16}) and (A_{18}), and that $\varepsilon \cap \varepsilon = \varepsilon$, by (A_{17}). Thus, it is possible to redefine the inductive case $\alpha \cap \beta$ of the function $\varepsilon : \mathsf{RE}_\cap \to \{\emptyset, \varepsilon\}$, as follows:

$$\varepsilon(\alpha \cap \beta) = \varepsilon(\alpha) \cap \varepsilon(\beta).$$

4.2 Derivatives

As mentioned in Remark 3.5, the recursive definition of derivative was also defined by Brzozowski for the operator intersection. Since the definition for all inductive cases of $\mathsf{RE}_\cap$, excepting for intersection, was presented in the section 3.2, for convenience, only the new case is given bellow.

Definition 4.2. ***(Derivative)*** *Let $\alpha \in \mathsf{RE}_\cap$ be a regular expression over Σ. The derivative of α w.r.t. a symbol $a \in \Sigma$, written $d_a(\alpha)$, is defined recursively as for RE, adding the inductive case:*

$$d_a(\alpha \cap \beta) = d_a(\alpha) \cap d_a(\beta).$$

Example 4.3. *The derivative of $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$ w.r.t. a is given by*

$$\begin{aligned} d_a(\alpha) &= d_a((b + ab + aab + abab) \cap (ab)^*) \\ &= d_a((b + ab + aab + abab)) \cap d_a((ab)^*) \\ &= (b + ab + bab) \cap b(ab)^*. \end{aligned}$$

Given $\alpha \in \mathsf{RE}_\cap$ over Σ, the derivative of a regular expression α is extended in the same way for a word $w \in \Sigma^*$ as for a regular expression in RE and the property $\mathcal{L}(d_w(\alpha)) =$

$w^{-1}\mathcal{L}(\alpha)$ holds. For every regular expression α, the equivalence $\alpha = \sum_{a\in\Sigma} ad_a(\alpha) + \varepsilon(\alpha)$ holds, cf. equation (3.1). Moreover, every regular expression in $\mathsf{RE}_\cap$ has only a finite number of ACI_+-dissimilar derivatives.

4.3 Partial Derivatives

Caron et al. [11] extended the notion of partial derivative to regular expressions with intersection and complement. While the partial derivative of simple (non-extended) regular expressions (Definition 3.6) returns a set of regular expressions, the partial derivative proposed by the authors returns a set of sets of regular expressions. This change of codomain is not necessary for partial derivatives of regular expressions with intersection only. Therefore, here, we present a recursive definition for partial derivative for regular expressions with intersection, returning a set of regular expressions, and study some properties of this partial derivatives.

Firstly, let the operation $\owedge : 2^{\mathsf{RE}_\cap} \times 2^{\mathsf{RE}_\cap} \to 2^{\mathsf{RE}_\cap}$ be an extension of the intersection for sets of regular expressions. This operation is defined recursively as follows:

$$\emptyset \owedge S = \emptyset, \tag{4.1}$$

$$\{\alpha\} \owedge S = \{\alpha \cap \beta \mid \beta \in S\}, \tag{4.2}$$

$$(\{\alpha\} \cup S) \owedge S' = (\{\alpha\} \owedge S') \cup (S \owedge S'), \tag{4.3}$$

for all $S, S' \subseteq 2^{\mathsf{RE}_\cap}$ and all $\alpha, \beta \in \mathsf{RE}_\cap \setminus \{\emptyset\}$. The operator $\owedge$ is also right-distributive over the operator $\cup$. Thereby, the operator $\owedge$ is distributive over $\cup$.

Lemma 4.4. *Given $S, S' \subseteq \mathsf{RE}_\cap$, $\mathcal{L}(S \owedge S') = \mathcal{L}(S) \cap \mathcal{L}(S')$.*

Proof. First recall that, for every $\alpha_1, \ldots, \alpha_n \in \mathsf{RE}_\cap$, the equality $\mathcal{L}(\{\alpha_1, \ldots, \alpha_n\}) = \mathcal{L}(\alpha_1 + \cdots + \alpha_n)$ holds. Let $S = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathsf{RE}_\cap$ and $S' = \{\beta_1, \ldots, \beta_m\} \subseteq \mathsf{RE}_\cap$ be sets of regular expressions. Then, we have the following:

$$\begin{aligned}
\mathcal{L}(S \owedge S') &= \mathcal{L}(\{\alpha_1, \ldots, \alpha_n\} \owedge \{\beta_1, \ldots, \beta_m\}) \\
&= \mathcal{L}(\{\alpha_1 \cap \beta_1, \ldots, \alpha_1 \cap \beta_m, \ldots, \alpha_n \cap \beta_1, \ldots, \alpha_n \cap \beta_m\} \\
&= \mathcal{L}(\alpha_1 \cap \beta_1 + \cdots + \alpha_1 \cap \beta_m + \cdots + \alpha_n \cap \beta_1 + \cdots + \alpha_n \cap \beta_m) \\
&= \mathcal{L}(\alpha_1 \cap (\beta_1 + \cdots + \beta_m) + \cdots + \alpha_n \cap (\beta_1 + \cdots + \beta_m))
\end{aligned}$$

$$
\begin{aligned}
&= \mathcal{L}(\alpha_1 + \cdots + \alpha_n) \cap (\beta_1 + \cdots + \beta_m)) \\
&= \mathcal{L}(\alpha_1 + \cdots + \alpha_n) \cap \mathcal{L}(\beta_1 + \cdots + \beta_m) \\
&= \mathcal{L}(\{\alpha_1, \ldots, \alpha_n\}) \cap \mathcal{L}(\{\beta_1, \ldots, \beta_m\}) \\
&= \mathcal{L}(S) \cap \mathcal{L}(S').
\end{aligned}
$$

□

Definition 4.5. ***(Partial Derivatives)*** *Let $\alpha \in \mathsf{RE}_\cap$ be a regular expression over Σ. The partial derivative of α w.r.t. a symbol $a \in \Sigma$, written $\partial_a(\alpha)$, is defined recursively as for RE, adding the inductive case:*

$$\partial_a(\alpha \cap \beta) = \partial_a(\alpha) \barwedge \partial_a(\beta).$$

The definition of the set of partial derivatives is extended to words, sets of words and to sets of regular expressions in the same way as for simple regular expressions.

Example 4.6. *The set of partial derivatives of $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$ w.r.t. a is given by*

$$
\begin{aligned}
\partial_a(\alpha) &= \partial_a((b + ab + aab + abab) \cap (ab)^*) \\
&= \partial_a((b + ab + aab + abab)) \barwedge \partial_a((ab)^*) \\
&= \{b, ab, bab\} \barwedge \{b(ab)^*\} \\
&= \{b \cap b(ab)^*, ab \cap b(ab)^*, bab \cap b(ab)^*\}.
\end{aligned}
$$

The relation between the partial derivative and the derivative of a regular expressions with intersection is given by Proposition 3.8. This relation is obtained by proving that the partial derivative of a regular expression $\alpha \in \mathsf{RE}_\cap$ over Σ denotes the same language as the left quotient of that regular expression, since $\mathcal{L}(d_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$, for every $w \in \Sigma^*$. In order to prove that $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$, we first need to prove that $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$, for $a \in \Sigma$.

Lemma 4.7. *For all $a \in \Sigma$ and all $\alpha \in \mathsf{RE}_\cap$, $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$.*

Proof. We proceed by induction on the structure of α. Antimirov [2] proved this for all cases, excepting for $\alpha \cap \beta$. Thus, here we will only present the proof for the inductive

case $\alpha \cap \beta$. Let $\alpha, \beta \in \mathsf{RE}_\cap$ and $a \in \Sigma$ and suppose by inductive hypothesis that $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$ and $\mathcal{L}(\partial_a(\beta)) = a^{-1}\mathcal{L}(\beta)$. Then

$$\begin{aligned}\mathcal{L}(\partial_a(\alpha \cap \beta)) &= \mathcal{L}(\partial_a(\alpha) \barwedge \partial_a(\beta)) \\ &= \mathcal{L}(\partial_a(\alpha)) \cap \mathcal{L}(\partial_a(\beta)) \\ &= a^{-1}\mathcal{L}(\alpha) \cap a^{-1}\mathcal{L}(\beta) \\ &= a^{-1}\mathcal{L}(\alpha \cap \beta).\end{aligned}$$

□

Proposition 4.8. *Given $w \in \Sigma^*$ and $\alpha \in \mathsf{RE}_\cap$, $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$.*

Proof. The proof follows in the same way as the proof for the case that $\alpha \in \mathsf{RE}$ (Proposition 3.8). □

The next lemma states an useful property of ∂_w extended to sets of regular expressions with intersection.

Lemma 4.9. *For all $S, S' \subseteq \mathsf{RE}_\cap$ over Σ and $a \in \Sigma$, the following property holds*

$$\partial_a(S \barwedge S') = \partial_a(S) \barwedge \partial_a(S'). \tag{4.4}$$

Proof. Let $a \in \Sigma$ and let $S, S' \subseteq \mathsf{RE}_\cap$, such that $S = \{\alpha_1, \ldots, \alpha_n\}$ and $S' = \{\beta_1, \ldots, \beta_m\}$. Then,

$$\begin{aligned}\partial_a(S \barwedge S') =& \partial_a(\{\alpha_1, \ldots, \alpha_n\} \barwedge \{\beta_1, \ldots, \beta_m\}) \\ =& \partial_a(\{\alpha_1 \cap \beta_1, \ldots, \alpha_1 \cap \beta_m, \ldots, \alpha_n \cap \beta_1, \ldots, \alpha_n \cap \beta_m\}) \\ =& \partial_a(\alpha_1 \cap \beta_1) \cup \cdots \cup \partial_a(\alpha_1 \cap \beta_m) \cup \cdots \cup \\ & \partial_a(\alpha_n \cap \beta_1) \cup \cdots \cup \partial_a(\alpha_n \cap \beta_m) \\ =& (\partial_a(\alpha_1) \barwedge \partial_a(\beta_1)) \cup \cdots \cup (\partial_a(\alpha_1) \barwedge \partial_a(\beta_m)) \cup \cdots \cup \\ & (\partial_a(\alpha_n) \barwedge \partial_a(\beta_1)) \cup \cdots \cup (\partial_a(\alpha_n) \barwedge \partial_a(\beta_m)) \\ =& \bigcup_{\alpha_i \in S, \beta_j \in S'} \{\alpha_i' \cap \beta_j' \mid \alpha_i' \in \partial_a(\alpha_i), \beta_j' \in \partial_a(\beta_j)\} \\ =& \bigcup_{\alpha_i \in S} \partial_a(\alpha_i) \barwedge \bigcup_{\beta_j \in S'} \partial_a(\beta_j) \\ =& \partial_a(S) \barwedge \partial_a(S').\end{aligned}$$

□

Recall that $Suf(w)$ is the set of all non-empty suffixes of w, being defined as $Suf(w) = \{v \in \Sigma^+ \mid \exists u \in \Sigma^* : uv = w\}$. The following lemma presents some properties of the function ∂_w.

Lemma 4.10. *For every regular expressions $\alpha, \beta \in RE_\cap$ and word $w \in \Sigma^+$, ∂_w satisfies the following:*

$$\partial_w(\alpha + \beta) = \partial_w(\alpha) \cup \partial_w(\beta), \tag{4.5}$$

$$\partial_w(\alpha \cap \beta) = \partial_w(\alpha) \cap \partial_w(\beta), \tag{4.6}$$

$$\partial_w(\alpha\beta) \subseteq \partial_w(\alpha)\beta \cup \bigcup_{v \in Suf(w)} \partial_v(\beta), \tag{4.7}$$

$$\partial_w(\alpha^*) \subseteq \bigcup_{v \in Suf(w)} \partial_v(\alpha)\alpha^*. \tag{4.8}$$

Proof. Antimirov[2] proved the equations (4.5), (4.7) and (4.8). Thus, we only present the proof for the equation (4.6).

The proof of the statement $\partial_w(\alpha \cap \beta) = \partial_w(\alpha) \cap \partial_w(\beta)$ is done by induction on w. If $w = \varepsilon$, then $\partial_\varepsilon(\alpha \cap \beta) = \{\alpha \cap \beta\} = \{\alpha\} \cap \{\beta\} = \partial_\varepsilon(\alpha) \cap \partial_\varepsilon(\beta)$. Suppose that $\partial_w(\alpha \cap \beta) = \partial_w(\alpha) \cap \partial_w(\beta)$ holds for a given w, we prove it for $w' = wa$, where $a \in \Sigma$. So, from Lemma 4.9, we have the following

$$\begin{aligned}\partial_{wa}(\alpha \cap \beta) &= \partial_a(\partial_w(\alpha \cap \beta)) = \partial_a(\partial_w(\alpha) \cap \partial_w(\beta)) \\ &= \partial_a(\partial_w(\alpha)) \cap \partial_a(\partial_w(\beta)) = \partial_{wa}(\alpha) \cap \partial_{wa}(\beta).\end{aligned}$$

□

As previously shown, for a simple regular expression, the set ∂^+ can be defined inductively by systems of equations. In this section, we prove that this is not possible for a regular expression with intersection. For now, we present some inclusions for this set.

Proposition 4.11. *For every regular expression $\alpha, \beta \in RE_\cap$, the following inclusions hold:*

$$\partial^+(\alpha + \beta) \subseteq \partial^+(\alpha) \cup \partial^+(\beta), \tag{4.9}$$

$$\partial^+(\alpha \cap \beta) \subseteq \partial^+(\alpha) \cap \partial^+(\beta), \tag{4.10}$$

$$\partial^+(\alpha\beta) \subseteq \partial^+(\alpha)\beta \cup \partial^+(\beta), \tag{4.11}$$

$$\partial^+(\alpha^*) \subseteq \partial^+(\alpha)\alpha^*. \tag{4.12}$$

Proof. First note that, given a set E and a regular expression $\alpha \in \mathsf{RE}_\cap$, if $\partial_w(\alpha) \subseteq E$, for all $w \in \Sigma^+$, then it holds that $\bigcup_{w\in\Sigma^+} \partial_w(\alpha) \subseteq E$ and thus $\partial^+(\alpha) \subseteq E$. Moreover, we know that for every $w \in \Sigma^+$, $\partial_w(\alpha) \subseteq \partial^+(\alpha)$, since $\partial^+(\alpha) = \bigcup_{w\in\Sigma^+} \partial_w(\alpha)$. Let $\alpha, \beta \in \mathsf{RE}_\cap$ be regular expressions over Σ. In order to prove the inclusions (4.9), (4.10), (4.11) and (4.12), the facts mentioned above are used. The proof of each inclusion is given, respectively, by the following four proofs:

1. From equation (4.5), for all $w \in \Sigma^+$, the following holds:

$$\partial_w(\alpha + \beta) = \partial_w(\alpha) \cup \partial_w(\beta) \subseteq \partial^+(\alpha) \cup \partial^+(\beta).$$

   And thus, we can conclude that $\partial^+(\alpha \cup \beta) \subseteq \partial^+(\alpha) \cup \partial^+(\beta)$.

2. In the same way, from equation (4.6), for all $w \in \Sigma^+$, the following holds:

$$\partial_w(\alpha \cap \beta) \subseteq \partial_w(\alpha) \cap \partial_w(\beta) \subseteq \partial^+(\alpha) \cap \partial^+(\beta).$$

   And then, $\partial^+(\alpha \cap \beta) \subseteq \partial^+(\alpha) \cap \partial^+(\beta)$.

3. From equation (4.7), for all $w \in \Sigma^+$, the following holds:

$$\begin{aligned}\partial_w(\alpha\beta) &\subseteq \partial_w(\alpha)\beta \cup \bigcup_{v\in Suf(w)} \partial_v(\beta) \\ &\subseteq \partial^+(\alpha)\beta \cup \partial^+(\beta).\end{aligned}$$

   Thus, $\partial^+(\alpha\beta) \subseteq \partial^+(\alpha)\beta \cup \partial^+(\beta)$.

4. Finally, from equation (4.8), for all $w \in \Sigma^+$, the following holds:

$$\partial_w(\alpha^*) \subseteq \bigcup_{v\in Suf(w)} \partial_v(\alpha)\alpha^* \subseteq \partial^+(\alpha)\alpha^*.$$

   Therefore, we have that $\partial^+(\alpha^*) \subseteq \partial^+(\alpha)\alpha^*$.

□

Example 4.12. *For the regular expression* $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$ *over* $\{a, b\}$*, we have*

$$\partial_a(\alpha) = \{b \cap b(ab)^*, ab \cap b(ab)^*, bab \cap b(ab)^*\}, \qquad \partial_b(\alpha) = \emptyset,$$
$$\partial_a(b \cap b(ab)^*) = \emptyset, \qquad \partial_b(b \cap b(ab)^*) = \{\varepsilon \cap (ab)^*\},$$

$\partial_a(ab \cap b(ab)^*) = \emptyset$, $\partial_b(ab \cap b(ab)^*) = \emptyset$,
$\partial_a(bab \cap b(ab)^*) = \emptyset$, $\partial_b(bab \cap b(ab)^*) = \{ab \cap (ab)^*\}$,
$\partial_a(\varepsilon \cap (ab)^*) = \emptyset$, $\partial_b(\varepsilon \cap (ab)^*) = \emptyset$,
$\partial_a(ab \cap (ab)^*) = \{b \cap b(ab)^*\}$, $\partial_b(ab \cap (ab)^*) = \emptyset$,

and thus $\partial^+(\alpha) = \{bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*, ab \cap (ab)^*, \varepsilon \cap (ab)^*\}$. *However, being* $\beta = (b + ab + aab + abab)$, *it follows that*

$$\begin{aligned}\partial^+(\beta) \barwedge \partial^+((ab)^*) = &\{bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*,\\ &\varepsilon \cap b(ab)^*, bab \cap (ab)^*, ab \cap (ab)^*, b \cap (ab)^*, \varepsilon \cap (ab)^*\}.\end{aligned}$$

And then, we can conclude that $\partial^+(\alpha) \subsetneqq \partial^+(b + ab + aab + abab) \barwedge \partial^+((ab)^*)$.

4.4 Linear Form

Since, partial derivatives are extensible to intersection, the notion of linear form can also be extended. In this section we present the definition of the linear form of a regular expression with intersection and prove that it computes all the partial derivatives w.r.t. every alphabet symbol of the regular expression.

Let us define $\mathrm{Mon}_\cap = \Sigma \times \mathsf{RE}_\cap$ as the set of all monomials for a given alphabet Σ. Recall that a monomial is a pair (a, α), where $a \in \Sigma$ and $\alpha \in \mathsf{RE}_\cap$, and that the properties given by equations (3.11) and (3.12) hold for monomials.

Furthermore, we need to extend the operator $\barwedge$ to sets of monomials. Let the operation $\barwedge : 2^{Mon_\cap} \times 2^{Mon_\cap} \to 2^{Mon_\cap}$ be an extension of intersection for sets of monomials. The operator is defined recursively as:

$$\emptyset \barwedge S = \emptyset, \tag{4.13}$$

$$\{(a, \alpha)\} \barwedge S = \{(a, \alpha \cap \beta) \mid (a, \beta) \in S\}, \tag{4.14}$$

$$(\{(a, \alpha)\} \cup S) \barwedge S' = (\{(a, \alpha)\} \barwedge S') \cup (S \barwedge S'), \tag{4.15}$$

where $a, b \in \Sigma$, $\alpha, \beta \in \mathsf{RE}_\cap \setminus \{\emptyset\}$ and $S, S' \subseteq 2^{\mathsf{RE}_\cap}$. The extended operator $\barwedge$ is also right-distributive over the operator $\cup$. And thus the operator $\barwedge$ is distributive over $\cup$.

Definition 4.13. *(**Linear Form**) The function* $lf : \mathsf{RE}_\cap \to 2^{Mon_\cap}$ *returns the linear form of a regular expression with intersection and it is defined recursively as for* RE,

adding the inductive case:

$$lf(\alpha \cap \beta) = lf(\alpha) \barwedge lf(\beta),$$

where $\alpha, \beta \in \mathsf{RE}$ *over* Σ.

Example 4.14. *The linear form of* $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$ *is*

$$\begin{aligned} lf(\alpha) =& lf((b + ab + aab + abab) \cap (ab)^*) \\ =& lf((b + ab + aab + abab)) \barwedge lf((ab)^*) \\ =& \{(b, \varepsilon), (a, b), (a, ab), (a, bab)\} \barwedge \{(a, b(ab)^*)\} \\ =& \{(a, b \cap b(ab)^*), (a, ab \cap b(ab)^*), (a, bab \cap b(ab)^*)\}. \end{aligned}$$

As for the case of simple regular expression, the set of partial derivatives w.r.t. a symbol of a regular expression with intersection can also be defined using the function *lf*. This is showed by the next proposition.

Proposition 4.15. *Given* $\alpha \in \mathsf{RE}_\cap$ *over* Σ *and* $a \in \Sigma$*, then*

$$\partial_a(\alpha) = \{\alpha' \in \mathsf{RE}_\cap \setminus \{\emptyset\} \mid (a, \alpha') \in lf(\alpha)\}.$$

Proof. We proceed by induction on the structure of α. The proof for all cases, except for $\alpha \cap \beta$, was given by Antimirov [2]. Therefore, here, we only present the proof for the inductive case $\alpha \cap \beta$. First note that, for a regular expression $\alpha \in \mathsf{RE}_\cap$ over Σ and $a \in \Sigma$, $\partial_a(\alpha) = \{\alpha' \in \mathsf{RE}_\cap \setminus \{\emptyset\} \mid (a, \alpha') \in lf(\alpha)\}$ is the same as having

$$lf(\alpha) = \bigcup_{a \in \Sigma} \{(a, \alpha') \mid \alpha' \in \partial_a(\alpha)\}. \tag{4.16}$$

Thus, given $\alpha, \beta \in \mathsf{RE}_\cap$ over $\Sigma = \{a_1, \ldots, a_k\}$, suppose by inductive hypothesis that the equation (4.16) holds for α and β. Then, we have

$$\begin{aligned} lf(\alpha \cap \beta) =& lf(\alpha) \barwedge lf(\beta) \\ =& \bigcup_{a \in \Sigma} \{(a, \alpha') \mid \alpha' \in \partial_a(\alpha)\} \barwedge \bigcup_{a \in \Sigma} \{(a, \beta') \mid \beta' \in \partial_a(\beta)\} \\ =& \left(\bigcup_{\alpha_{1i} \in \partial_{a_1}(\alpha)} (a_1, \alpha_{1i}) \cup \cdots \cup \bigcup_{\alpha_{ki} \in \partial_{a_k}(\alpha)} (a_k, \alpha_{ki}) \right) \barwedge \\ & \barwedge \left(\bigcup_{\beta_{1j} \in \partial_{a_1}(\beta)} (a_1, \beta_{1j}) \cup \cdots \cup \bigcup_{\beta_{kj} \in \partial_{a_k}(\beta)} (a_k, \beta_{kj}) \right) \end{aligned}$$

$$
\begin{aligned}
=&\left(\bigcup_{\alpha_{1i}\in\partial_{a_1}(\alpha)}(a_1,\alpha_{1i})\cap\bigcup_{\beta_{1j}\in\partial_{a_1}(\beta)}(a_1,\beta_{1j})\right)\cup\cdots\cup\\
&\cup\left(\bigcup_{\alpha_{1i}\in\partial_{a_1}(\alpha)}(a_1,\alpha_{1i})\cap\bigcup_{\beta_{kj}\in\partial_{a_k}(\beta)}(a_k,\beta_{kj})\right)\cup\cdots\cup\\
&\cup\left(\bigcup_{\alpha_{ki}\in\partial_{a_k}(\alpha)}(a_k,\alpha_{ki})\cap\bigcup_{\beta_{1j}\in\partial_{a_1}(\beta)}(a_1,\beta_{1j})\right)\cup\cdots\cup\\
&\cup\left(\bigcup_{\alpha_{ki}\in\partial_{a_k}(\alpha)}(a_k,\alpha_{ki})\cap\bigcup_{\beta_{kj}\in\partial_{a_k}(\beta)}(a_k,\beta_{kj})\right)\\
=&\{(a_1,\alpha_{1i}\cap\beta_{1j})\mid\alpha_{1i}\in\partial_{a_1}(\alpha),\beta_{1j}\in\partial_{a_1}(\beta)\}\cup\cdots\cup\\
&\cup\{(a_k,\alpha_{ki}\cap\beta_{kj})\mid\alpha_{ki}\in\partial_{a_k}(\alpha),\beta_{kj}\in\partial_{a_k}(\beta)\}\\
=&\bigcup_{a\in\Sigma}\{(a,\alpha'\cap\beta')\mid\alpha'\in\partial_a(\alpha),\beta'\in\partial_a(\beta)\}\\
=&\bigcup_{a\in\Sigma}\{(a,\alpha'\cap\beta')\mid\alpha'\cap\beta'\in\partial_a(\alpha)\cap\partial(\beta)\}\\
=&\bigcup_{a\in\Sigma}\{(a,\alpha'\cap\beta')\mid\alpha'\cap\beta'\in\partial_a(\alpha\cap\beta)\}.
\end{aligned}
$$

□

4.5 Systems of Equations

In this section, we present a recursive definition of a support of regular expressions with intersection. We also prove an upper bound for the size of this support and that the upper bound is tight by exhibiting a witness.

Let $\alpha_0 \in \mathsf{RE}_\cap$ be a regular expression over $\Sigma = \{a_1, \ldots a_k\}$. As seen in the section 3.5, a support of α_0 is defined as being a set $\{\alpha_1, \ldots, \alpha_n\}$, which for each $\alpha_i \in \{\alpha_0\} \cup \{\alpha_1, \ldots, \alpha_n\}$, the following equation holds:

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i),$$

where each α_{li}, with $l = 1, \ldots, k$, is a, possibly empty, sum of elements of $\{\alpha_1, \ldots, \alpha_n\}$. The next proposition extends the notion of support for regular expressions in $\mathsf{RE}_\cap$.

Proposition 4.16. *Let $\alpha \in RE_\cap$ be a regular expression over Σ. Then the set π, defined recursively as follows, is a support of α:*

$$\begin{aligned}
&\pi(\emptyset) = \emptyset, && \pi(\alpha + \beta) = \pi(\alpha) \cup \pi(\beta), && \pi(\alpha^*) = \pi(\alpha)\alpha^*.\\
&\pi(\varepsilon) = \emptyset, && \pi(\alpha \cap \beta) = \pi(\alpha) \owedge \pi(\beta),\\
&\pi(a) = \{\varepsilon\}, && \pi(\alpha\beta) = \pi(\alpha)\beta \cup \pi(\beta),
\end{aligned}$$

Proof. We proceed by induction on the structure of the regular expression $\gamma_0 \in \mathsf{RE}_\cap$ The proof for all inductive cases of γ_0, excluding $\alpha_0 \cap \beta_0$, was presented in Proposition 3.14, so here we need prove that case. In order to demonstrate that a set π is a support of a regular expression, we need to prove that the equation (3.14) holds for the regular expression and for each element in π.

Given the regular expressions $\alpha_0, \beta_0 \in \mathsf{RE}_\cap$ over $\Sigma = \{a_1, \ldots, a_k\}$, suppose by induction hypothesis that $\pi(\alpha_0) = \{\alpha_1, \ldots, \alpha_n\}$ and $\pi(\beta_0) = \{\beta_1, \ldots, \beta_m\}$ are supports of α_0 and β_0, respectively. Therefore, from the definition of support, we have

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i), \quad \text{for all } i \in \{0, \ldots, n\},$$

and

$$\beta_j = a_1\beta_{1j} + \cdots + a_k\beta_{kj} + \varepsilon(\beta_j), \quad \text{for all } j \in \{0, \ldots, m\},$$

where, for all $l \in \{1, \ldots, k\}$, α_{li} and β_{lj} are sums of elements of $\pi(\alpha_0)$ and $\pi(\beta_0)$, respectively.

In the case that $\gamma_0 \equiv \alpha_0 \cap \beta_0$, we have that $\pi(\gamma_0) = \pi(\alpha_0 \cap \beta_0) = \pi(\alpha_0) \owedge \pi(\beta_0) = \{\alpha_1 \cap \beta_1, \ldots, \alpha_1 \cap \beta_m, \ldots, \alpha_n \cap \beta_1, \ldots, \alpha_n \cap \beta_m\}$. Notice that, if we prove that the equation (3.14) holds for each $\alpha_i \cap \beta_j$, such that $i = 0, \ldots, n$ and $j = 0, \ldots, m$, we are proving it for the regular expression $\gamma_0 \equiv \alpha_0 \cap \beta_0$ and for each element belonging to π. That is what we will do. So, we have

$$\begin{aligned}
\alpha_i \cap \beta_j =& (a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i)) \cap (a_1\beta_{1j} + \cdots + a_k\beta_{kj} + \varepsilon(\beta_j)) && (4.17)\\
=& (a_1\alpha_{1i} \cap a_1\beta_{1j}) + \cdots + (a_1\alpha_{1j} \cap a_k\alpha_{ki}) + (a_1\alpha_{1j} \cap \varepsilon(\beta_j)) + \cdots +\\
&+ (a_k\alpha_{ki} \cap a_1\beta_{1j}) + \cdots + (a_k\alpha_{ki} \cap a_k\beta_{kj}) + (a_k\alpha_{ki} \cap \varepsilon(\beta_j)) + \cdots +\\
&+ (\varepsilon(\alpha_i) \cap a_1\beta_{1j}) + \cdots + (\varepsilon(\alpha_i) \cap a_k\beta_{kj}) + (\varepsilon(\alpha_i) \cap \varepsilon(\beta_j)) && (4.18)\\
=& (a_1 \cap a_1)(\alpha_{1i} \cap \beta_{1j}) + \cdots + (a_1 \cap a_k)(\alpha_{1j} \cap \alpha_{ki}) +\\
&+ (a_1 \cap \varepsilon(\beta_j))(\alpha_{1j} \cap \varepsilon) + \ldots + (a_k \cap a_1)(\alpha_{ki} \cap \beta_{1j}) + \cdots +
\end{aligned}$$

$$
\begin{aligned}
&+ (a_k \cap a_k)(\alpha_{ki} \cap \beta_{kj}) + (a_k \cap \varepsilon(\beta_j))(\alpha_{ki} \cap \varepsilon) + \cdots + \\
&+ (\varepsilon(\alpha_i) \cap a_1)(\varepsilon \cap \beta_{1j}) + \cdots + (\varepsilon(\alpha_i) \cap a_k)(\varepsilon \cap \beta_{kj}) + \\
&+ (\varepsilon(\alpha_i) \cap \varepsilon(\beta_j)) && (4.19)\\
=&(a_1 \cap a_1)(\alpha_{1i} \cap \beta_{1j}) + \cdots + (a_k \cap a_k)(\alpha_{ki} \cap \beta_{kj}) + (\varepsilon(\alpha_i) \cap \varepsilon(\beta_j)) && (4.20)\\
=&a_1(\alpha_{1i} \cap \beta_{1j}) + \cdots + a_k(\alpha_{kj} \cap \beta_{kj}) + \varepsilon(\alpha_i \cap \beta_j), && (4.21)
\end{aligned}
$$

for each $i = 0, \ldots, n$ and $j = 0, \ldots, m$. In these calculations, we made use of some axioms of the system F_1. In the step (4.18), the axiom A_{20} was used, corresponding to the distributivity of $+$ over $\cap$. In the step (4.19), A_7, A_{13} and A_{22} were the axioms used. In the step (4.20), we made use of A_{15}, A_{16} and A_{24}. In the last step (4.21), it was used the axiom A_{17}, corresponding to the idempotence of $\cap$.

We know that each α_{li} and β_{lj}, $l = 1, \ldots, k$, is a sum of elements of $\pi(\alpha_0)$ and $\pi(\beta_0)$, respectively. Let $I_{li} \subseteq \{1, \ldots, n\}$ and $J_{lj} \subseteq \{1, \ldots, m\}$ be sets of indexes of regular expressions in $\pi(\alpha_0)$ and $\pi(\beta_0)$, respectively, such that $\alpha_{li} = \sum_{i' \in I_{li}} \alpha_{i'}$ and $\beta_{lj} = \sum_{j' \in J_{lj}} \beta_{j'}$. Since, for all $l = 1, \ldots, k$, $i = 0, \ldots, n$ and $j = 0, \ldots, m$, the following equation holds

$$\alpha_{li} \cap \beta_{lj} = \sum_{i' \in I_{li}} \alpha_{i'} \cap \sum_{j' \in J_{lj}} \beta_{j'} = \sum_{i' \in I_{li}, j' \in J_{lj}} (\alpha_{i'} \cap \beta_{j'}),$$

the regular expression $\alpha_{li} \cap \beta_{lj}$ is a sum of expressions in $\pi(\alpha_0) \owedge \pi(\beta_0) = \pi(\gamma_0)$ and thus the set $\pi(\alpha_0) \owedge \pi(\beta_0)$ is a support of $\alpha_0 \cap \beta_0$. □

Example 4.17. *For the regular expression $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$, the set $\pi(\alpha)$ is computed as follows:*

$$
\begin{aligned}
\pi(\alpha) =&\pi((b + ab + aab + abab) \cap (ab)^*)\\
=&\pi((b + ab + aab + abab)) \owedge \pi((ab)^*)\\
=&(\pi(b) \cup \pi(ab) \cup \pi(aab) \cup \pi(abab)) \owedge \pi((ab)^*)\\
=&(\{\varepsilon\} \cup \{b, \varepsilon\} \cup \{ab, b, \varepsilon\} \cup \{bab, ab, b, \varepsilon\}) \owedge \{b(ab)^*, (ab)^*\}\\
=&\{bab, ab, b, \varepsilon\} \owedge \{b(ab)^*, (ab)^*\}\\
=&\{bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*, \varepsilon \cap b(ab)^*, bab \cap (ab)^*,\\
&\quad ab \cap (ab)^*, b \cap (ab)^*, \varepsilon \cap (ab)^*\}.
\end{aligned}
$$

Let us define $|\alpha|_\cap$ as the number of occurrences of the operator $\cap$ in α. The next proposition provides an upper bound on the cardinality of $\pi(\alpha)$, $\alpha \in \mathsf{RE}_\cap$, proving that the set is finite.

Proposition 4.18. *For every $\alpha \in \mathsf{RE}_\cap$, the inequality $|\pi(\alpha)| \leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1}$ holds.*

Proof. We proceed by induction on the structure of the regular expression α. The proof that the statement holds for the base cases ε, $\emptyset$ and $a \in \Sigma$ is trivial. Assume that the result holds for some $\alpha, \beta \in \mathsf{RE}_\cap$. We will make use of the fact that $2^m + 2^n \leq 2^{m+n+1}$, for any $m, n \geq 0$.

Case $\alpha + \beta$:

$$\begin{aligned}
|\pi(\alpha+\beta)| &= |\pi(\alpha) \cup \pi(\beta)| = |\pi(\alpha)| + |\pi(\beta)| \\
&\leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1} + 2^{|\beta|_\Sigma - |\beta|_\cap - 1} \\
&\leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1 + |\beta|_\Sigma - |\beta|_\cap - 1 + 1} \\
&= 2^{|\alpha+\beta|_\Sigma - |\alpha+\beta|_\cap - 1}.
\end{aligned}$$

Case $\alpha \cap \beta$:

$$\begin{aligned}
|\pi(\alpha \cap \beta)| &= |\pi(\alpha) \cap \pi(\beta)| \leq |\pi(\alpha)| \cdot |\pi(\beta)| \\
&\leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1} \cdot 2^{|\beta|_\Sigma - |\beta|_\cap - 1} \\
&= 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1 + |\beta|_\Sigma - |\beta|_\cap - 1} \\
&= 2^{|\alpha\cap\beta|_\Sigma - (|\alpha\cap\beta|_\cap - 1) - 2} \\
&= 2^{|\alpha\cap\beta|_\Sigma - |\alpha\cap\beta|_\cap - 1}.
\end{aligned}$$

Case $\alpha\beta$:

$$\begin{aligned}
|\pi(\alpha\beta)| &= |\pi(\alpha)\beta \cup \pi(\beta)| \leq |\pi(\alpha)\beta| + |\pi(\beta)| \\
&= |\pi(\alpha)| + |\pi(\beta)| \\
&\leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1} + 2^{|\beta|_\Sigma - |\beta|_\cap - 1} \\
&\leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1 + |\beta|_\Sigma - |\beta|_\cap - 1 + 1} \\
&= 2^{|\alpha\beta|_\Sigma - |\alpha\beta|_\cap - 1}.
\end{aligned}$$

Case α^*:

$$\begin{aligned}
|\pi(\alpha^*)| &= |\pi(\alpha)\alpha^*| = |\pi(\alpha)| \leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1} \\
&= 2^{|\alpha^*|_\Sigma - |\alpha^*|_\cap - 1}.
\end{aligned}$$

□

The next example presents a family of regular expressions, denoted r_n, that proves that the upper bound proved in Proposition 4.18 is tight.

Example 4.19. *Let the regular expression $r_n \in \mathsf{RE}_\cap$ be defined inductively by*

$$\begin{aligned} r_0 &\equiv a^*a, \\ r_n &\equiv r_{n-1} \cap a^*a. \end{aligned}$$

Using the recursively definition of support it is straightforward that we have the following

$$\begin{aligned} \pi(r_0) &= \pi(a^*a) = \{a^*a, \varepsilon\}, \\ \pi(r_n) &= \underbrace{\{a^*a, \varepsilon\} \cap \cdots \cap \{a^*a, \varepsilon\}}_{n+1} \\ &= \{\alpha_1 \cap \cdots \cap \alpha_n \mid \forall i \in \{1, \ldots, n\}.\, \alpha_i \in \{a^*a, \varepsilon\}\} \quad \textit{for } n \geq 1, \end{aligned}$$

and thus $|\pi(r_0)| = 2$ and $|\pi(r_n)| = |\pi(r_0)|^{n+1} = 2^{n+1}$. Note that $|r_n|_\Sigma = 2n + 2$ and $|r_n|_\cap = n$. So, we have that the support of r_n is given by the following

$$\begin{aligned} |\pi(r_n)| &= 2^{n+1} = 2^{2n+2-n-1} \\ &= 2^{|r_n|_\Sigma - |r_n|_\cap - 1}, \end{aligned}$$

which is the upper bound given in Proposition 4.18.

4.5.1 Support and Partial Derivatives

As seen in the section 3.5, Champarnaud and Ziadi proved that, for every regular expression $\alpha \in \mathsf{RE}$, $\pi(\alpha) \cup \{\alpha\} = \mathcal{PD}(\alpha)$. Here, we will prove that this relation is not verified for every regular expression in $\mathsf{RE}_\cap$, that is, $\mathcal{PD}(\alpha') \subsetneqq \pi(\alpha') \cup \{\alpha'\}$.

Since $\mathcal{PD}(\alpha) = \partial^+(\alpha) \cup \{\alpha\}$, then relating $\mathcal{PD}(\alpha)$ with $\pi(\alpha) \cup \{\alpha\}$ is the same as to relate $\partial^+(\alpha)$ with $\pi(\alpha)$. Throughout this section, the latter approach will be preferred.

Proposition 4.20. *Given $\alpha \in \mathsf{RE}_\cap$, $\partial^+(\alpha) \subseteq \pi(\alpha)$.*

Proof. The proof proceeds by induction on the structure of α. It is trivial that $\partial^+(\emptyset) = \pi(\emptyset)$, $\partial^+(\varepsilon) = \pi(\varepsilon)$ and $\partial^+(a) = \pi(a)$, for a symbol $a \in \Sigma$. Assuming that $\partial^+(\alpha) \subseteq \pi(\alpha)$ and $\partial^+(\beta) \subseteq \pi(\beta)$ holds, for $\alpha, \beta \in \mathsf{RE}_\cap$, consider the following cases:

1. For the case $\alpha + \beta$, from the inclusion (4.9), we have the following:

$$\partial^+(\alpha + \beta) \subseteq \partial^+(\alpha) \cup \partial^+(\beta) \subseteq \pi(\alpha) \cup \pi(\beta).$$

2. For the case $\alpha \cap \beta$, since the inclusion (4.10) holds, we have:

$$\partial^+(\alpha \cap \beta) \subseteq \partial^+(\alpha) \cap \partial^+(\beta) \subseteq \pi(\alpha) \cap \pi(\beta).$$

3. For the case $\alpha\beta$, from the inclusion (4.11), we have:

$$\partial^+(\alpha\beta) \subseteq \partial^+(\alpha)\beta \cup \partial^+(\beta) \subseteq \pi(\alpha)\beta \cup \pi(\beta).$$

4. Finally, for α^*, we can conclude, from the inclusion (4.12), that:

$$\partial^+(\alpha^*) \subseteq \partial^+(\alpha)\alpha^* \subseteq \pi(\alpha)\alpha^*.$$

□

Since, for every regular expression $\alpha \in \mathsf{RE}_\cap$, the set $\pi(\alpha)$ is finite, Proposition 4.20 also proves that the set $\partial^+(\alpha)$ is finite.

The next example demonstrates that there exists $\alpha \in \mathsf{RE}_\cap$ such that $\pi(\alpha) \neq \partial^+(\alpha)$.

Example 4.21. *For the regular expression $\alpha \equiv (b + ab + aab + abab) \cap (ab)^*$), from Example 4.17, we have*

$$\begin{aligned}\pi(\alpha) = \{&bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*, \varepsilon \cap b(ab)^*, bab \cap (ab)^*, \\ &ab \cap (ab)^*, b \cap (ab)^*, \varepsilon \cap (ab)^*\}.\end{aligned}$$

However, from Example 4.12, $\partial^+(\alpha) = \{bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*, ab \cap (ab)^*, \varepsilon \cap (ab)^*\}$. So we can conclude that $\pi(\alpha) \neq \partial^+(\alpha)$.

Although, for $(b+ab+aab+abab)\cap(ab)^*$, $\pi(\alpha) \neq \partial^+(\alpha)$, there are regular expressions for which that equality holds, for example, for the regular expression a^*. In this way, we are able to conclude that $\partial^+(\alpha) \subseteq \pi(\alpha)$, for every $\alpha \in \mathsf{RE}_\cap$, but the inverse is not true.

For regular expressions $\alpha, \beta \in \mathsf{RE}_\cap$, the following proposition presents a sufficient and necessary condition for the equality of $\pi(\alpha \cap \beta)$ and $\partial^+(\alpha \cap \beta)$.

Proposition 4.22. *Given $\alpha, \beta \in RE_\cap$, $\pi(\alpha \cap \beta) = \partial^+(\alpha \cap \beta)$ if and only if $\pi(\alpha) = \partial^+(\alpha)$, $\pi(\beta) = \partial^+(\beta)$ and $\partial^+(\alpha \cap \beta) = \partial^+(\alpha) \cap \partial^+(\beta)$.*

Proof. ($\Rightarrow$) Let $\alpha, \beta \in \mathsf{RE}_\cap$. First, from the equation (4.10), note that $\partial^+(\alpha \cap \beta) \subseteq \partial^+(\alpha) \cap \partial^+(\beta)$ and thus $|\partial^+(\alpha \cap \beta)| \leq |\partial^+(\alpha)| \times |\partial^+(\beta)|$. Moreover, from proposition 4.20, it follows that $\partial^+(\alpha) \subseteq \pi(\alpha)$ and $\partial^+(\beta) \subseteq \pi(\beta)$. Now, suppose by contradiction that $\partial^+(\alpha) \subset \pi(\alpha)$ or $\partial^+(\beta) \subset \pi(\beta)$. Then $|\partial^+(\alpha)| < |\pi(\alpha)|$ or $|\partial^+(\beta)| < |\pi(\beta)|$ and we have the following:

$$\begin{aligned} |\partial^+(\alpha \cap \beta)| &\leq |\partial^+(\alpha)| \times |\partial^+(\beta)| \\ &< |\pi(\alpha)| \times |\pi(\beta)| \\ &= |\pi(\alpha) \cap \pi(\beta)| = |\pi(\alpha \cap \beta)|. \end{aligned}$$

So $|\partial^+(\alpha \cap \beta)| < |\pi(\alpha \cap \beta)|$ and then $\partial^+(\alpha \cap \beta) \subset \pi(\alpha \cap \beta)$, which is a contradiction since $\pi(\alpha \cap \beta) = \partial^+(\alpha \cap \beta)$. Thus, $\pi(\alpha) = \partial^+(\alpha)$ and $\pi(\beta) = \partial^+(\beta)$. Consequently, we conclude that $\partial^+(\alpha \cap \beta) = \pi(\alpha \cap \beta) = \partial^+(\alpha) \cap \partial^+(\beta)$.

($\Leftarrow$) This follows trivially from the definition of support, i.e., $\pi(\alpha \cap \beta) = \pi(\alpha) \cap \pi(\beta)$, since $\pi(\alpha) = \partial^+(\alpha)$ and $\pi(\beta) = \partial^+(\beta)$. □

The lemma bellow is a useful tool when proving that $\partial^+(\alpha \cap \beta) = \partial^+(\alpha) \cap \partial^+(\beta)$, where $\alpha, \beta \in \mathsf{RE}_\cap$,

Lemma 4.23. *Given $\alpha, \beta \in RE_\cap$, such that $\partial_w(\alpha) = \pi(\alpha)$ or $\partial_w(\beta) = \pi(\beta)$ holds for all $w \in \Sigma^+$, then $\partial^+(\alpha \cap \beta) = \partial^+(\alpha) \cap \partial^+(\beta)$.*

Proof. First, notice that if $\gamma \in \mathsf{RE}_\cap$ and $\partial_w(\gamma) = \pi(\gamma)$ for every $w \in \Sigma^+$, then $\partial^+(\gamma) = \bigcup_{w \in \Sigma^+} \partial_w(\gamma) = \pi(\gamma)$.

Given $\alpha, \beta \in \mathsf{RE}_\cap$, there are three possible cases to prove. First, suppose that, for all $w \in \Sigma^+$, we have $\partial_w(\alpha) = \pi(\alpha)$ and $\partial_w(\beta) = \pi(\beta)$. Then

$$\begin{aligned} \partial^+(\alpha \cap \beta) &= \bigcup_{w \in \Sigma^+} (\partial_w(\alpha) \cap \partial_w(\beta)) \\ &= \pi(\alpha) \cap \pi(\beta) = \partial^+(\alpha) \cap \partial^+(\beta). \end{aligned}$$

It remains to prove the cases that either $\partial_w(\alpha) = \pi(\alpha)$ or $\partial_w(\beta) = \pi(\beta)$, for all $w \in \Sigma^+$. The proof is the same for both cases. We only present the proof for the first case.

Suppose that, for all $w \in \Sigma^+$, $\partial_w(\alpha) = \pi(\alpha)$, it holds that

$$\begin{aligned}
\partial^+(\alpha \cap \beta) &= \bigcup_{w \in \Sigma^+} (\partial_w(\alpha) \curlywedge \partial_w(\beta)) \\
&= \bigcup_{w \in \Sigma^+} (\pi(\alpha) \curlywedge \partial_w(\beta)) \\
&= \bigcup_{w \in \Sigma^+} \{\alpha_i \cap \beta_j \mid \alpha_i \in \pi(\alpha),\ \beta_j \in \partial_w(\beta)\} \\
&= \left\{ \alpha_i \cap \beta_j \;\middle|\; \alpha_i \in \pi(\alpha),\ \beta_j \in \bigcup_{w \in \Sigma^+} \partial_w(\beta) \right\} \\
&= \{\alpha_i \cap \beta_j \mid \alpha_i \in \pi(\alpha),\ \beta_j \in \partial^+(\beta)\} \\
&= \pi(\alpha) \curlywedge \partial^+(\beta) \\
&= \partial^+(\alpha) \curlywedge \partial^+(\beta).
\end{aligned}$$

□

For every regular expression $\alpha \in \mathsf{RE}_\cap$, Proposition 4.20 showed that the set $\partial^+(\alpha)$ is a subset of $\pi(\alpha)$. In this way, the upper bound on the cardinality of $\partial^+(\alpha)$ is at most the upper bound of $|\pi(\alpha)|$, given by Proposition 4.18, i.e., $|\partial^+(\alpha)| \leq 2^{|\alpha|_\Sigma - |\alpha|_\cap - 1}$. This upper bound can be reached and the following proposition proves it.

Proposition 4.24. *For any $n \in \mathbb{N}$ there exists a regular expression $r_n \in \mathsf{RE}_\cap$ of size $O(n)$ such that $|\partial^+(r_n)| = 2^{|r_n|_\Sigma - |r_n|_\cap - 1}$.*

Proof. Let the regular expression $r_n \in \mathsf{RE}_\cap$ be defined inductively by the following:

$$\begin{aligned}
r_0 &= a^*a, \\
r_n &= r_{n-1} \cap a^*a.
\end{aligned}$$

The alphabetic length of r_n is given by $|r_n|_\Sigma = 2n + 2$ and thus $r_n = O(n)$. The number of occurrences of the operator $\cap$ is given by $|r_n|_\cap = n$. The cardinality of the support of r_n is given by $|\pi(r_n)| = 2^{2n+2-n-1} = 2^{|r_n|_\Sigma - |r_n|_\cap - 1}$ (see Example 4.19).

Now, we will prove that $\pi(r_n) = \partial^+(r_n)$. The proof proceed by induction on n. For $n = 0$, $\partial^+(a^*a) = \{a^*a, \varepsilon\} = \pi(a^*a)$. Let us assume by induction hypothesis that $\pi(r_n) = \partial^+(r_n)$, for $n \geq 1$. Then we want to show that it holds for r_{n+1}. First, note that $r_{n+1} = r_n \cap a^*a$. Since, for all $w \in \Sigma^*$, $\partial_w(a^*a) = \pi(a^*a)$, from Lemma 4.23, we can conclude that $\partial^+(r_n \cap a^*a) = \partial^+(r_n) \curlywedge \partial^+(a^*a)$. Being $\pi(a^*a) = \partial^+(a^*a)$, $\pi(r_n) =$

$\partial^+(r_n)$, by induction hypothesis, and $\partial^+(r_n \cap a^*a) = \partial^+(r_n) \frown \partial^+(r_n)$, therefore, from Proposition 4.22, $\pi(r_n \cap a^*a) = \partial^+(r_n \cap a^*a)$ or, equivalently, $\pi(r_{n+1}) = \partial^+(r_{n+1})$. □

4.6 Regular Expressions in $\mathsf{RE}_\cap$ to Finite Automata

As presented in the section 3.6, the conversion of a simple regular expression into an nondeterministic finite automata can be done efficiently. For regular expressions with intersection, however, Gelade [15] gives a $2^{\Omega(n)}$ lower bound to the conversion into an NFA and a $2^{2^{\Omega(n)}}$ lower bound to the conversion into a DFA, with respect to the number of states.

In the same way as for the simple regular expression, the methods of conversion of regular expressions with intersection into finite automata can also be divided in two classes: the methods that the converted automaton allows ε-transitions and the methods that the automaton does not allow. In this section, we present an extended version of the Thompson's automaton. The Brzozowski's automaton and the partial derivative's automaton are defined in the same way as for simple regular expression. Remember that the Brzozowski's construction is only finite under ACI_+-dissimilarity and thus it may not terminate for extended regular expressions.

The Glushkov's automaton, in constrast, does not extend to regular expression with intersection, since the operator $\cap$ is not compatible with notion of position. Let us take as example the regular expression $\alpha \equiv (ab^*) \cap a$. The marked version of α is $\widetilde{\alpha} \equiv (a_1 b_2^*) \cap a_3$. Although $(ab^*) \cap a = a$, the marked version gets $(a_1 b_2^*) \cap a_3 = \emptyset$. So, the language denoted by α is $\mathcal{L}(\alpha) = \{a\}$ and the language denoted by $\widetilde{\alpha}$ is $\mathcal{L}(\widetilde{\alpha}) = \emptyset$. And an automaton that accepts the language denoted by $\widetilde{\alpha}$ does not accept the language of α. Thereby, the Glushkov's construction is not compatible with the intersection.

4.6.1 Extended Thompson's Automata

The Thompson's construction transforms a given regular expression into an equivalent ε-NFA. Although Thompson does not present an extension of the construction for the intersection, it is possible to extend it by using the product of automata.

Given $\alpha_1 \in \mathsf{RE}$ and $\alpha_2 \in \mathsf{RE}$ and its corresponding ε-NFA, $\mathcal{N}_{\varepsilon_1}$ and $\mathcal{N}_{\varepsilon_2}$. The language denoted by the product of $\mathcal{N}_{\varepsilon_1}$ and $\mathcal{N}_{\varepsilon_2}$ is the same as the language denoted by the intersection of α_1 and α_2, i.e., $\mathcal{L}(\mathcal{N}_{\varepsilon_1} \times \mathcal{N}_{\varepsilon_2}) = \mathcal{L}(\alpha_1 \cap \alpha_2)$. And the automaton resulting from the product of $\mathcal{N}_{\varepsilon_1}$ and $\mathcal{N}_{\varepsilon_2}$ is also an ε-NFA.

Definition 4.25. *(**Extended Thompson's Automaton**) Let α be a regular expression in $\mathsf{RE}_\cap$. The ε-NFA $\mathcal{N}_\varepsilon$ is the extended Thompson's automaton and it is constructed recursively on the structure of α. All inductive cases, excepting α is $\alpha_1 \cap \alpha_2$, are defined in Definition 3.16. So here we only define the undefined case.*

Let $\alpha_1, \alpha_2 \in \mathsf{RE}_\cap$ be regular expressions over Σ, the $\mathcal{N}_{\varepsilon_1} = \langle Q_1, \Sigma, q_1, \delta_1, F_1 \rangle$ and $\mathcal{N}_{\varepsilon_2} = \langle Q_2, \Sigma, q_2, \delta_2, F_2 \rangle$ are the Thompson's automata for α_1 and α_2. If α is $\alpha_1 \cap \alpha_2$, then $\mathcal{N}_\varepsilon = \mathcal{N}_{\varepsilon_1} \times \mathcal{N}_{\varepsilon_2}$ and $\mathcal{L}(\mathcal{N}_\varepsilon) = \mathcal{L}(\mathcal{N}_{\varepsilon_1}) \cap \mathcal{L}(\mathcal{N}_{\varepsilon_2})$. That is $\mathcal{N}_\varepsilon = \langle Q, \Sigma, q, \delta, F \rangle$, where the set of states is $Q = \{(q_i, q_j) \mid q_i \in Q_1 \text{ and } q_j \in Q_2\}$, the initial state is $q = (q_1, q_2)$, the set of final states is $F = \{(q_i, q_j) \mid q_i \in F_1 \text{ and } q_j \in Q_2\}$ and the transition function is given by

$$\begin{aligned}
\delta((q_i, q_j), \varepsilon) = &\{(q_i', q_j') \mid q_i' \in \delta(q_i, \varepsilon) \text{ and } q_j' \in \delta(q_j, \varepsilon)\} \\
&\cup \{(q_i', q_j) \mid q_i' \in \delta(q_i, \varepsilon)\} \\
&\cup \{(q_i, q_j') \mid q_j' \in \delta(q_j, \varepsilon)\}, \\
\delta((q_i, q_j), a) = &\{(q_i', q_j') \mid q_i' \in \delta(q_i, a) \text{ and } q_j' \in \delta(q_j, a)\},
\end{aligned}$$

for all $a \in \Sigma$, $q_i \in Q_1$ and $q_j \in Q_2$.

The number of states of the automaton resulting from this construction for a regular expression with intersection is exponential with respect to the size of the regular expression.

Given the regular expression $\alpha = (a+b) \cap a$, the diagram of the Thompson's automaton for α is shown in Figure 4.1.

4.6.2 Partial Derivative's Automata

As previously mentioned, the partial derivative's automaton is defined in the same way as for simple regular expressions. For a regular expression with intersection α, the partial derivative's automaton has at most $2^{|\alpha|_\Sigma - |\alpha|_\cap - 1} + 1$ states (cf. Proposition 4.24).

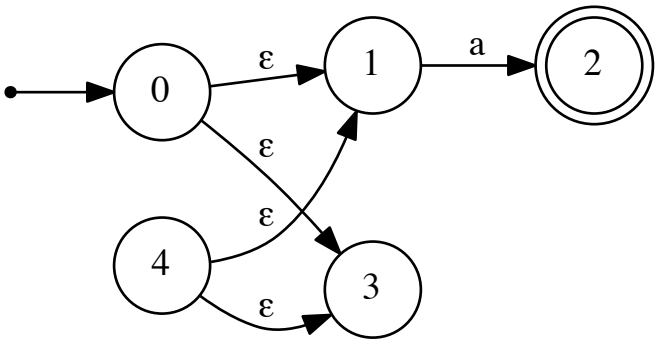


Figure 4.1: Diagram of the Thompson's automaton for $(a + b) \cap a$.

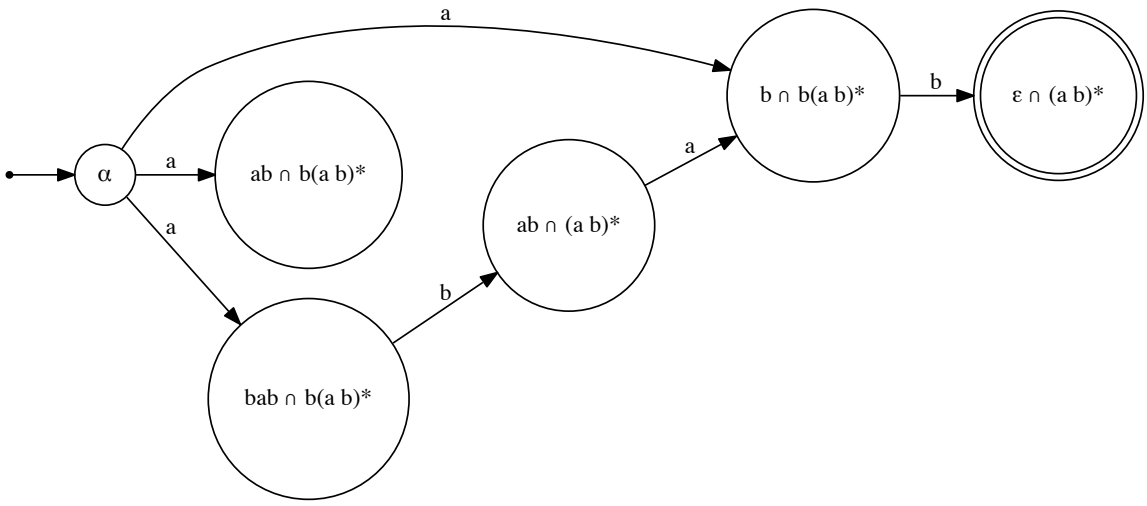


Figure 4.2: Diagram of the partial derivative's automaton for $(b+ab+aab+abab)\cap(ab)^*$.

Example 4.26. *Let α be the regular expression $(b+ab+aab+abab)\cap(ab)^*$ over $\Sigma = \{a, b\}$. The partial derivative's automaton of α is the automaton $\mathcal{N}_{pd} = \langle Q, \Sigma, q_0, \delta, F\rangle$, where the set of states is $Q = \partial^+(\alpha) = \{bab \cap b(ab)^*, ab \cap b(ab)^*, b \cap b(ab)^*, ab \cap (ab)^*, \varepsilon \cap (ab)^*\}$, as computed in Example 4.21, the initial state $q_0 = \alpha$, the set of final states is $F = \{\varepsilon \cap (ab)^*\}$ and the transition function is*

$\delta(\alpha, a) = \{b \cap b(ab)^*, ab \cap b(ab)^*, bab \cap b(ab)^*\}$, $\quad \delta(\alpha, b) = \emptyset$,

$\delta(b \cap b(ab)^*, a) = \emptyset$, $\quad \delta(b \cap b(ab)^*, b) = \{\varepsilon \cap (ab)^*\}$,

$\delta(ab \cap b(ab)^*, a) = \emptyset$, $\quad \delta(ab \cap b(ab)^*, b) = \emptyset$,

$\delta(bab \cap b(ab)^*, a) = \emptyset$, $\quad \delta(bab \cap b(ab)^*, b) = \{ab \cap (ab)^*\}$,

$\delta(ab \cap (ab)^*, a) = \{b \cap b(ab)^*\}$, $\quad \delta(ab \cap (ab)^*, b) = \emptyset$.

The diagram of $\mathcal{N}_{pd}$ is depicted in Figure 4.2.

4.7 FAdo

To represent regular expressions with intersection, we define the class `conj` that inherits from `regexp`. The new class diagram of the module `reex` is depicted in Figure 4.3.

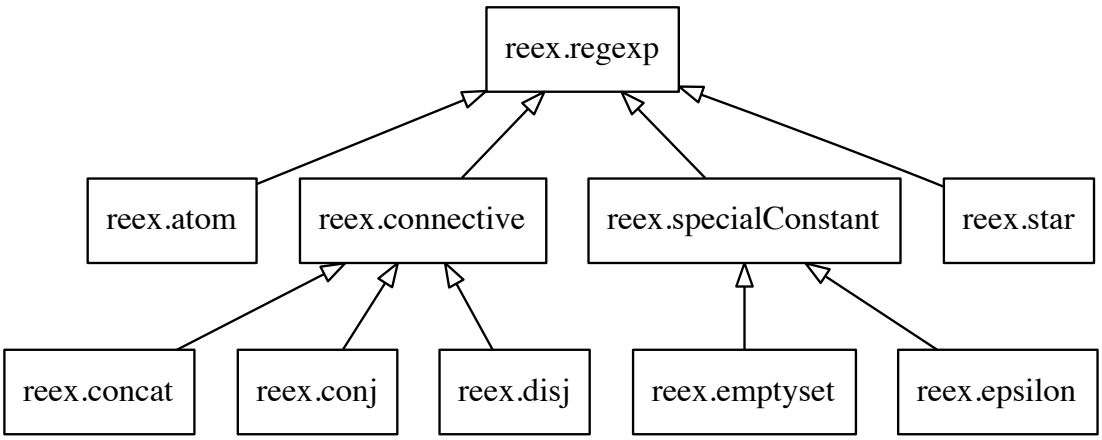


Figure 4.3: Class diagram of `reex` for regular expressions with intersection.

The function `reex.str2regexp(str)` can also be used to convert a string into a class `reex.str2regexp(conj)`. In FAdo the string & represents the intersection.

```
>>> from FAdo.reex import *
>>> re = str2regexp("a*a&a*&aa")
>>> re
conj(conj(concat(star(atom(a)),atom(a)),star(atom(a))),concat(
   atom(a),atom(a)))
>>> print re
((a*a)&a*)&(aa)
```

The measure methods of `regexp` were also implemented for the class `conj`. The method `regexp.conjLength()`, which returns the number of occurences of the operator intersection, is now available for all subclasses of `regexp`.

The existing methods to calculate the derivative, the set of partial derivatives, the linear form, the set of all partial derivatives and the support of a regular expression were extended for the class `conj`.

We can convert a regular expression with intersection into a finite automaton by using the method `regexp.nfaPD()`, building the partial derivative's automaton. The Gluskov's automaton, Thompson's automaton and Brzozowski's automaton are not available for regular expressions with intersection.

4.8 Experimental Results

In order to compare the number of states of the partial derivative's automaton and the support for regular expressions with intersections, we ran some experiments. Since the grammar for $\mathsf{RE}_\cap$ generates regular expressions denoting the empty language with high probability, for these experiments, we chose a simpler version of this grammar in prefix notation:

$$\alpha := a \in \Sigma \mid + \ \alpha \ \alpha \mid \cap \ \alpha \ \alpha \mid \cdot \ \alpha \ \alpha \mid * \ \alpha,$$

For the results to be statistically significant, regular expressions were uniformly random generated using the grammar above. For each size $n \in \{25, 50, 100, 150, 200\}$ and alphabet size $k \in \{1, 2, 5, 10\}$, samples of 10000 regular expressions were generated. Then, for each sample we calculated the average and the maximum value of several measures, which are presented in the table in Appendix A.

As we can observe, by the values in Table A.1, the set of partial derivatives is on average much smaller than the support, and than the worst-case. Moreover, it also suggests that the state complexity of partial derivative's automaton may even be polynomial for these expressions.

Chapter 5

Extended Regular Expressions

The regular languages, in addition to be closed under intersection, are also closed under complement. Thus, regular expressions can be enriched with this operation. We call *extended regular expressions* the result of extending $\mathsf{RE}_\cap$ to complement. The unary operator that represents the complement is $\neg$.

Definition 5.1. ***(Extended Regular Expressions)*** *Let Σ be an alphabet. An extended regular expression over Σ is given by the following grammar:*

$$\alpha := \emptyset \mid \varepsilon \mid \Sigma^+ \mid \Sigma^* \mid a \in \Sigma \mid (\alpha + \alpha) \mid (\alpha \cap \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha)^* \mid \neg(\alpha).$$

The language $\mathcal{L}(\alpha)$ associated with α is defined inductively as for $\mathsf{RE}_\cap$, adding the following cases: $\mathcal{L}(\Sigma^+) = \Sigma^* \setminus \{\varepsilon\}$, $\mathcal{L}(\Sigma^*) = \Sigma^*$ and $\mathcal{L}(\neg\alpha) = \Sigma^* \setminus \mathcal{L}(\alpha)$. The set of all extended regular expressions is denoted by $\mathsf{RE}_{\cap,\neg}$.

A regular expression in $\mathsf{RE}_{\cap,\neg} \setminus \{\Sigma^*, \Sigma^+\}$ is called a *regular expression with intersection and complement*. We assume that the operator $\neg$ has higher precedence than $*$. Therefore, we have the following precedences: $\neg > * > \cdot > \cap > +$.

Observe that $\Sigma^* = \neg\emptyset$ and $\Sigma^+ = \neg\varepsilon$. This follows from the fact that $\mathcal{L}(\neg\emptyset) = \Sigma^* \setminus \mathcal{L}(\emptyset) = \mathcal{L}(\Sigma^*)$ and $\mathcal{L}(\neg\varepsilon) = \Sigma^* \setminus \mathcal{L}(\varepsilon) = \mathcal{L}(\Sigma^+)$, respectively.

The definitions of equivalence, size, alphabetic size and empty word property (e.w.p.) are defined in the same way as for RE. The function $\varepsilon : \mathsf{RE}_{\cap,\neg} \to \{\varepsilon, \emptyset\}$ is defined recursively as for $\mathsf{RE}_\cap$, adding the following cases:

$$\varepsilon(\Sigma^*) = \varepsilon, \qquad \varepsilon(\Sigma^+) = \emptyset, \qquad \varepsilon(\neg\alpha) = \varepsilon(\neg\varepsilon(\alpha)).$$

5.1 Algebra of $\mathsf{RE}_{\cap,\neg}$

The set of regular expression $\mathsf{RE}_{\cap,\neg}$ over Σ forms an algebraic structure $(\mathsf{RE}_{\cap,\neg}, +, \cdot, \cap, *, \neg, \emptyset, \varepsilon)$, where $(\mathsf{RE}, +, \cdot, *, \emptyset, \varepsilon)$ is a Kleene algebra, and $\cap$ and $\neg$ are a binary and an unary operators, respectively, satisfying the axioms (A_{25}), (A_{26}) and (A_{27}), which are defined bellow.

Salomaa and Tixier [26] suggest a complete and sound axiomatization for the extended regular expressions - the system F. The axiomatic system F is an extension of the system F_1, presented in the section 3.1. The axioms in F are the axioms of F_1 (A_1)-(A_{11}) and the following three axioms:

$$\alpha \cap \beta = \neg(\neg\alpha + \neg\beta), \tag{A_{25}}$$

$$\neg(\alpha_1 a_1 + \ldots + \alpha_k a_k + \varepsilon) = (\neg\alpha_1)a_1 + \ldots + (\neg\alpha_k)a_k, \tag{A_{26}}$$

$$\neg(\alpha_1 a_1 + \ldots + \alpha_n a_n) = (\neg\alpha_1)a_1 + \ldots + (\neg\alpha_n)a_n + \varepsilon, \tag{A_{27}}$$

for all $\alpha, \alpha_i, \beta \in \mathsf{RE}_{\cap,\neg}$ and all $a_i \in \Sigma$, $i = 1, \ldots, n$. The rules of inference of F are the same as those for F_1 (see page 17).

The symbol Σ^* is the identity element w.r.t. $\cap$ and the zero element w.r.t. $+$ on extended regular expressions. This is due to the fact that, for every extended regular expression α over Σ, $\mathcal{L}(\alpha) \subseteq \Sigma^*$ and thus

$$\mathcal{L}(\alpha \cap \Sigma^*) = \mathcal{L}(\alpha) \cap \mathcal{L}(\Sigma^*) = \mathcal{L}(\alpha) \cap \Sigma^* = \mathcal{L}(\alpha),$$
$$\mathcal{L}(\alpha + \Sigma^*) = \mathcal{L}(\alpha) \cup \mathcal{L}(\Sigma^*) = \mathcal{L}(\alpha) \cup \Sigma^* = \Sigma^* = \mathcal{L}(\Sigma^*).$$

By the commutativity of $\cap$ and $+$, $\mathcal{L}(\Sigma^* \cap \alpha) = \mathcal{L}(\alpha \cap \Sigma^*) = \mathcal{L}(\alpha)$ and $\mathcal{L}(\Sigma^* + \alpha) = \mathcal{L}(\alpha + \Sigma^*) = \mathcal{L}(\Sigma^*)$.

Observe that, from (A_{26}) and (A_{27}), for every $\alpha_i \in \mathsf{RE}_{\cap,\neg}$, we also have:

$$\neg(a_1\alpha_1 + \ldots + a_n\alpha_n + \varepsilon) = a_1(\neg\alpha_1) + \ldots + a_n(\neg\alpha_n), \tag{5.1}$$

$$\neg(a_1\alpha_1 + \ldots + a_n\alpha_n) = a_1(\neg\alpha_1) + \ldots + a_n(\neg\alpha_n) + \varepsilon. \tag{5.2}$$

As we have previously seen, given a regular expression α over Σ, $\alpha = a_1\alpha_1 + \cdots + a_k\alpha_k + \varepsilon(\alpha)$, where $a_i \in \Sigma$ and $\alpha_i \in \mathsf{RE}_{\cap,\neg}$. Moreover, from the definition of the function ε and from the equations (5.1) and (5.2), we obtain the following equation:

$$\neg\alpha = a_1\neg\alpha_1 + \cdots a_k\neg\alpha_k + \varepsilon(\neg\alpha). \tag{5.3}$$

5.2 Derivatives

The recursive definition of derivative was also extended to the complement of a regular expression [10]. Here we present this extension and the derivative of the regular expressions Σ^* and Σ^+.

Definition 5.2. ***(Derivative)*** *Let α be an extended regular expression over Σ. The derivative of α w.r.t. a symbol $a \in \Sigma$, written $d_a(\alpha)$, is defined recursively as for $\mathsf{RE}_\cap$, adding the following inductive cases:*

$$d_a(\Sigma^*) = \Sigma^*, \qquad d_a(\Sigma^+) = \Sigma^*, \qquad d_a(\neg\alpha) = \neg d_a(\alpha).$$

We want to prove that for all $w \in \Sigma^*$ and all $\alpha \in \mathsf{RE}_{\cap,\neg}$, $\mathcal{L}(d_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$. Brzozowski [10, Theorem 3.2] proved that this holds for every regular expression with intersection and complement. Since the proof only depends on $\mathcal{L}(d_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$, we just need to prove this for Σ^* and Σ^+. As previously seen, $\Sigma^* = \neg\emptyset$ and $\Sigma^+ = \neg\varepsilon$, and thus, for all $a \in \Sigma$, $d_a(\Sigma^*) = d_a(\neg\emptyset) = \neg d_a(\emptyset) = \neg\emptyset = \Sigma^*$. The proof proceeds in the same way for Σ^+. Therefore, $\mathcal{L}(d_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$, for all $\alpha \in \mathsf{RE}_{\cap,\neg}$ and $w \in \Sigma^*$.

Moreover, as was also proved by Brzozowski, every regular expression $\alpha \in \mathsf{RE}_{\cap,\neg}$ over Σ is equivalent to $\sum_{a\in\Sigma} a d_a(\alpha) + \varepsilon(\alpha)$, cf. equation (3.1), and has only a finite number of ACI_+-dissimilar derivatives.

5.3 Partial Derivatives

In this section, the notion of partial derivative defined for $\mathsf{RE}_{\cap,\neg}$ is generalized to extended regular expressions. We will present two different definitions for this. First, we define a naive extension and then we improve it.

Let the operation $\neg : 2^{\mathsf{RE}_{\cap,\neg}} \to 2^{\mathsf{RE}_{\cap,\neg}}$ be an extension of the complement for sets of regular expressions. Given $\alpha, \alpha_1, \ldots, \alpha_n \in \mathsf{RE}_{\cap,\neg}$, the operation is defined recursively as follows:

$$\neg\emptyset = \{\Sigma^*\}, \tag{5.4}$$

$$\neg\{\alpha\} = \{\neg\alpha\}, \tag{5.5}$$

$$\neg\{\alpha_1, \ldots, \alpha_n\} = \{\neg\alpha_1 \cap \cdots \cap \neg\alpha_n\}. \tag{5.6}$$

Lemma 5.3. *For every set* $S \subseteq \mathsf{RE}_{\cap,\neg}$, $\mathcal{L}(\neg S) = \Sigma^* \backslash \mathcal{L}(S)$.

Proof. Let $\alpha, \alpha_1, \ldots, \alpha_n$ be extended regular expressions and S be a set of extended regular expressions. We will make use of the fact that $\mathcal{L}(S) = \bigcup_{\alpha_i \in S} \mathcal{L}(\alpha_i)$ and $\mathcal{L}(\neg\alpha) = \Sigma^* \backslash \mathcal{L}(\alpha)$. The proof of each inductive case is given, respectively, by the following three proofs.

Case $S = \emptyset$:

$$\mathcal{L}(\neg\emptyset) = \mathcal{L}(\{\Sigma^*\}) = \Sigma^* = \Sigma^* \backslash \mathcal{L}(\emptyset).$$

Case $S = \{\alpha\}$:

$$\mathcal{L}(\neg\{\alpha\}) = \mathcal{L}(\{\neg\alpha\}) = \Sigma^* \backslash \mathcal{L}(\alpha).$$

Case $S = \{\alpha_1, \ldots, \alpha_n\}$:

$$\begin{aligned}\mathcal{L}(\neg\{\alpha_1, \ldots, \alpha_n\}) &= \mathcal{L}(\{\neg\alpha_1 \cap \cdots \cap \neg\alpha_n\}) = \mathcal{L}(\neg\alpha_1 \cap \cdots \cap \neg\alpha_n) \\ &= \mathcal{L}(\neg(\alpha_1 + \cdots + \ \alpha_n)) = \Sigma^* \backslash \mathcal{L}(\alpha_1 + \cdots + \alpha_n) \\ &= \Sigma^* \backslash \mathcal{L}(\{\alpha_1, \ldots, \alpha_n\}).\end{aligned}$$

□

5.3.1 Natural Extension

Caron et al. [11] proposed a natural extension of partial derivatives using the derivative of $\neg\alpha$, i.e., $\partial_a(\neg\alpha) = \{d_a(\neg\alpha)\}$. With this extension, there is only one regular expression belonging to the set of partial derivatives of $\neg\alpha$. Here, we present an extension, also natural, by using the operator $\neg : 2^{\mathsf{RE}_{\cap,\neg}} \to 2^{\mathsf{RE}_{\cap,\neg}}$. This definition of partial derivatives will be called *natural extension.*

Definition 5.4. ***(Partial Derivative)*** *Let α be an extended regular expression over Σ. The set of partial derivatives of α w.r.t. a symbol $a \in \Sigma$, written $\partial_a(\alpha)$, is defined recursively as for $\mathsf{RE}_\cap$, adding the following inductive cases:*

$$\partial_a(\Sigma^*) = \{\Sigma^*\}, \qquad \partial_a(\Sigma^+) = \{\Sigma^*\}, \qquad \partial_a(\neg\alpha) = \neg\partial_a(\alpha).$$

In the same way as the natural extension of Caron et al., the set $\neg\partial_a(\alpha)$ is a singleton set. The definition of set of partial derivatives of an extended regular expression is extended to words, to sets of words and to sets of regular expressions in the same way as for simple regular expressions.

The following facts relate the natural extension of partial derivatives and the left quotient of a language.

Lemma 5.5. *Given $\alpha \in \mathsf{RE}_{\cap,\neg}$ over Σ and $a \in \Sigma$, $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$.*

Proof. Let γ be an extended regular expression over Σ and $a \in \Sigma$. The proof proceeds by induction on the structure of γ. The proof for all cases, excluding Σ^+, Σ^* and $\gamma \equiv \neg\alpha$, has already been given (see Lemma 4.7 and [2]). Therefore, here we only present the proof for the remaining cases. First, recall that $\Sigma^* = \neg\emptyset$ and $\Sigma^+ = \neg\varepsilon$. If $\alpha \equiv \varepsilon$ or $\alpha \equiv \emptyset$, then the following holds:

$$\begin{aligned} a^{-1}\mathcal{L}(\neg\alpha) &= \Sigma^*\backslash a^{-1}\mathcal{L}(\alpha) = \Sigma^*\backslash\emptyset \\ &= \mathcal{L}(\Sigma^*) = \mathcal{L}(\{\Sigma^*\}) = \mathcal{L}(\partial_a(\neg\alpha)). \end{aligned}$$

Let α be an extended regular expression. Suppose by inductive hypothesis that $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$. So, if $\gamma \equiv \neg\alpha$, from Lemma 5.3, we have:

$$\begin{aligned} \mathcal{L}(\partial_a(\neg\alpha)) &= \mathcal{L}(\neg\partial_a(\alpha)) = \Sigma^*\backslash\mathcal{L}(\partial_a(\alpha)) \\ &= \Sigma^*\backslash a^{-1}\mathcal{L}(\alpha) = a^{-1}\mathcal{L}(\neg\alpha). \end{aligned}$$

□

Proposition 5.6. *Given $\alpha \in \mathsf{RE}_{\cap,\neg}$ over Σ and $w \in \Sigma^*$, $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$.*

Proof. The proof follows in the same way as the proof for the case that $\alpha \in \mathsf{RE}$ (see Proposition 3.8). □

Although, for regular expressions with intersection, the set of all partial derivatives is finite, for extended regular expressions this cannot be guaranteed. The next example is a witness that the set ∂^+ can be an infinite set for extended regular expressions.

Example 5.7. *For the regular expression $\alpha = \neg(a^*a)$ over $\Sigma = \{a\}$, we have the following sets of partial derivatives of α:*

$$\begin{aligned}
\partial_a(\neg(a^*a)) &= \neg\partial_a(a^*a) = \neg\{a^*a, \varepsilon\} = \{\neg(a^*a) \cap \neg\varepsilon\} \\
&= \{\neg(a^*a) \cap \Sigma^+\}, \\
\partial_{aa}(\neg(a^*a)) &= \partial_a(\neg(a^*a) \cap \Sigma^+) = \partial_a(\neg(a^*a)) \cap \partial_a(\Sigma^+) \\
&= \{\neg(a^*a) \cap \Sigma^+\} \cap \{\Sigma^*\} = \{\neg(a^*a) \cap \Sigma^+ \cap \Sigma^*\}, \\
\partial_{aaa}(\neg(a^*a)) &= \partial_a(\neg(a^*a) \cap \Sigma^+ \cap \Sigma^*) = \partial_a(\neg(a^*a)) \cap \partial_a(\Sigma^+) \cap \partial_a(\Sigma^*) \\
&= \{\neg(a^*a) \cap \Sigma^+\} \cap \{\Sigma^*\} \cap \{\Sigma^*\} = \{\neg(a^*a) \cap \Sigma^+ \cap \Sigma^* \cap \Sigma^*\}, \\
&\cdots \\
\partial_{\underbrace{a\cdots a}_{n}}(\neg(a^*a)) &= \{\neg(a^*a) \cap \Sigma^+ \cap \underbrace{\Sigma^* \cap \cdots \cap \Sigma^*}_{n-1}\}.
\end{aligned}$$

Since, for every $w \in \Sigma^*$, the size of the regular expression belonging to $\partial_w(\alpha)$ increases with the length of w, $\partial^+(\alpha) = \bigcup_{w\in\Sigma^+} \partial_w$ is an infinite set.

Remark 5.8. *Let α, β be an extended regular expressions. The derivative of $\alpha\beta$ w.r.t. $a_1 \cdots a_r \in \Sigma^*$ can has the form[1]:*

$$\begin{aligned}
d_{a_1\cdots a_r}(\alpha\beta) = d_{a_1\cdots a_r}(\alpha)\beta &+ \varepsilon(d_{a_1\cdots a_{r-1}}(\alpha))d_{a_r}(\beta) + \cdots \\
&+ \varepsilon(d_{a_1}(\alpha))d_{a_2\cdots a_r}(\beta) + \varepsilon(\alpha)d_{a_1\cdots a_r}(\beta).
\end{aligned}$$

And thus the number of terms in $d_w(\alpha\beta)$ can increase with the length of $w \in \Sigma^*$. The same problem can happen with regular expressions with star as outer most operator, since the derivative w.r.t. a symbol is a concatenation. The operators disjunction, intersection and complement do not lead to this problem. As previously seen, if the size of $d_w(\alpha)$ increases indefinitely, the set of all derivatives is an infinite set. Brzozowski [10, Theorem 5.2] proved that every extended regular expression has a finite set of ACI_+-dissimilar derivatives. As such, we use the ACI_+ axioms to identify syntactically identical terms in d_w and to express d_w with the previous derivatives.

If, instead of $d_a(\neg(\alpha\beta)) = \neg(d_a(\alpha)\beta + \varepsilon(\alpha)(d_a(\beta))$, it was $d_a(\neg(\alpha\beta)) = \neg(d_a(\alpha)\beta) \cap \neg(d_a(\beta))$, $d_w(\neg(\alpha\beta))$ it would have the form:

$$d_{a_1\cdots a_r}(\neg(\alpha\beta)) = \neg(d_{a_1\cdots a_r}(\alpha)\beta) \cap \neg(\varepsilon(d_{a_1\cdots a_{r-1}}(\alpha))d_{a_r}(\beta)) \cap \cdots$$

[1]The function $\varepsilon : \mathsf{RE}_{\cap,\neg} \to \{\emptyset, \varepsilon\}$ is used to show the form of $d_w(\alpha\beta)$, although it is not in the definition of derivative (cf. Definition 3.2).

$$\cap \neg(\varepsilon(d_{a_1}(\alpha))d_{a_2\cdots a_r}(\beta)) \cap \neg(\varepsilon(\alpha)d_{a_1\cdots a_r}(\beta)).$$

Then the number of terms $d_w(\neg(\alpha\beta))$ would increase with the length of w. In the other hand, $d_w(\neg(\alpha\beta))$ is an intersection of terms, instead of being a disjunction of terms. So, to identify syntactically identical terms in d_w we would also need the associativity, commutativity and idempotence of intersection ($ACI_\cap$).

With the set of partial derivatives (see Definition 3.6) and the associativity, commutativity and idempotence of the operator $\cup$ on sets, the finiteness problem of d_w for simple regular expressions was solved by Antimirov. The problem remained solved, when we extended the notion of partial derivatives to regular expressions with intersection.

However, when extending partial derivatives to extended regular expressions, the set properties are no longer sufficient (cf. Example 5.7). Suppose $\partial_a(\alpha) = \{\alpha_1, \ldots, \alpha_n\}$, $\partial_a(\beta) = \{\beta_1, \ldots, \beta_m\}$ and $\varepsilon(\alpha) = \varepsilon$, then $\partial_a(\neg(\alpha\beta)) = \{\neg(\alpha_1\beta) \cap \cdots \cap \neg(\alpha_n\beta) \cap \neg\beta_1 \cap \cdots \cap \neg\beta_m\}$. Note that this construction corresponds to the application of the De Morgan law to $\partial_a(\alpha\beta)$, as applied in $d_a(\neg(\alpha\beta)) = \neg(d_a(\alpha)\beta) \cap \neg(d_a(\beta))$. So, in the same way as for this derivatives, the ACI_+ axioms are not sufficient to ensure the finiteness of ∂_w. This is also verified when $\varepsilon(\alpha) = \emptyset$ and for the inductive case α^*.

Let us denote by $CII_\cap$ the set of axioms corresponding to the commutativity, idempotence and identity of $\cap$. We show that the set of all $CII_\cap$-dissimilar partial derivatives is finite. In order to prove this, we use the following lemma.

Lemma 5.9. *Given $\alpha \in RE_{\cap,\neg}$ over Σ and $w \in \Sigma^*$,*

$$\partial_w(\neg\alpha) =_{CII_\cap} \neg\partial_w(\alpha).$$

Proof. The proof proceeds by induction on the length of $w \in \Sigma^*$. If $w = \varepsilon$, the following holds:

$$\partial_\varepsilon(\neg\alpha) = \{\neg\alpha\} = \neg\{\alpha\} = \neg\partial_\varepsilon(\alpha).$$

Suppose by inductive hypothesis that $\partial_w(\neg\alpha) =_{CII_\cap} \neg\partial_w(\alpha)$ and $\partial_w(\alpha) = \{\alpha_1, \ldots, \alpha_n\}$. We want to prove that the claim also holds for wa, where $a \in \Sigma$. Note that

$$\neg\partial_{wa}(\alpha) = \neg(\partial_a(\partial_w(\alpha)) = \neg(\partial_a(\{\alpha_1, \ldots, \alpha_n\})) = \neg(\partial_a(\alpha_1) \cup \cdots \cup \partial_a(\alpha_n)),$$

And thus, from the commutativity and idempotence of $\cup$ on sets, the intersection belonging to $\neg\partial_{wa}(\alpha)$ has only syntactically different terms. Since $\partial_w(\neg\alpha) =_{CII_\cap} \neg\partial_w(\alpha) = \{\neg\alpha_1 \cap \cdots \cap \neg\alpha_n\}$,

$$\begin{aligned}\partial_{wa}(\neg\alpha) &= \partial_a(\partial_w(\neg\alpha)) = \partial_a(\neg\partial_w(\alpha)) \\ &= \partial_a(\{\neg\alpha_1 \cap \cdots \cap \neg\alpha_n\}) \\ &= \partial_a(\neg\alpha_1) \barwedge \cdots \barwedge \partial_a(\neg\alpha_n) \\ &= \neg\partial_a(\alpha_1) \barwedge \cdots \barwedge \neg\partial_a(\alpha_n).\end{aligned}$$

We know that, for each $i = 1, \ldots, n$, there is no syntactically identical terms in the intersection belonging to $\neg\partial_a(\alpha_i)$. However, it is not guaranteed that $\partial_a(\alpha_i) \cap \partial_a(\alpha_i') = \emptyset$, for all $\alpha_i, \alpha_i' \in \partial_a(\alpha)$ with $\alpha_i \neq \alpha_i'$. And thus, it is possible the existence of syntactically identical terms in the intersection in $\neg\partial_a(\alpha_1) \barwedge \cdots \barwedge \neg\partial_a(\alpha_n)$. Therefore, we need the commutativity and the idempotence of $\cap$ to identify syntactically identical terms. Moreover, if there is α_i such that $\partial_a(\alpha_i) = \emptyset$, then $\neg\partial_a(\alpha_i) = \{\Sigma^*\}$ and it is not necessary that Σ^* exists in $\partial_{wa}(\alpha)$. So, in that case, we also need the identity property of $\cap$. □

Proposition 5.10. *Given $\alpha \in RE$, $\partial^+(\neg\alpha) \subseteq_{CII_\cap} \{\neg S \mid S \in 2^{\partial^+(\alpha)}\}$.*

Proof. Let E be a set and α be an extended regular expression. If $\partial_w(\alpha) \subseteq E$, for all $w \in \Sigma^+$, then we have $\bigcup_{w\in\Sigma^+} \partial_w(\alpha) \subseteq E$ and thus $\partial^+(\alpha) \subseteq E$. Moreover, since $\partial^+(\alpha) =_{CII_\cap} \bigcup_{w\in\Sigma^+} \partial_w(\alpha)$, we know that for every $w \in \Sigma^+$, $\partial_w(\alpha) \subseteq_{CII_\cap} \partial^+(\alpha)$. Let $\partial_w(\alpha) = \{\alpha_1, \ldots, \alpha_n\}$. Then, the following holds:

$$\begin{aligned}\partial_w(\neg\alpha) &=_{CII_\cap} \neg\partial_w(\alpha) =_{CII_\cap} \neg\{\alpha_1, \ldots, \alpha_n\} \\ &=_{CII_\cap} \{\neg\alpha_1 \cap \cdots \cap \neg\alpha_n\} \subseteq_{CII_\cap} \{\neg S \mid S \in 2^{\partial^+(\alpha)}\}.\end{aligned}$$

And thus, we can conclude that $\partial^+(\neg\alpha) \subseteq_{CII_\cap} \{\neg S \mid S \in 2^{\partial^+(\alpha)}\}$. □

5.3.2 Partial Derivatives $\overline{\partial}$

Now, we present an alternative extension of the notion of partial derivatives to extended regular expressions. In contrast with the previous extension, this new recursive definition of partial derivatives has the advantage of to make nonderterminism of $\partial_a(\neg\alpha)$ explicit.

Definition 5.11. ***(Partial Derivatives)*** *Let $\alpha \in \mathsf{RE}_{\cap,\neg}$ be a regular expression over Σ. The set of partial derivatives of α w.r.t. a symbol $a \in \Sigma$, written $\partial_a(\alpha)$, is defined recursively as for $\mathsf{RE}_{\cap}$, adding the inductive cases:*

$$\partial_a(\Sigma^*) = \{\Sigma^*\}, \qquad \partial_a(\Sigma^+) = \{\Sigma^*\}, \qquad \partial_a(\neg\alpha) = \overline{\partial}_a(\alpha).$$

The set $\overline{\partial}_a(\alpha)$ is defined inductively as follows:

$$\begin{array}{ll}
\overline{\partial}_a(\Sigma^*) = \emptyset, & \overline{\partial}_a(\alpha + \beta) = \overline{\partial}_a(\alpha) \owedge \overline{\partial}_a(\beta), \\
\overline{\partial}_a(\Sigma^+) = \emptyset, & \overline{\partial}_a(\alpha \cap \beta) = \overline{\partial}_a(\alpha) \cup \overline{\partial}_a(\beta), \\
\overline{\partial}_a(\emptyset) = \{\Sigma^*\}, & \overline{\partial}_a(\alpha\beta) = \neg(\partial_a(\alpha)\beta) \owedge \overline{\partial}_a(\beta), \ \ \textit{if } \varepsilon(\alpha) = \varepsilon, \\
\overline{\partial}_a(\varepsilon) = \{\Sigma^*\}, & \overline{\partial}_a(\alpha\beta) = \neg(\partial_a(\alpha)\beta), \ \ \textit{if } \varepsilon(\alpha) = \emptyset, \\
\overline{\partial}_a(a) = \{\Sigma^+\}, & \overline{\partial}_a(\alpha^*) = \neg(\partial_a(\alpha^*)), \\
\overline{\partial}_a(b) = \{\Sigma^*\}, \ \ b \in \Sigma \textit{ and } b \neq a, & \overline{\partial}_a(\neg\alpha) = \partial_a(\alpha).
\end{array}$$

While, for every $\alpha \in \mathsf{RE}_{\cap,\neg}$, $\neg\partial_a(\alpha)$ is always a singleton set, it does not holds for $\overline{\partial}_a(\alpha)$. This is demonstrated in the next example.

Example 5.12. *Let $\alpha \equiv (aa + a) \cap a^* \cap \neg(a^*a)$. The set of partial derivatives of $\neg\alpha$ w.r.t. a given by the natural extensions is*

$$\begin{aligned}
\partial_a(\neg\alpha) &= \neg\partial_a((aa + a) \cap a^* \cap \neg(a^*a)) \\
&= \neg(\partial_a(aa + a) \owedge \partial_a(a^*) \owedge \partial_a(\neg(a^*a))) \\
&= \neg(\{a, \varepsilon\} \owedge \{a^*\} \owedge \neg\{a^*a, \varepsilon\}) \\
&= \neg(\{a, \varepsilon\} \owedge \{a^*\} \owedge \{\neg(a^*a) \cap \neg\varepsilon\}) \\
&= \neg\{a \cap a^* \cap \neg(a^*a) \cap \neg\varepsilon, \varepsilon \cap a^* \cap \neg(a^*a) \cap \neg\varepsilon\} \\
&= \{\neg(a \cap a^* \cap \neg(a^*a) \cap \neg\varepsilon) \cap \neg(\varepsilon \cap a^* \cap \neg(a^*a) \cap \neg\varepsilon)\}.
\end{aligned}$$

However, if we use the new extension, we have $\partial_a(\neg\alpha) = \overline{\partial}_a(\alpha)$, which is given by

$$\begin{aligned}
\overline{\partial}_a(\alpha) &= \overline{\partial}_a((aa + a) \cap a^* \cap \neg(a^*a)) \\
&= \overline{\partial}_a(aa + a) \cup \overline{\partial}_a(a^*) \cup \overline{\partial}_a(\neg(a^*a)) \\
&= (\overline{\partial}_a(aa) \owedge \overline{\partial}_a(a)) \cup \overline{\partial}_a(a^*) \cup \partial_a(a^*a) \\
&= (\neg(\partial_a(a)a) \owedge \{\Sigma^+\}) \cup \neg(\partial_a(a)a^*) \cup (\partial_a(a^*)a \cup \partial_a(a)) \\
&= (\{\neg a\} \owedge \{\Sigma^+\}) \cup \{\neg a^*\} \cup \{a^*a, \varepsilon\}
\end{aligned}$$

$$= \{\neg a \cap \Sigma^+, \neg a^*, a^*a, \varepsilon\}.$$

The set $\overline{\partial}_a(\alpha)$ is not a singleton set. Moreover, the partial derivatives belonging to $\overline{\partial}_a(\alpha)$ have a smaller size than the partial derivative in $\neg\partial_a(\alpha)$.

The definition of the set of partial derivatives is extended to words, to sets of words and to sets of regular expressions in the same way as for simple regular expressions.

In order to prove that the language denoted by the set of partial derivatives of an extended regular expression is the same as the left-quotient of the language of the regular expression, i.e., $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$, for all $w \in \Sigma^*$, we first need to prove that $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$, for all $a \in \Sigma$. The next lemma is an useful tool to prove it.

Lemma 5.13. *Given $\alpha \in \mathsf{RE}_{\cap,\neg}$ over Σ and $a \in \Sigma$, the following holds:*

$$\mathcal{L}(\overline{\partial}_a(\alpha)) = \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha)).$$

Proof. Let $\gamma \in \mathsf{RE}_{\cap,\neg}$ over Σ. The proof proceed by induction on the structure of γ. We begin by proving the base cases. If $\gamma = \Sigma^*$ or $\gamma = \Sigma^+$, then

$$\mathcal{L}(\overline{\partial}_a(\gamma)) = \mathcal{L}(\emptyset) = \Sigma^* \backslash \Sigma^* = \Sigma^* \backslash \mathcal{L}(\{\Sigma^*\}) = \Sigma^* \backslash \mathcal{L}(\partial_a(\gamma)).$$

In the case that $\gamma = \emptyset$, $\gamma = \varepsilon$ or $\gamma = b$, where $b \in \Sigma$ and $b \neq a$, we have:

$$\mathcal{L}(\overline{\partial}_a(\gamma)) = \mathcal{L}(\Sigma^*) = \Sigma^* = \Sigma^* \backslash \mathcal{L}(\emptyset) = \Sigma^* \backslash \mathcal{L}(\partial_\sigma(\gamma)).$$

Finally, if $\alpha = a$, where $a \in \Sigma$, the following holds:

$$\mathcal{L}(\overline{\partial}_a(a)) = \mathcal{L}(\{\Sigma^+\}) = \Sigma^* \backslash \mathcal{L}(\{\varepsilon\}) = \Sigma^* \backslash \mathcal{L}(\partial_a(a)).$$

Let α and β be extended regular expressions over Σ. Suppose by induction hypothesis that $\mathcal{L}(\overline{\partial}_a(\alpha)) = \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha))$ and $\mathcal{L}(\overline{\partial}_a(\beta)) = \Sigma^* \backslash \mathcal{L}(\partial_a(\beta))$.

1. If $\gamma \equiv \alpha + \beta$:

$$\begin{aligned}
\mathcal{L}(\overline{\partial}_a(\alpha + \beta)) &= \mathcal{L}(\overline{\partial}_a(\alpha) \curlywedge \overline{\partial}_a(\beta)) = \mathcal{L}(\overline{\partial}_a(\alpha)) \cap \mathcal{L}(\overline{\partial}_a(\beta)) \\
&= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha)) \cap \Sigma^* \backslash \mathcal{L}(\partial_a(\beta)) \\
&= \Sigma^* \backslash (\mathcal{L}(\partial_a(\alpha) \cup \mathcal{L}(\partial_a(\beta))) \\
&= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha + \beta)).
\end{aligned}$$

2. If $\gamma \equiv \alpha \cap \beta$:

$$\begin{aligned}\mathcal{L}(\overline{\partial}_a(\alpha \cap \beta)) &= \mathcal{L}(\overline{\partial}_a(\alpha) \cup \overline{\partial}_a(\beta)) = \mathcal{L}(\overline{\partial}_a(\alpha)) \cup \mathcal{L}(\overline{\partial}_a(\beta)) \\ &= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha)) \cup \Sigma^* \backslash \mathcal{L}(\partial_a(\beta)) \\ &= \Sigma^* \backslash (\mathcal{L}(\partial_a(\alpha) \cap \mathcal{L}(\partial_a(\beta))) \\ &= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha \cap \beta)).\end{aligned}$$

3. If $\gamma \equiv \alpha\beta$, there are two possible cases: $\varepsilon(\alpha) = \varepsilon$ and $\varepsilon(\alpha) = \emptyset$. The Lemma 5.3 is used in both proofs. If $\varepsilon(\alpha) = \emptyset$, then we have:

$$\begin{aligned}\mathcal{L}(\overline{\partial}_a(\alpha\beta)) &= \mathcal{L}(\neg(\partial_a(\alpha)\beta)) = \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha)\beta) \\ &= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha\beta)).\end{aligned}$$

On the other hand, if $\varepsilon(\alpha) = \varepsilon$, we have:

$$\begin{aligned}\mathcal{L}(\overline{\partial}_a(\alpha\beta)) &= \mathcal{L}(\neg(\partial_a(\alpha)\beta) \barwedge \overline{\partial}_a(\beta)) = \mathcal{L}(\neg(\partial_a(\alpha)\beta)) \cap \mathcal{L}(\overline{\partial}_a(\beta)) \\ &= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha)\beta) \cap \Sigma^* \backslash \mathcal{L}(\partial_a(\beta)) \\ &= \Sigma^* \backslash (\mathcal{L}(\partial_a(\alpha)\beta) \cup \mathcal{L}(\partial_a(\beta)) \\ &= \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha\beta)).\end{aligned}$$

4. If $\gamma \equiv \alpha^*$, by Lemma 5.3:

$$\mathcal{L}(\overline{\partial}_a(\alpha^*)) = \mathcal{L}(\neg(\partial_a(\alpha^*))) = \Sigma^* \backslash \mathcal{L}(\partial_a(\alpha^*)).$$

5. If $\gamma \equiv \neg\alpha$:

$$\begin{aligned}\mathcal{L}(\overline{\partial}_a(\neg\alpha)) &= \mathcal{L}(\partial_a(\alpha)) = \Sigma^* \backslash (\Sigma^* \backslash \mathcal{L}(\partial_a(\alpha))) \\ &= \Sigma^* \backslash \mathcal{L}(\overline{\partial}_a(\alpha)) = \Sigma^* \backslash \mathcal{L}(\partial_a(\neg\alpha)).\end{aligned}$$

□

Lemma 5.14. *For all $\alpha \in \mathsf{RE}_{\cap,\neg}$ over Σ and $a \in \Sigma$, $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$.*

Proof. Let γ be an extended regular expression over Σ and $a \in \Sigma$. The proof proceeds by induction on the structure of γ. The proof for all cases, excluding $\gamma \equiv \neg\alpha$, has already been proved (see Lemma 4.7 and Antimirov [2]). Therefore, here we only present the proof for that case. Given $\alpha \in \mathsf{RE}_{\cap,\neg}$, suppose by inductive hypothesis

that $\mathcal{L}(\partial_a(\alpha)) = a^{-1}\mathcal{L}(\alpha)$. Note that, from the definition of left-quotient of a language, we have $a^{-1}\mathcal{L}(\neg\alpha) = \Sigma^*\backslash a^{-1}\mathcal{L}(\alpha)$ and, from Lemma 5.13, $\mathcal{L}(\overline{\partial}_a(\alpha)) = \Sigma^*\backslash\mathcal{L}((\partial_a(\alpha))$. Then, the following holds:

$$\begin{aligned}\mathcal{L}(\partial_a(\neg\alpha)) &= \mathcal{L}(\partial_a(\neg\alpha)) = \mathcal{L}(\overline{\partial}_a(\alpha)) \\ &= \Sigma^*\backslash\mathcal{L}((\partial_a(\alpha)) = \Sigma^*\backslash a^{-1}\mathcal{L}(\alpha).\end{aligned}$$

□

Proposition 5.15. *Given $w \in \Sigma^*$ and $\alpha \in \mathsf{RE}_\cap$, $\mathcal{L}(\partial_w(\alpha)) = w^{-1}\mathcal{L}(\alpha)$.*

Proof. The proof follows in the same way as the proof for simple regular expressions (see Proposition 3.8). □

There are regular expressions such that $\neg\partial_a(\alpha)$ is equal to $\overline{\partial}_a(\alpha)$. For example, given $\alpha, \beta \in \mathsf{RE}_{\cap,\neg}$, if $\varepsilon(\alpha) = \emptyset$, we have $\overline{\partial}_a(\alpha\beta) = \neg(\partial_a(\alpha)\beta) = \neg\partial(\alpha\beta)$. Moreover, $\overline{\partial}_a(\alpha^*) = \neg\partial_a(\alpha^*)$.

Example 5.16. *For the extended regular expression $\alpha \equiv \neg(a^*a)$, we have the following sets of partial derivatives:*

$$\begin{aligned}&\partial_a(\alpha) = \overline{\partial}_a(a^*a) = \neg(\partial_a(a^*)a) \cap \overline{\partial}_a(a) = \{\neg(a^*a)\} \cap \{\Sigma^+\} = \{\neg(a^*a) \cap \Sigma^+\} \\ &\partial_{aa}(\alpha) = \partial_a(\neg(a^*a) \cap \Sigma^+) = \overline{\partial}_a(a^*a) \cap \partial_a(\Sigma^+) = \{\neg(a^*a) \cap \Sigma^+\} \cap \{\Sigma^*\} \\ &\qquad \{\neg(a^*a) \cap \Sigma^+ \cap \Sigma^*\} \\ &\cdots \\ &\partial_{\underbrace{a\cdots a}_{n}}(\neg(a^*a)) = \{\neg(a^*a) \cap \Sigma^+ \cap \underbrace{\Sigma^* \cap \cdots \cap \Sigma^*}_{n-1}\}\end{aligned}$$

Note that it leads to the same sets of partial derivatives as the natural extension (cf. Example 5.7).

Proposition 5.17. *Every extended regular expression has a finite number of $CII_\cap$-dissimilar partial derivatives.*

Proof. The extension of partial derivatives using the set $\overline{\partial}_a$ is an improved version of the natural extension. However, in the worst case, it is equal to the natural extension (for example $\overline{\partial}_a(\alpha^*) = \neg\partial_a(\alpha^*)$ and Example 5.16). Since $CII_\cap$-dissimilarity is sufficient to guarantee the finitiness of the natural extension, it is also sufficient for the improved version. □

5.4 Systems of Equations

In this section, we extend the recursive definition of a support of regular expressions with intersection to extended regular expressions.

Proposition 5.18. *Let α be an extended regular expression over Σ. Then the set π, defined recursively as follows, is a support of α:*

$$\begin{array}{lll}
\pi(\Sigma^*) = \{\Sigma^*\}, & \pi(a) = \{\varepsilon\}, & \pi(\alpha^*) = \pi(\alpha)\alpha^*, \\
\pi(\Sigma^+) = \{\Sigma^+\}, & \pi(\alpha + \beta) = \pi(\alpha) \cup \pi(\beta), & \pi(\neg\alpha) = \{\alpha' \in \neg S \mid S \in 2^{\pi(\alpha)}\}, \\
\pi(\varepsilon) = \emptyset, & \pi(\alpha \cap \beta) = \pi(\alpha) \curlywedge \pi(\beta), & \\
\pi(\emptyset) = \emptyset, & \pi(\alpha\beta) = \pi(\alpha)\beta \cup \pi(\beta), &
\end{array}$$

Proof. Let $\gamma_0 \in \mathsf{RE}_{\cap,\neg}$ over Σ. The proof proceeds by induction on the structure of γ_0. The proof for all the cases, excepting for Σ^+, Σ^* and $\neg\alpha$, can be found in Proposition 3.14 and in Proposition 4.16. So here we only prove that cases. In order to prove that a given set π is a support of γ_0, we need to prove that the equation (3.14) holds for γ_0 and for each regular expression in π. We start the proof by proving it for the base cases. If $\gamma_0 \equiv \Sigma^*$, we have that

$$\Sigma^* = a_1 \cdot \Sigma^* + \cdots + a_k \cdot \Sigma^* + \varepsilon,$$

and thus the set $\{\Sigma^*\}$ is a support of Σ^*. In the case that $\gamma_0 \equiv \Sigma^+$, we have

$$\Sigma^+ = a_1 \cdot \Sigma^* + \cdots + a_k \cdot \Sigma^* + \emptyset$$
$$\Sigma^* = a_1 \cdot \Sigma^* + \cdots + a_k \cdot \Sigma^* + \varepsilon,$$

and we can conclude that the set $\{\Sigma^*\}$ is a support of Σ^+.

Now, given $\alpha_0 \in \mathsf{RE}_{\cap,\neg}$, suppose by inductive hypothesis that $\pi(\alpha_0) = \{\alpha_1, \ldots, \alpha_n\}$ is a support of α_0. Therefore, from the definition of support, we have that

$$\alpha_i = a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i), \quad \text{for all } i \in \{0, \ldots, n\}$$

where each α_{li}, for $l = 1, \ldots, k$, is a sum of elements of $\pi(\alpha_0)$.

If $\gamma_0 \equiv \neg\alpha_0$, then $\pi(\gamma_0) = \pi(\neg\alpha_0) = \{\alpha' \in \neg S \mid S \in 2^{\pi(\alpha_0)}\}$. That is, $\pi(\gamma_0) = \{\Sigma^*\} \cup \{\bigcap_{i\in I} \neg\alpha_i \mid I \subseteq \{1, \ldots, n\}\}$. Firstly, we prove that the equation (3.14) holds for $\neg\alpha_0$. From the equation (5.3),

$$\neg\alpha_0 = \neg(a_1\alpha_{10} + \cdots + a_k\alpha_{k0} + \varepsilon(\alpha_0))$$

$$= a_1\neg\alpha_{10} + \cdots + a_k\neg\alpha_{k0} + \varepsilon(\neg\alpha_0).$$

We know that each α_{l0}, where $l = 1, \ldots, k$, is a sum of elements of $\pi(\alpha_0)$. So, let I_{l0} be the set of indexes of the regular expressions in $\pi(\alpha_0)$ such that $\alpha_{l0} = \sum_{i' \in I_{l0}} \alpha_{i'}$. Then, for each $l = 1, \ldots, k$, the following holds:

$$\neg\alpha_{l0} = \neg \sum_{i' \in I_{l0}} \alpha_{i'} = \bigcap_{i' \in I_{l0}} \neg\alpha_{i'} \in \{\Sigma^*\} \cup \Big\{ \bigcap_{i \in I} \neg\alpha_i \mid I \subseteq \{1, \ldots, n\} \Big\}.$$

Therefore, we can conclude that each α_{l0} is a sum of elements of $\pi(\neg\alpha_0)$. We still have to prove it for each element of $\pi(\neg\alpha_0)$. The proof for the element Σ^* is already done. So, let $I_j \subseteq \{1, \ldots, n\}$ and $\beta_j = \bigcap_{i \in I_j} \neg\alpha_i$. Using the equations (4.21) and (5.3), we have the following:

$$\begin{aligned}
\beta_j &= \bigcap_{i \in I_j} \neg\alpha_i \\
&= \bigcap_{i \in I_j} \neg(a_1\alpha_{1i} + \cdots + a_k\alpha_{ki} + \varepsilon(\alpha_i)) \\
&= \bigcap_{i \in I_j} (a_1\neg\alpha_{1i} + \cdots + a_k\neg\alpha_{ki} + \varepsilon(\neg\alpha_i)) \\
&= a_1 \bigcap_{i \in I_j} (\neg\alpha_{1i}) + \cdots + a_k \bigcap_{i \in I_j} (\neg\alpha_{ki}) + \varepsilon\Big(\bigcap_{i \in I_j} \neg\alpha_i \Big).
\end{aligned}$$

We also know that each α_{li}, for $l = 1, \ldots, k$ and $i \in I$, is a sum of elements of $\pi(\alpha_0)$. So, let I_{li} be the set of indexes of the regular expressions in $\pi(\alpha_0)$ such that $\alpha_{li} = \sum_{i' \in I_{l0}} \alpha_{i'}$. Then, it holds that:

$$\begin{aligned}
\bigcap_{i \in I_j} \neg\alpha_{li} = \bigcap_{i \in I_j} \Big(\neg \sum_{i' \in I_{li}} \alpha_{i'} \Big) &= \bigcap_{i \in I_j} \Big(\bigcap_{i' \in I_{li}} \neg\alpha_{i'} \Big) \\
&= \bigcap_{i \in I_j, i' \in I_{li}} \neg\alpha_{i'} \\
&\in \{\Sigma^*\} \cup \Big\{ \bigcap_{i \in I} \neg\alpha_i \mid I \subseteq \{1, \ldots, n\} \Big\}
\end{aligned}$$

And thus each α_{li} is a sum of elements of $\pi(\neg\alpha_0)$. With this, we can conclude that the set $\pi(\neg\alpha_0)$ is a support of $\neg\alpha_0$. □

As consequence of the last proposition, the support of an extended regular expression is ensured to be finite even not considering extended regular expressions modulo-CII$_\cap$.

5.4.1 Support and Partial Derivatives

In the previous chapter, we proved that, for every $\alpha \in \mathsf{RE}_{\cap}$, the set of all partial derivatives of α is a subset of $\pi(\alpha)$. Here, with α being any extended regular expression, we prove that the set of all $\mathrm{CII}_{\cap}$-dissimilar partial derivatives of α, using the natural extension, is a subset of $\pi(\alpha)$. Thereby, in this subsection, we use the definition of the natural extension of partial derivatives, presented in the section 5.3.1.

Proposition 5.19. *Given $\alpha \in RE_{\cap,\neg}$, $\partial^+(\alpha) \subseteq_{CII_\cap} \pi(\alpha)$.*

Proof. The proof proceeds by induction on the structure of $\gamma \in \mathsf{RE}_{\cap,\neg}$. The proof for all cases, excluding $\neg\alpha$, is given in Proposition 4.20. Thus here we only present the proof for that case. Given $\alpha \in \mathsf{RE}_{\cap,\neg}$, suppose by inductive hypothesis that $\partial^+(\alpha) \subseteq_{CII_\cap} \pi(\alpha)$. If $\gamma \equiv \neg\alpha$, then the following holds:

$$\begin{aligned}\partial^+(\neg\alpha) &\subseteq_{CII_\cap} \{\neg S \mid S \in 2^{\partial^+(\alpha)}\} \\ &\subseteq_{CII_\cap} \{\neg S \mid S \in 2^{\pi(\alpha)}\} = \pi(\neg\alpha).\end{aligned}$$

□

5.5 $\mathsf{RE}_{\cap,\neg}$ to Finite Automata

As presented in the previous sections, the set of all derivatives and the set of all partial derivatives of extended regular expressions can be infinite. In this way, the construction of Brzozowski's and partial derivative's automaton may not terminate. In the other hand, Glushkov's and Thompson's automaton does not generalize to extended regular expressions. m

Chapter 6

Special Regular Expressions

As seen in the previous chapters, to ensure the finitude of the set of all derivatives and of the set of all partial derivatives of an extended regular expression, it is necessary to consider ACI_+-dissimilarity and $\mathrm{CII}_\cap$-dissimilarity. In order to guarantee these dissimilarities, we introduce in this chapter a new set of regular expressions, called special regular expression.

Thus, for a *special regular expression* the following properties are considered:

- the associative property of disjunction, intersection and concatenation:

$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma, \tag{6.1}$$
$$\alpha \cap (\beta \cap \gamma) = (\alpha \cap \beta) \cap \gamma, \tag{6.2}$$
$$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma; \tag{6.3}$$

- the commutative property of disjunction and intersection:

$$\alpha + \beta = \beta + \alpha, \tag{6.4}$$
$$\alpha \cap \beta = \beta \cap \alpha; \tag{6.5}$$

- the idempotent property of disjunction and intersection:

$$\alpha + \alpha = \alpha, \tag{6.6}$$
$$\alpha \cap \alpha = \alpha; \tag{6.7}$$

- the identity element property of disjunction, intersection and concatenation:

$$\alpha + \emptyset = \emptyset + \alpha = \alpha, \tag{6.8}$$

$$\alpha \cap \Sigma^* = \Sigma^* \cap \alpha = \alpha, \tag{6.9}$$

$$\alpha \cdot \varepsilon = \varepsilon \cdot \alpha = \alpha; \tag{6.10}$$

- and the zero element property of disjunction, intersection and concatenation:

$$\alpha + \Sigma^* = \Sigma^* + \alpha = \Sigma^*, \tag{6.11}$$

$$\alpha \cap \emptyset = \emptyset \cap \alpha = \emptyset, \tag{6.12}$$

$$\alpha \cdot \emptyset = \emptyset \cdot \alpha = \emptyset. \tag{6.13}$$

We denote by S the collection of the properties above, (6.1) to (6.13).

Note that the set of all special regular expressions is the set of all S-dissimilar extended regular expressions. And the ACI_+ and the $\text{CII}_\cap$ properties belong to S. Thus, the set of all derivatives of any special regular expression is finite, with the same being true for the set of all partial derivatives.

6.1 Rewriting System S

One possible approach to obtain the set of all S-dissimilar extended regular expression would be to rewrite every extended regular expression in such way that S-similar regular expressions are identified. Krauss and Nipkow [20] presented a rewriting system for simple regular expression which identifies expressions that are equivalent modulo associativity, commutativity and idempotence of $+$, associativity of $\cdot$, zero of $+$ and $\cdot$ and identity of $\cdot$.

If we add the associativity, commutativity and idempotence rules of $\cap$ ($\text{ACI}_\cap$), the zero rule of $\cap$, and the identity rules of $+$ and $\cap$ to the Krauss and Nipkow's system, we obtain a rewriting system to identify S-similar expressions. This extension is trivial, since the $\text{ACI}_\cap$ and ACI_+ rules are similar, as well the zero and identity rules of all the operators. The resulting extension is called *rewriting system S* and it is given by the following:

$$\emptyset + \alpha \to \alpha, \qquad\qquad \Sigma^* \cap \alpha \to \alpha,$$

$$
\begin{array}{ll}
\alpha + \emptyset \to \alpha, & \alpha \cap \Sigma^* \to \alpha, \\
\Sigma^* + \alpha \to \Sigma^*, & \emptyset \cap \alpha \to \emptyset, \\
\alpha + \Sigma^* \to \Sigma^*, & \alpha \cap \emptyset \to \emptyset, \\
(\alpha + \beta) + \gamma \to \alpha + (\beta + \gamma), & (\alpha \cap \beta) \cap \gamma \to \alpha \cap (\beta + \gamma), \\
\alpha + (\beta + \gamma) \to \alpha + \gamma, \text{ if } \alpha = \beta, & \alpha \cap (\beta \cap \gamma) \to \alpha \cap \gamma, \text{ if } \alpha = \beta, \\
\alpha + (\beta + \gamma) \to \beta + (\alpha + \gamma), \text{ if } \beta < \alpha, & \alpha \cap (\beta \cap \gamma) \to \beta \cap (\alpha \cap \gamma), \text{ if } \beta < \alpha, \\
\alpha + \alpha \to \alpha, & \alpha \cap \alpha \to \alpha, \\
\alpha + \beta \to \beta + \alpha, \text{ if } \beta < \alpha, & \alpha \cap \beta \to \beta \cap \alpha, \text{ if } \beta < \alpha,
\end{array}
$$

$$
\begin{array}{l}
\varepsilon \cdot \alpha \to \alpha, \\
\alpha \cdot \varepsilon \to \alpha, \\
\emptyset \cdot \alpha \to \emptyset, \\
\alpha \cdot \emptyset \to \emptyset, \\
(\alpha \cdot \beta) \cdot \gamma \to \alpha \cdot (\beta + \gamma),
\end{array}
$$

where α, β and γ are any extended regular expression and $<$ some arbitrary total order on extended regular expressions. The expression resulting from the application of the rewriting system S is in *S-normal formal*.

Example 6.1. *Considering* $\emptyset < \varepsilon < \Sigma^+ < \Sigma^* < a \in \Sigma < \neg < * < \cdot < \cap < +$ *and* Σ *lexicographically ordered, the S-normal form of the extended regular expression* $\varepsilon aa + \neg a + aa + (\emptyset + (a^* \cap \Sigma^* \cap ab))$ *is* $\neg a + (aa + a^* \cap ab)$.

6.2 Special Representation

Sets are a natural way to enforce associativity, commutativity and idempotence. Lists, in the other hand, enforce the associativity, non-commutativity and non-idempotence. Thereby, we represent disjunctions and intersections as sets and concatenations as lists.

Definition 6.2. ***(Special Regular Expression)*** *Let* Σ *be an alphabet. A special regular expression over* Σ *is given by the following grammar:*

$$\varphi := \emptyset \mid \Sigma^* \mid \alpha,$$

$$
\begin{aligned}
&\alpha := +\{\beta_1, \ldots, \beta_n\} \mid \beta,\\
&\beta := \cap\{\beta'_1, \ldots, \beta'_n\} \mid \delta,\\
&\beta' := +\{\beta_1, \ldots, \beta_n\} \mid \delta,\\
&\delta := \cdot[\delta'_1, \ldots, \delta'_n] \mid \varepsilon \mid \gamma,\\
&\delta' := +\{\beta_1, \ldots, \beta_n\} \mid \cap\{\beta'_1, \ldots, \beta'_n\} \mid \Sigma^* \mid \gamma,\\
&\gamma := \varepsilon \mid \Sigma^+ \mid a \in \Sigma \mid (\alpha)^* \mid \neg(\alpha),
\end{aligned}
$$

where n is greater than one. The structures $+\{\_\}$ and $\cap\{\_\}$ have the same properties as sets and the structure $\cdot[\_]$ has the properties of lists. The language $\mathcal{L}(\varphi)$ associated with φ is defined for the base cases ($\emptyset$, ε, Σ^*, Σ^+ and $a \in \Sigma$), for $\neg\alpha$ and for α^* as for $\mathsf{RE}_{\cap,\neg}$. For the other cases is defined as:

$$
\begin{aligned}
&\mathcal{L}(+\{\beta_1, \ldots, \beta_n\}) = \mathcal{L}(\beta_1 + \cdots + \beta_n),\\
&\mathcal{L}(\cap\{\beta'_1, \ldots, \beta'_n\}) = \mathcal{L}(\beta'_1 \cap \cdots \cap \beta'_n),\\
&\mathcal{L}(\cdot[\delta'_1, \ldots, \delta'_n]) = \mathcal{L}(\delta'_1 \cdots \delta'_n).
\end{aligned}
$$

The set of all special regular expressions is denoted by RE_S.

The grammar defining RE_S and the algebra of the structures defining disjuctions and intersections enforce the S-dissimilarity. The grammar of RE_S ensures the identity element and zero element properties and that syntactically identical expressions modulo associativity of disjunction, intersection and concatenation are not generated. For example, for concatenation, the regular expressions $\cdot[a,\ \varepsilon, a]$, $\cdot[a,\ \emptyset,\ a]$ and $\cdot[a,\ \cdot[a,\ b]]$ are not generated by the grammar. The structures $+\{\_\}$ and $\cap\{\_\}$ ensure that there are no syntactically identical regular expressions modulo commutativity and idempotence of disjunction and intersection, respectively.

We will assume the precedence $\neg > *$. We define equivalence, empty word property and alphabetic size in the same way as for RE. Being φ a special regular expression, the size of φ, written $|\varphi|$, is defined inductively by:

$$
\begin{aligned}
&|\Sigma^*| = |\Sigma^+| = 1, && |+\{\beta_1, \ldots, \beta_n\}| = |\beta_1| + \cdots + |\beta_n| + n - 1,\\
&|\varepsilon| = |\emptyset| = 1, && |\cap\{\beta'_1, \ldots, \beta'_n\}| = |\beta'_1| + \cdots + |\beta'_n| + n - 1,\\
&|a| = 1, \quad \text{where } a \in \Sigma && |\cdot[\delta'_1, \ldots, \delta'_n]| = |\delta'_1| + \cdots + |\delta'_n| + n - 1.\\
&|\alpha^*| = |\neg\alpha| = 1 + |\alpha|, &&
\end{aligned}
$$

Example 6.3. *The extended regular expression* $\alpha = \neg a + aa + a^* \cap ab$ *is equivalent to the special regular expression* $\varphi = +\{\neg a, \cdot[a,a], \cap\{a^*, \cdot[a,b]\}\}$. *The size of* φ *is 13.*

The disjunction, intersection and concatenation of extended regular expressions are represented by binary operators with domain $\mathsf{RE}_{\cap,\neg} \times \mathsf{RE}_{\cap,\neg}$. In special regular expressions these operations are represented by structures with restricted elements. For example, a disjunction is not an element of the structure of a disjunction and the symbol ε is not an element of the structure of a concatenation. In this way, we need to extend the disjunction, intersection and concatenation operators to special regular expressions.

We begin by defining the binary operator $\oplus : \mathsf{RE}_S \times \mathsf{RE}_S \to \mathsf{RE}_S$, which is an extension of disjunction to special regular expressions. This operator is commutative and it is defined inductively as follows:

$$\varphi \oplus \Sigma^* = \Sigma^*, \tag{6.14}$$

$$\varphi \oplus \emptyset = \varphi, \tag{6.15}$$

$$\varphi \oplus \varphi = \varphi, \tag{6.16}$$

$$\beta_1 \oplus \beta_2 = +\{\beta_1, \beta_2\}, \text{ where } \beta_1 \not\equiv \beta_2, \tag{6.17}$$

$$+\{\beta_1, \ldots, \beta_n\} \oplus \beta_{n+1} = +\{\beta_1, \ldots, \beta_n\}, \text{ if } \exists i \in \{1, \ldots, n\}.\, \beta_i \equiv \beta_{n+1}, \tag{6.18}$$

$$+\{\beta_1, \ldots, \beta_n\} \oplus \beta_{n+1} = +\{\beta_1, \ldots, \beta_n, \beta_{n+1}\}, \text{ if } \forall i \in \{1, \ldots, n\}.\, \beta_i \not\equiv \beta_{n+1}, \tag{6.19}$$

$$+\{\beta_1, \ldots, \beta_n\} \oplus +\{\beta_{n+1}, \beta_{n+2}, \ldots, \beta_{n+m}\} = (+\{\beta_1, \ldots, \beta_n\} \oplus \beta_{n+1}) \oplus +\{\beta_{n+2}, \ldots, \beta_{n+m}\}. \tag{6.20}$$

The binary operator $\circledcap : \mathsf{RE}_S \times \mathsf{RE}_S \to \mathsf{RE}_S$ is an extension of intersection to special regular expressions, which is commutative and defined inductively by:

$$\varphi \circledcap \emptyset = \emptyset \tag{6.21}$$

$$\varphi \circledcap \Sigma^* = \varphi \tag{6.22}$$

$$\varphi \circledcap \varphi = \varphi \tag{6.23}$$

$$\beta'_1 \circledcap \beta'_2 = \cap\{\beta'_1, \beta'_2\}, \text{ where } \beta'_1 \not\equiv \beta'_2, \tag{6.24}$$

$$\cap\{\beta_1, \ldots, \beta_n\} \circledcap \beta_{n+1} = \cap\{\beta'_1, \ldots, \beta'_n\}, \text{ if } \exists i \in \{1, \ldots, n\}.\, \beta'_i \equiv \beta'_{n+1}, \tag{6.25}$$

$$\cap\{\beta'_1, \ldots, \beta'_n\} \circledcap \beta'_{n+1} = \cap\{\beta'_1, \ldots, \beta'_n, \beta'_{n+1}\}, \text{ if } \forall i \in \{1, \ldots, n\}.\, \beta'_i \not\equiv \beta'_{n+1}, \tag{6.26}$$

$$\cap\{\beta'_1, \ldots, \beta'_n\} \circledcap \cap\{\beta'_{n+1}, \beta'_{n+2}, \ldots, \beta'_{n+m}\} =$$

$$(\cap\{\beta'_1, \ldots, \beta'_n\} \oplus \beta'_{n+1}) \circledcap \cap\{\beta'_{n+2}, \ldots, \beta'_{n+m}\}. \tag{6.27}$$

Finnaly, we define the binary operator $\odot : \mathsf{RE}_S \times \mathsf{RE}_S \to \mathsf{RE}_S$ as an extension of concatenation to special regular expressions. This operator is defined inductively by the following:

$$\varphi \odot \varepsilon = \varepsilon \odot \varphi = \varphi, \tag{6.28}$$
$$\varphi \odot \emptyset = \emptyset \odot \varphi = \emptyset, \tag{6.29}$$
$$\delta'_1 \odot \delta'_2 = \cdot[\delta'_1, \delta'_2], \tag{6.30}$$
$$\delta'_1 \odot \cdot[\delta'_2, \ldots, \delta'_n] = \cdot[\delta'_1, \delta'_2, \ldots, \delta'_n], \tag{6.31}$$
$$\cdot[\delta'_1, \ldots, \delta'_n] \odot \delta'_{n+1} = \cdot[\delta'_1, \ldots, \delta'_n, \delta'_{n+1}], \tag{6.32}$$
$$\cdot[\delta'_1, \ldots, \delta'_n] \odot \cdot[\delta'_{n+1}, \ldots, \delta'_m] = \cdot[\delta'_1, \ldots, \delta'_n, \delta'_{n+1}, \ldots, \delta'_{n+m}]. \tag{6.33}$$

Example 6.4. *According to the definition of the above operators, we have the following equivalences:*

$$\cap\{a, b\} \oplus \cap\{a, b\} = \cap\{a, b\},$$
$$\cdot[+\{a, \varepsilon\}, \Sigma^*] \circledcap \cap\{a, \cdot[+\{a, \varepsilon\}, \Sigma^*]\} = \cap\{a, \cdot[+\{a, \varepsilon\}, \Sigma^*]\},$$
$$\cdot[a, +\{a^*, b\}] \odot \varepsilon = \cdot[a, +\{a^*, b\}].$$

The function $\varepsilon : \mathsf{RE}_S \to \{\emptyset, \varepsilon\}$ is defined inductively for the base cases and for the inductive cases α^* and $\neg\alpha$ as for $\mathsf{RE}_{\cap,\neg}$, for the other cases it is defined as follows:

$$\varepsilon(+\{\beta_1, \ldots, \beta_n\}) = \varepsilon(\beta_1) \oplus \cdots \oplus \varepsilon(\beta_n),$$
$$\varepsilon(\cap\{\beta'_1, \ldots, \beta'_n\}) = \varepsilon(\beta'_1) \circledcap \cdots \circledcap \varepsilon(\beta'_n),$$
$$\varepsilon(\cdot[\delta'_1, \ldots, \delta'_n]) = \varepsilon(\delta'_1) \odot \cdots \odot \varepsilon(\delta'_n).$$

6.3 Derivatives

With the extension of the function ε and of the binary operators representing the disjunction, intersection and concatenation, it becomes trivial the extension of the notion of derivative to special regular expressions.

Definition 6.5. ***(Derivative)*** *The derivative of a special regular expression φ w.r.t. a symbol $a \in \Sigma$, written $d_a(\varphi)$, is defined for the base cases as for $\mathsf{RE}_{\cap,\neg}$, the other*

cases are defined as follows:

$$
\begin{aligned}
&d_a(+\{\beta_1,\ldots,\beta_n\}) = d_a(\beta_1)\oplus\cdots\oplus d_a(\beta_n),\\
&d_a(\cap\{\beta'_1,\ldots,\beta'_n\}) = d_a(\beta'_1)\mathbin{\text{⩀}}\cdots\mathbin{\text{⩀}} d_a(\beta'_n),\\
&d_a(\cdot[\delta'_1,\delta'_2]) = (d_a(\delta'_1)\odot\delta'_2)\oplus(\varepsilon(\delta'_1)\odot d_a(\delta'_2)),\\
&d_a(\cdot[\delta'_1,\delta'_2,\ldots,\delta'_n]) = (d_a(\delta'_1)\odot\cdot[\delta'_2,\ldots,\delta'_n])\oplus(\varepsilon(\delta'_1)\odot d_a(\cdot[\delta'_2,\ldots,\delta'_n])),\\
&d_a(\alpha^*) = d_a(\alpha)\odot\alpha^*,\\
&d_a(\neg\alpha) = \neg d_a(\alpha).
\end{aligned}
$$

Example 6.6. *The derivative of the special regular expression $\neg\cdot[a^*,a]$ w.r.t. to the symbol a is*

$$
\begin{aligned}
d_a(\neg\cdot[a^*,a]) &= \neg((d_a(a^*)\odot a)\oplus(\varepsilon(a^*)\odot d_a(a)))\\
&= \neg((d_a(a)\odot a^*\odot a)\oplus(\varepsilon\odot d_a(a)))\\
&= \neg((\varepsilon\odot a^*\odot a)\oplus(\varepsilon\odot\varepsilon))\\
&= \neg(\cdot[a^*,a]\oplus\varepsilon)\\
&= \neg(+\{\cdot[a^*,a],\varepsilon\}).
\end{aligned}
$$

The derivative of a special regular expression w.r.t. a word is defined in the same way as for extended regular expressions. Given $\varphi\in\mathsf{RE}_S$ over Σ and $w\in\Sigma^*$, since every special regular expression is equivalent to an extended regular expression, we know that $\mathcal{L}(d_w(\varphi)) = w^{-1}\mathcal{L}(\varphi)$ holds. From the equation (3.1), it is easy to obtain the following equivalence:

$$\varphi = \left(\bigoplus_{a\in\Sigma} a\odot d_a(\varphi)\right)\oplus\varepsilon(\varphi).$$

Furthermore, the set of all derivatives of any special regular expression is finite.

6.4 Partial Derivatives

Analogously as what was done for derivatives, it is easy to extend the definition of partial derivatives to special regular expressions. To do this, we must extend the operators of concatenation, intersection and complement to sets of special regular

expressions. We trivially extend this by using the definition of these operators for extended regular expressions.

An extension of concatenation to sets of special regular expressions is given by the operator $\odot : 2^{\mathsf{RE}_S} \times \mathsf{RE}_S \to 2^{\mathsf{RE}_S}$, which is defined inductively by:

$$\begin{aligned}
&\emptyset \odot \varphi = \emptyset,\\
&S \odot \emptyset = \emptyset,\\
&\{\varphi_1, \varphi_2, \ldots, \varphi_n\} \odot \varphi = \{\varphi_1 \odot \varphi\} \cup (\{\varphi_2, \ldots, \varphi_n\} \odot \varphi).
\end{aligned}$$

The binary operator $\owedge : 2^{\mathsf{RE}_S} \times 2^{\mathsf{RE}_S} \to 2^{\mathsf{RE}_S}$, in other hand, is an extension of intersection to sets of special regular expression and it is defined inductively as follows:

$$\begin{aligned}
&\emptyset \owedge S = \emptyset,\\
&\{\varphi\} \owedge S = \{\varphi \owedge \varphi_i \mid \varphi_i \in S\},\\
&(\{\varphi\} \cup S) \owedge S' = (\{\varphi\} \owedge S') \cup (S \owedge S').
\end{aligned}$$

Finally, the binary operator $\neg : 2^{\mathsf{RE}_S} \to 2^{\mathsf{RE}_S}$ is an extension of complement to sets of special regular expressions, being defined inductively by:

$$\begin{aligned}
&\neg\emptyset = \{\Sigma^*\},\\
&\neg\{\varphi_1, \ldots, \varphi_n\} = \{\neg\varphi_1 \owedge \cdots \owedge \neg\varphi_n\}.
\end{aligned}$$

Definition 6.7. ***(Partial Derivatives)*** *Given a symbol $a \in \Sigma$ and a special regular expression φ, the set of partial derivatives of φ w.r.t. a, written $\partial_a(\varphi)$ is defined inductively by the following:*

$$\begin{aligned}
&\partial_a(\Sigma^*) = \partial_a(\Sigma^+) = \{\Sigma^*\},\\
&\partial_a(\emptyset) = \partial_a(\varepsilon) = \partial_a(b) = \emptyset, \text{ where } b \in \Sigma \text{ and } b \neq a,\\
&\partial_a(a) = \{\varepsilon\},\\
&\partial_a(+\{\beta_1, \ldots, \beta_n\}) = \partial_a(\beta_1) \cup \cdots \cup \partial_a(\beta_n),\\
&\partial_a(\cap\{\beta'_1, \ldots, \beta'_n\}) = \partial_a(\beta'_1) \owedge \cdots \owedge \partial_a(\beta'_n),\\
&\partial_a(\cdot[\delta'_1, \delta'_2]) = (\partial_a(\delta'_1) \odot \delta'_2) \cup (\varepsilon(\delta'_1) \odot \partial_a(\delta'_2)),\\
&\partial_a(\cdot[\delta'_1, \delta'_2, \ldots, \delta'_n]) = (\partial_a(\delta'_1) \odot \cdot[\delta'_2, \ldots, \delta'_n]) \cup (\varepsilon(\delta'_1) \odot \partial_a(\cdot[\delta'_2, \ldots, \delta'_n])),\\
&\partial_a(\alpha^*) = \partial_a(\alpha) \odot \alpha^*,
\end{aligned}$$

$$\partial_a(\neg\alpha) = \overline{\partial}_a(\alpha),$$

The set $\overline{\partial}_a(\varphi)$ *is defined inductively as follows:*

$$\begin{aligned}
&\overline{\partial}_a(\Sigma^*) = \overline{\partial}_a(\Sigma^+) = \emptyset,\\
&\overline{\partial}_a(\emptyset) = \overline{\partial}_a(\varepsilon) = \overline{\partial}_a(b) = \{\Sigma^*\},\ \textit{where } b \in \Sigma \textit{ and } b \neq a,\\
&\overline{\partial}_a(a) = \{\Sigma^+\},\\
&\overline{\partial}_a(+\{\beta_1, \ldots, \beta_n\}) = \overline{\partial}_a(\beta_1) \cap \cdots \cap \overline{\partial}_a(\beta_n),\\
&\overline{\partial}_a(\cap\{\beta'_1, \ldots, \beta'_n\}) = \overline{\partial}_a(\beta'_1) \cup \cdots \cup \overline{\partial}_a(\beta'_n),\\
&\overline{\partial}_a(\cdot[\delta'_1, \delta'_2]) = \neg(\partial_a(\delta'_1) \odot \delta'_2) \cap \overline{\partial}_a(\delta'_2),\ \ \textit{if } \varepsilon(\delta'_1) = \varepsilon,\\
&\overline{\partial}_a(\cdot[\delta'_1, \delta'_2]) = \neg(\partial_a(\delta'_1) \odot \delta'_2),\ \ \textit{if } \varepsilon(\delta'_1) = \emptyset,\\
&\overline{\partial}_a(\cdot[\delta'_1, \delta'_2, \ldots, \delta'_n]) = \neg(\partial_a(\delta'_1) \odot \cdot[\delta'_2, \ldots, \delta'_n]) \cap \overline{\partial}_a(\cdot[\delta'_2, \ldots, \delta'_n]),\ \ \textit{if } \varepsilon(\delta'_1) = \varepsilon,\\
&\overline{\partial}_a(\cdot[\delta'_1, \delta'_2, \ldots, \delta'_n]) = \neg(\partial_a(\delta'_1) \odot \cdot[\delta'_2, \ldots, \delta'_n]),\ \ \textit{if } \varepsilon(\delta'_1) = \emptyset,\\
&\overline{\partial}_a(\alpha^*) = \neg\partial_a(\alpha^*),\\
&\overline{\partial}_a(\neg\alpha) = \partial_a(\alpha).
\end{aligned}$$

Example 6.8. *The set of partial derivatives of the special regular expression* $\neg \cdot [a^*, a]$ *w.r.t.* a *is*

$$\begin{aligned}
\partial_a(\neg \cdot [a^*, a]) &= \overline{\partial}_a(\cdot[a^*, a]) = \neg(\partial_a(a^*) \odot a) \cap \overline{\partial}_a(a)\\
&= \neg(\partial_a(a) \odot a^* \odot a) \cap \overline{\partial}_a(a) = \neg(\{\varepsilon\} \odot a^* \odot a) \cap \{\Sigma^+\}\\
&= \neg(\{\cdot[a^*, a]\}) \cap \{\Sigma^+\} = \{\neg \cdot [a^*, a]\}) \cap \{\Sigma^+\}\\
&= \{\cap\{\neg \cdot [a^*, a], \Sigma^+\}\}.
\end{aligned}$$

We define the extension of the set of partial derivatives to words, sets of words and sets of special regular expressions in the same way as we defined for simple regular expressions. As for derivatives, $\mathcal{L}(\partial_w(\varphi)) = w^{-1}\mathcal{L}(\varphi)$ holds for every $\varphi \in \mathsf{RE}_{\cap,\neg}$ over any alphabet Σ and for every $w \in \Sigma^*$. Moreover, the set of all partial derivatives of every special regular expression is finite.

Note that this construction of the set of partial derivatives improves the one of Caron et al. because of our compact representation.

6.5 Systems of Equations

In this section we generalize the inductive definition of a support given in Proposition 5.18 to a special regular expression. This extension is given by the next proposition.

Proposition 6.9. *Let φ be a special regular expression over Σ. The set π, defined recursively as follows, is a support of φ:*

$$\pi(\Sigma^+) = \pi(\Sigma^*) = \{\Sigma^*\},$$
$$\pi(\emptyset) = \pi(\varepsilon) = \emptyset,$$
$$\pi(a) = \{\varepsilon\}, \text{ where } a \in \Sigma,$$
$$\pi(+\{\beta_1, \ldots, \beta_n\}) = \pi(\beta_1) \cup \cdots \cup \pi(\beta_n),$$
$$\pi(\cap\{\beta'_1, \ldots, \beta'_n\}) = \pi(\beta'_1) \owedge \cdots \owedge \pi(\beta'_n),$$
$$\pi(\cdot[\delta'_1, \delta'_2]) = \pi(\delta'_1) \odot \delta'_2 \cup \pi(\delta'_2),$$
$$\pi(\cdot[\delta'_1, \delta'_2, \ldots, \delta'_n]) = \pi(\delta'_1) \odot \cdot[\delta'_2, \ldots, \delta'_n] \cup \pi([\delta'_2, \ldots, \delta'_n]),$$
$$\pi(\alpha^*) = \pi(\alpha) \odot \alpha^*$$
$$\pi(\neg\alpha) = \{\beta \in \neg S \mid S \in 2^{\pi(\alpha)}\}.$$

Proof. The proof follows trivially from Proposition 5.18 and from the definition of the set operators $\owedge$, $\odot$ and $\neg$. □

Given $\alpha \in \mathsf{RE}_{\cap,\neg}$, let $\partial^+(\alpha)$ be the set of all partial derivatives of α, considering the natural extension of partial derivatives of an extended regular expression. From Proposition 5.19, we know that $\partial^+(\alpha) \subseteq_{CII_\cap} \pi(\alpha)$, for every $\alpha \in \mathsf{RE}_{\cap,\neg}$. Since the set of all special regular expressions is $\mathrm{CII}_\cap$-dissimilar, we conclude that $\partial^+(\varphi) \subseteq \pi(\varphi)$, for every $\varphi \in \mathsf{RE}_S$.

6.6 RE_S to Finite Automata

The Brzozowski's automaton and the partial derivative's automaton are defined for special regular expressions in the same way as for simple regular expressions. The main advantage of using special regular expressions, compared with extended regular expressions, lies in the fact that for special regular expressions both constructions are always finite.

As previously seen, the construction of the Brzozowski's automaton for the regular expression a^*a is infinite. The following example shows this construction for the equivalent special regular expression $\cdot[a^*, a]$.

Example 6.10. *The Brzozowski's automaton of the regular expression* $\alpha \equiv \cdot[a^*, a]$ *over* $\Sigma = \{a\}$ *is the automaton* $\mathcal{D} = \langle Q, \Sigma, q_0, \delta, F\rangle$*, where* $Q = \{\cdot[a^*, a], +\{\cdot[a^*, a], \varepsilon\}\}$*, the initial state is* $q_0 = \alpha$*, the set of final states is* $F = \{+\{\cdot[a^*, a], \varepsilon\}\}$ *and the transition function is*

$$\begin{aligned}\delta(\cdot[a^*, a], a) &= d_a(\cdot[a^*, a]) = \ \oplus\ \{\cdot[a^*, a], \varepsilon\},\\ \delta(+\{\cdot[a^*, a], \varepsilon\}, a) &= d_a(\cdot[a^*, a]) \oplus d_a(\varepsilon)\\ &= +\{\cdot[a^*, a], \varepsilon\} \oplus \emptyset = +\{\cdot[a^*, a], \varepsilon\}.\end{aligned}$$

Figure 6.1 shows the diagram of $\mathcal{D}$*.*

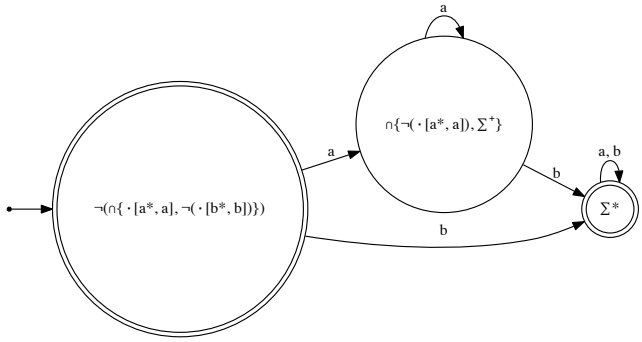


Figure 6.1: Brzozowski's automaton of $\cdot[a^*, a]$.

As we also know, the set of partial derivatives of $\neg(a^*a)$ is infinite. The next example presents the partial derivative's automaton for the equivalent special regular expression, $\neg(\cdot[a^*, a])$.

Example 6.11. *The partial derivative's automaton of the special regular expression* $\alpha \equiv \neg(\cdot[a^*, a])$ *over* $\Sigma = \{a\}$ *is the automaton* $\mathcal{N} = \langle Q, \Sigma, q_0, \delta, F\rangle$*, where* $Q = \{\neg(\cdot[a^*, a]), \cap\{\cdot[a^*, a], \Sigma^+\}\}$*, the initial state is* $q_0 = \alpha$*, the set of final states is* $F = \{\neg(\cdot[a^*, a])\}$ *and the transition function is*

$$\begin{aligned}\delta(\neg(\cdot[a^*, a]), a) &= \partial_a(\neg(\cdot[a^*, a])) = \{\cap\{\neg(\cdot[a^*, a]), \Sigma^+\}\},\\ \delta(\cap\{\neg(\cdot[a^*, a]), \Sigma^+\}, a) &= \partial_a(\neg(\cdot[a^*, a])) \cap \partial_a(\Sigma^*)\\ &= \cap\{\neg(\cdot[a^*, a]), \Sigma^+\} \cap \{\Sigma^*\} = \{\neg(\cdot[a^*, a]), \Sigma^+\}.\end{aligned}$$

The diagram of the automaton $\mathcal{N}$ *is depicted in Figure 6.2.*

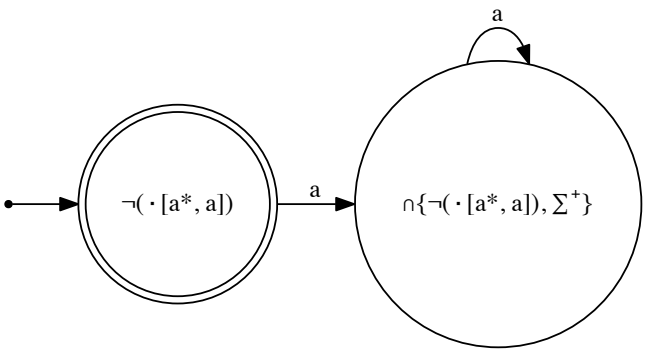


Figure 6.2: Partial derivative's automaton of $\neg(\cdot[a^*, a])$.

6.7 FAdo

For the purpose of representing special regular expressions, new classes were implemented in the module reex: `sigmaS`, `sigmaP`, `sdisj`, `sconj`, `sconcat`, `sstar` and `snot`. The classes `sdisj`, `sconj`, `sconcat`, `sstar` and `snot` represent the disjunction, the intersection, the concatenation, the Kleene star and the complement of special regular expressions, respectively. The classes `sigmaS` and `sigmaP` represent the regular expressions Σ^* and Σ^+, respectively. The alphabet symbols, the empty word and the emptyset are represented as before by the classes `atom`, `epsilon` and `emptyset`. The class diagram of the implementation is shown in Figure 6.3.

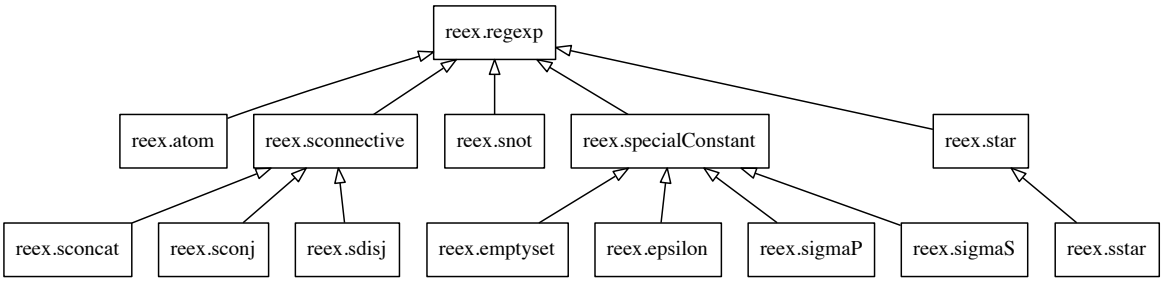


Figure 6.3: Class diagram of `reex` for special regular expressions.

In order to ensure the associative, commutative and idempotent properties of the disjunction and the intersection, these operators were implemented as sets. In the other hand, ordered lists are used to implement concatenation, allowing us to take advantage of the associativity of the concatenation. The operators $\oplus$, $\Cap$ and $\odot$ are

defined by the methods `reex._plus(re)`, `reex._inter(re)` and `reex._dot(re)`, respectively.

To build a special regular expression in FAdo, we use the function `str2sre(str)`. This function converts a string into a special regular expression. The regular expressions Σ^* and Σ^+ are represented by the strings @sigmaS and @sigmaP, respectively. The string $\sim$ represents the complement.

```
>>> from FAdo.reex import *
>>> re = str2sre("~(a*+(@sigmaS & aa + aa))@epsilon")
>>> re
snot(sdisj(frozenset([sconcat((atom("a"), atom("a"))),sstar(
atom("a"))])))
>>> print re
~(aa + a*)
```

The measure methods implemented for reex, including `reex.conjLength()`, are also methods of the new classes. Furthermore, a new measure, which counts the number of nodes of a regular expression, was implemented. This measure is defined by the method `reex.syntacticLength()`.

```
>>> re = str2sre("a*bbb")
>>> re.syntacticLength()
8
>>> re.treeLength()
6
```

The definitions of derivative, set of partial derivatives, linear form and support were also implemented for the new classes. In this way, we are able to build the Brzozowski's automaton or the partial derivative's automaton of a special regular expression. The method to build the Brzozowski's automaton is `reex.dfaBrzozowski()`. For special regular expressions without intersection and complement we are also able to build the Glushkov's automaton. All the methods available for a special regular expression can be found in the FAdo webpage [14].

Chapter 7

Conclusion

In this work we presented several derivative based methods for the conversion of regular expressions with additional operators to finite automata.

For regular expressions with intersection, we proved that Antimirov's and Mirkin's constructions are not identical. Futhermore, we presented a tight worst-case upper bound for the size of both constructions that is exponential with respect to the number of states of the regular expression. This could lead to the assumption of the infeasibility of the manipulation of these expressions. However, as seen in the presented experimental results, the average state complexity of partial derivative's automaton may even be polynomial. A recent study [4], for which this work contributed, shows that the upper bound of the average state complexity of this automata is exponential but with a base only slightly above 1.

Since $\mathsf{RE}_{\cap} \subset \mathsf{RE}_{\cap,\neg}$, it is trivial that Antimirov's and Mirkin's constructions are not identical for extended regular expressions. Nevertheless, while the set of partial derivatives is ensured to be finite only if extended regular expressions are $\mathrm{CII}_{\cap}$-dissimilar, the support is finite for every $\alpha \in \mathsf{RE}_{\cap,\neg}$. Moreover, considering the natural extension, the set of $\mathrm{CII}_{\cap}$-dissimilar partial derivatives is a subset of the support.

The special regular expressions, introduced in Chapter 6, in addition to ensure the termination of Brzozowski's and partial derivative's automaton, are also more compact and allow the construction of smaller automata. This is due to the fact that, with the S-dissimilarity, more regular expressions are identified in the construction of these

derivatives based automata. When compared with the Caron et al. method, we also observed that not only we represent partial derivatives with sets of sets, but also the regular expressions themselves have a more compact representation. The implementation of these expressions and several associated algorithms can be found in the FAdo system. A preliminary version of this work has already been reported [5].

Future Research Directions

There are several lines of future research. The first is to further develop the analytical study of the average state complexity of partial derivative's automaton for extended regular expressions. These can also include the analytic combinatorial study of special regular expressions even without intersection and negation. Besides that, since alternating finite automata [28] (AFAs) allow boolean operators in the transition function, we could develop algorithms for the construction of small AFAs using derivatives.

Appendix A

Experimental Results

The following table presents the experimental comparative results of the sizes of the partial derivative automaton and the support of regular expressions with intersection. The sampling and experimental study were conducted as described in the section 4.8, using datasets with 10000 regular expressions of size n over an alphabet of k symbols.

For each n and k, the first row has the average values and the second the maximal values. The column labelled with $\emptyset$ indicates the ratio of expressions which are equivalent to the empty language. The column labelled with $|\delta_{\mathsf{pd}}|$ indicates the number of transitions of the partial derivative's automaton.

The values with "-" indicates that the computation did not end after 4 weeks of cpu time.

Table A.1: Experimental Results.

k	$\lvert\alpha\rvert$	$\lvert\alpha\rvert_{\Sigma}$	$\lvert\alpha\rvert_{\cap}$	$\emptyset$	$\lvert\delta_{\mathsf{pd}}\rvert$	$\lvert\mathcal{PD}\rvert$	$\lvert\pi\rvert$
1	25	10.1	3.04	0.08	11.39	5.87	6.44
		13	9	1	116	23	27
	50	19.9	6.30	0.09	31.45	10.65	18.13
		24	15	1	3729	151	360
	100	39.19	12.7	0.09	161.2	25.95	101.64
		47	25	1	69496	1085	9220
	150	58.65	19.18	0.1	898.18	55.50	640.77
		67	34	1	1376936	9063	845376
	200	–	–	–	–	–	–
		–	–	–	–	–	–
2	25	10.85	3.26	0.27	5.10	3.50	6.81
		13	9	1	52	11	46
	50	21.2	6.75	0.29	6.83	4.14	19.67
		25	17	1	121	29	279
	100	41.9	13.68	0.29	8.65	4.78	122.9
		48	26	1	378	64	6664
	150	62.75	20.54	0.284	10.27	5.25	663.5
		70	36	1	710	89	74737
	200	83.5	27.5	0.29	10.11	5.25	3691.15
		92	45	1	635	115	969658
5	25	11.54	3.50	0.40	3.13	2.70	7.12
		13	10	1	44	12	30
	50	22.63	7.25	0.43	3.56	2.85	20.99
		25	16	1	93	17	328
	100	44.75	14.57	0.43	3.76	2.95	134.6
		50	26	1	83	22	5760
	150	66.89	21.95	0.44	3.95	3.04	831.38
		73	39	1	141	29	72241
	200	89.02	29.33	0.44	3.99	3.05	4921.26
		97	47	1	128	29	959412

Table A.1: Experimental Results.

k	$\lvert\alpha\rvert$	$\lvert\alpha\rvert_{\Sigma}$	$\lvert\alpha\rvert_{\cap}$	$\emptyset$	$\lvert\delta_{\mathsf{pd}}\rvert$	$\lvert\mathcal{PD}\rvert$	$\lvert\pi\rvert$
10	25	11.96	3.66	0.47	2.59	2.47	7.22
		13	10	1	41	12	40
	50	23.40	7.47	0.49	2.86	2.60	21.82
		25	16	1	50	16	324
	100	46.31	15.02	0.49	3.03	2.66	144.60
		50	30	1	76	21	6336
	150	69.22	22.76	0.50	3.10	2.69	867.50
		75	39	1	92	23	60711
	200	92.13	30.37	0.50	3.03	2.67	5579.60
		99	50	1	90	32	1619575

Bibliography

[1] A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: tools for automata manipulation and visualization. In S. Maneth, editor, *14th International Conference on Implementation and Application of Automata, CIAA 2009. Proceedings*, volume 5642 of *LNCS*, pages 65–74, Sidney, July 2009. Springer.

[2] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comput. Sci.*, 155(2):291–319, 1996.

[3] V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions. In G. Rozenberg and A. Salomaa, editors, *Developments in Language Theory*, pages 195 – 209. World Scientific, 1994.

[4] R. Bastos, S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the state complexity of partial derivative automata for regular expressions with intersection. Submited.

[5] R. Bastos, N. Moreira, and R. Reis. Manipulation of extended regular expressions with derivatives. Technical Report DCC-2013-11, DCC-FC, Universidade do Porto, September 2013.

[6] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. The average transition complexity of Glushkov and partial derivative automata. In G. Mauri and A. Leporati, editors, *15th International Conference on Developments in Language Theory, DLT 2011. Proceedings*, volume 6795 of *LNCS*, pages 93–104, Milano, Italy, July 2011. Springer.

[7] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the average state complexity of partial derivative automata. *International Journal of Foundations of Computer Science*, 22(7):1593–1606, 2011.

[8] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the average size of Glushkov and partial derivative automata. *International Journal of Foundations of Computer Science*, 23(5):969–984, 2012.

[9] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoret. Comput. Sci.*, 48:197–213, 1993.

[10] J. A. Brzozowski. Derivatives of regular expressions. *JACM*, 11(4):481–494, October 1964.

[11] Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Partial derivatives of an extended regular expression. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 6638 of *Lecture Notes in Computer Science*, pages 179–191. Springer Berlin Heidelberg, 2011.

[12] J. M. Champarnaud and D. Ziadi. From Mirkin's prebases to Antimirov's word partial derivatives. *Fundam. Inform.*, 45(3):195–205, 2001.

[13] J. H. Conway. *Regular algebra and finite machines.* William Clowes& Sons, Great Britain, 1971.

[14] Project FAdo. FAdo: tools for formal languages manipulation. `http://fado.dcc.fc.up.pt/`, Access date:1.09.2015.

[15] W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theor. Comput. Sci.*, 411(31-33):2987–2998, June 2010.

[16] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. *CoRR*, abs/0802.2869, 2008.

[17] V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.

[18] J. E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[19] D. C. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, 05 1994.

[20] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Automated Reasoning*, 49:95–106, 2012. published online March 2011.

[21] M.V. Lawson. *Finite Automata*. Taylor & Francis, 2003.

[22] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[23] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–47, 1960.

[24] B. G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5:51—57, 1966.

[25] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the Association for Computing Machinery*, 13(1):158–169, 1966.

[26] A. Salomaa and V. Tixier. Two complete axiom systems for the extended language of regular expressions. *IEEE Transactions on Computers*, 17(7):700–701, 1968.

[27] K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):410–422, 1968.

[28] S. Yu. Regular languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1997.