

Co-inductive Axiomatizations of equality of regular expressions

Lasse Nielsen & Michael Nissen

15th February 2007

Contents

1	Introduction	2
1.1	Prerequisites	2
1.2	Overview	2
2	Regular expressions	3
2.1	Semantics of regular expressions	3
2.2	Semantical equality	6
2.3	Empty word property	6
3	Differentiation of regular expressions	9
3.1	Definition of differentiation (derivative)	9
3.2	Simple properties of the differentiation map	11
3.3	Language of a derivate	14
4	Bi-simulations	18
5	The Axiomatization	20
6	Soundness	22
6.1	Level stratified interpretation	22
6.2	The axiomatization is sound	23
7	Completeness	25
7.1	The problem with Grabmeyers axiomatization	25
7.2	The equality bi-simulation	26
7.3	There are only finitely many derivatives	27
7.4	Completeness theorem	33
8	Proof searching	35
8.1	Algorithm	35
8.2	Termination	35
8.3	Correctness	36
9	Conclusion	38
9.1	What has been solved	38
9.2	Future work	38
A	Program	39
A.1	The implementation	39
A.2	The test results	43
B	Literature	44

1 Introduction

In this project we will develop a sound and complete inference system to prove *equality* of regular expressions.

The classical systems for proving equality of regular expressions, is based on a set of axioms, together with a few inference rules for substituting into contexts and for solving recursive equations. Typically there are around ten to fifteen axioms, and furthermore there is implicitly assumed that we have axioms for reflexivity, symmetry and transitivity (see [7] and [5]), even though they are not explicitly stated.

The axiomatization of equality in this project is *co-inductive*, which is suggested in [4, p. 328-332]. Unfortunately the axiomatization in [4] is not complete, because there are some mix-ups between syntax and semantics.

The *co-inductive* axiomatization of equality of regular expressions only contains one axiom and one inference rule, and it does not have rules for reflexivity, symmetry and transitivity (of course they are derivable).

1.1 Prerequisites

In order to understand the proofs and theoretic results you need to be familiar with

- Regular expressions.
- Mathematics corresponding to a bachelors degree (especially induction and logic is required).
- That you are familiar with the concept of co-induction (see [8] chapter 21)

Furthermore it is advised to read [1], [2] and [3] in order to see other examples on *co-inductive* axiomatizations, because in this project we will not explicitly explain why the axiomatization is *co-inductive*, because an explanation already exists in [2] section 5.

1.2 Overview

In Section 2 we define *regular expressions* together with their semantics, which leads to the definition of semantically equal regular expressions.

Section 3 introduces *differentiation* of regular expressions, which maps sets of regular expressions to sets of regular expressions. In Section 4 we define the notion of *bi-simulation* on regular expressions. The definition of bi-simulation uses the differentiation map.

The axiomatization of equality of regular expressions is defined in Section 5, and this definition is constructed from the definition of *bi-simulation*. Section 6 and Section 7 proves that this axiomatization is sound and complete.

2 Regular expressions

Definition 2.1 Alphabet

An alphabet Σ is a finite set of symbols (letters).

All the definitions and theorems in this project should be read as, if they were quantified over all alphabets Σ . Ie. all the definitions and theorems should be over all alphabets, but in order to save space we will not explicitly write this. So throughout this document Σ will denote any alphabet.

We can now define the set of regular expressions.

Definition 2.2 Regular expressions

Regular expressions are written in the grammar:

$$E := 0 \mid 1 \mid a \mid E + F \mid EF \mid E^* \quad \text{where } a \in \Sigma$$

The set of regular expressions is written as $\mathbf{Reg}(\Sigma)$, and the set of finite subsets of $\mathbf{Reg}(\Sigma)$ is denoted by $\mathbf{Regs}(\Sigma)$.

Note that the definition of regular expressions says that regular expressions have tree like structure. We will write regular expressions in linear form, and we will use parenthesis to make the structure of the expressions clear. Furthermore we will assume that $*$ binds tighter than product, which binds tighter than sum, and that the product and sum are right associative.

Let us look at some examples of regular expressions.

Example 2.3 Regular expressions.

We assume that $a, b \in \Sigma$

- $a(ab)^*b$ is the same as $a((ab)^*b)$.
- $ab + ba + aa1$ is the same as $(ab) + ((ba) + (a(a1)))$

2.1 Semantics of regular expressions

Before we can introduce the semantics of regular expressions we need to define concatenation of regular expressions on sets of regular expressions.

Definition 2.4 Concatenation

Concatenation of regular expressions on sets is defined as

$$\begin{aligned} 1A &= A \\ 0A &= \emptyset \\ EA &= \{EF \mid F \in A \wedge F \neq 0, 1\} \cup \{E \mid 1 \in A\} \end{aligned}$$

where $A \in \mathbf{Regs}(\Sigma)$, $E \in \mathbf{Reg}(\Sigma)$ and $E \neq 0, 1$.

The definitions for $A1, A0$ and AE are symmetric.

Note that EA is a set of regular expressions, because the product construction is in regular expressions, ie. concatenation is a mapping from a set of regular expressions and a regular expression to a set of regular expressions.

We now introduce the semantics of regular expressions.

Definition 2.5 *Semantics of regular expressions*

The semantics (or language) of a regular expression is given by the map

$$\begin{aligned} \overline{\mathcal{L}}_\Sigma : \mathbf{Reg}(\Sigma) &\rightarrow \mathcal{P}(\Sigma^*) \\ \overline{\mathcal{L}}_\Sigma(0) &= \emptyset \\ \overline{\mathcal{L}}_\Sigma(1) &= \{\epsilon\} \\ \overline{\mathcal{L}}_\Sigma(a) &= \{a\}, \quad \forall a \in \Sigma \\ \overline{\mathcal{L}}_\Sigma(E + F) &= \overline{\mathcal{L}}_\Sigma(E) \cup \overline{\mathcal{L}}_\Sigma(F) \\ \overline{\mathcal{L}}_\Sigma(EF) &= \overline{\mathcal{L}}_\Sigma(E)\overline{\mathcal{L}}_\Sigma(F) \\ \overline{\mathcal{L}}_\Sigma(E^*) &= \overline{\mathcal{L}}_\Sigma(E)^* \end{aligned}$$

where the concatenation of two languages L_1 and L_2 is defined as

$$L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$$

and where

$$\begin{aligned} L^* &= \bigcup_{i=0}^{\infty} L^i & L^0 &= \{\epsilon\} \\ & & L^{i+1} &= LL^i \end{aligned}$$

Furthermore we define the semantics of a set of regular expressions by the map

$$\mathcal{L}_\Sigma : \mathbf{Regs}(\Sigma) \rightarrow \mathcal{P}(\Sigma^*)$$

which is given by

$$\mathcal{L}_\Sigma(A) = \bigcup_{E \in A} \overline{\mathcal{L}}_\Sigma(E)$$

for all $A \in \mathbf{Regs}(\Sigma)$.

The reason that we introduce the semantics of a set of regular expressions is, because later we will introduce a differentiation operator on sets of regular expressions, and then this definition of semantics will become handy.

Example 2.6 *Semantics of regular expressions*

We will use the same regular expressions as in Example 2.3

$$\begin{aligned} \overline{\mathcal{L}}_\Sigma(ab + ba + aa1) &= \overline{\mathcal{L}}_\Sigma(ab) \cup \overline{\mathcal{L}}_\Sigma(ba + aa1) \\ &= \overline{\mathcal{L}}_\Sigma(a)\overline{\mathcal{L}}_\Sigma(b) \cup \overline{\mathcal{L}}_\Sigma(b)\overline{\mathcal{L}}_\Sigma(a) \cup \overline{\mathcal{L}}_\Sigma(a)\overline{\mathcal{L}}_\Sigma(a)\overline{\mathcal{L}}_\Sigma(\epsilon) \\ &= \{a\}\{b\} \cup \{b\}\{a\} \cup \{a\}\{a\}\{\epsilon\} \\ &= \{ab\} \cup \{ba\} \cup \{aa\} \\ &= \{ab, ba, aa\} \\ \overline{\mathcal{L}}_\Sigma(a(ba)^*b) &= \{a\}(\overline{\mathcal{L}}_\Sigma(ba))^*\{b\} \\ &= \{a\} \bigcup_{i=0}^{\infty} \{ba\}^i \{b\} \\ &= \{ab, abab, ababab, \dots\} \end{aligned}$$

Before we continue, we need to prove some simple properties of the semantical map, which will be used later.

Lemma 2.7 *For all $E \in \mathbf{Reg}(\Sigma)$ and for all $A \in \mathbf{Regs}(\Sigma)$ the following properties hold*

- 1 $\mathcal{L}_\Sigma(AE) = \mathcal{L}_\Sigma(A)\mathcal{L}_\Sigma(\{E\})$ and $\mathcal{L}_\Sigma(EA) = \mathcal{L}_\Sigma(\{E\})\mathcal{L}_\Sigma(A)$
- 2 \mathcal{L}_Σ is distributive, i.e. $\mathcal{L}_\Sigma(\bigcup_{i \in I} A_i) = \bigcup_{i \in I} (\mathcal{L}_\Sigma(A_i))$
- 3 $\mathcal{L}_\Sigma(\{E^*\}) = \mathcal{L}_\Sigma(\{E\})^*$

Proof:

ad 1: We make the following rewritings

$$\mathcal{L}_\Sigma(AE) = \bigcup_{F \in AE} \overline{\mathcal{L}_\Sigma}(F) \quad (1)$$

$$= \bigcup_{F \in \{F'E \mid F' \in A \wedge F' \neq 0, 1\} \cup \{E \mid 1 \in A\}} \overline{\mathcal{L}_\Sigma}(F) \quad (2)$$

$$= \bigcup_{F \in \{F'E \mid F' \in A \wedge F' \neq 0\}} \overline{\mathcal{L}_\Sigma}(F) \quad (3)$$

$$= \bigcup_{F \in \{F'E \mid F' \in A\}} \overline{\mathcal{L}_\Sigma}(F) \quad (4)$$

$$= \bigcup_{F' \in A} \overline{\mathcal{L}_\Sigma}(F'E) \quad (5)$$

$$= \bigcup_{F' \in A} \overline{\mathcal{L}_\Sigma}(F')\overline{\mathcal{L}_\Sigma}(E) \quad (6)$$

$$= \left(\bigcup_{F' \in A} \overline{\mathcal{L}_\Sigma}(F') \right) \overline{\mathcal{L}_\Sigma}(E) \quad (7)$$

$$= \mathcal{L}_\Sigma(A)\mathcal{L}_\Sigma(\{E\}) \quad (8)$$

(3): Follows since $\overline{\mathcal{L}_\Sigma}(1E) = \overline{\mathcal{L}_\Sigma}(E)$.

(4): Follows since $\overline{\mathcal{L}_\Sigma}(0E) = \emptyset$.

(7): Clearly concatenation of languages is distributive over set union.

Correspondingly for the symmetric case.

This completes the first part of the proof.

ad 2: Since

$$\begin{aligned} \mathcal{L}_\Sigma\left(\bigcup_{i \in I} A_i\right) &= \bigcup_{E \in \bigcup_{i \in I} A_i} \overline{\mathcal{L}_\Sigma}(E) \\ &= \bigcup_{i \in I} \left(\bigcup_{E \in A_i} \overline{\mathcal{L}_\Sigma}(E) \right) \\ &= \bigcup_{i \in I} \mathcal{L}_\Sigma(A_i) \end{aligned}$$

we are done.

ad 3: By definition we get that

$$\begin{aligned} \mathcal{L}_\Sigma(\{E^*\}) &= \overline{\mathcal{L}_\Sigma(E^*)} \\ &= \overline{\mathcal{L}_\Sigma(E)^*} \\ &= \mathcal{L}_\Sigma(\{E\})^* \end{aligned}$$

which completes the proof. ■

2.2 Semantical equality

We are now ready to state, what we mean, when we talk about *equality* of regular expressions. Like in the definition of the language of a regular expression, we need to have a semantic entailment that works on sets of regular expressions. The definition below says, not surprisingly, that we consider two sets of regular expressions as semantically equal, if they have the same language.

Definition 2.8 *Equality of sets of regular expressions.*

Assume that $A, B \in \mathbf{Regs}(\Sigma)$ then

$$\models A = B \quad \Leftrightarrow \quad \mathcal{L}_\Sigma(A) = \mathcal{L}_\Sigma(B)$$

Note that this definition also enables us to check, whether regular expressions E and F have the same language (check $\models \{E\} = \{F\}$).

2.3 Empty word property

Before we can define *differentiation* on regular expressions, we need to introduce a map, that on the syntactical structure of a regular expression can determine, whether or not ϵ is in the language of the expression (and of course this again needs to be generalized to sets of regular expressions). The following definition introduces this map, and the lemma after the definition proves that the map has the desired property.

Definition 2.9 *Empty word property (e.w.p)*

We now define the maps

$$\begin{aligned} \bar{o} &: \mathbf{Reg}(\Sigma) \rightarrow \{0, 1\} \\ o &: \mathbf{Regs}(\Sigma) \rightarrow \{0, 1\} \end{aligned}$$

where 0 and 1 denotes the regular expressions 0 and 1. The definitions are

$$\begin{aligned} \bar{o}(0) &= 0 \\ \bar{o}(1) &= 1 \\ \bar{o}(E + F) &= \begin{cases} 1 & \bar{o}(E) = 1 \vee \bar{o}(F) = 1 \\ 0 & \text{otherwise} \end{cases} \\ \bar{o}(EF) &= \begin{cases} 1 & \bar{o}(E) = \bar{o}(F) = 1 \\ 0 & \text{otherwise} \end{cases} \\ \bar{o}(E^*) &= 1 \\ o(A) &= \begin{cases} 1 & \exists E \in A. \bar{o}(E) = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

If $o(A) = 1$ then the set of regular expressions A is said to have the empty word property, and if $\bar{o}(E) = 1$ then the regular expressions E is said to have the empty word property.

Note that as long as we only operate on finite sets of regular expressions ($\mathbf{Reg}(\Sigma)$), then the empty word property is clearly decidable.

Example 2.10 *Example of empty word property.*

We expect that $1a^* + ab$ has the empty word property whereas $ab + 1a$ has not.

- Since $\bar{o}(1) = \bar{o}(a^*) = 1$ we get that $\bar{o}(1a^*) = 1$, and hence $\bar{o}(1a^* + ab) = 1$.
- Since $\bar{o}(ab) = \bar{o}(1a) = 0$ we get that $\bar{o}(1a + ab) = 0$

We now prove that \bar{o} has the desired property.

Lemma 2.11 *For all $E \in \mathbf{Reg}(\Sigma)$ then*

$$\bar{o}(E) = 1 \quad \Leftrightarrow \quad \epsilon \in \overline{\mathcal{L}_\Sigma}(E)$$

Proof:

“ \Rightarrow ”: By structural induction on E .

$E = 0, a$: Since $\bar{o}(0) = \bar{o}(a) = 0$ the implication trivially holds.

$E = 1$: Since $\overline{\mathcal{L}_\Sigma}(1) = \{\epsilon\}$ the implication trivially holds.

$E = E' + F'$: Assume $\bar{o}(E' + F') = 1$, ie. $\bar{o}(E') = 1 \vee \bar{o}(F') = 1$. Assume $\bar{o}(E') = 1$. By induction we get that $\epsilon \in \overline{\mathcal{L}_\Sigma}(E')$, and hence

$$\epsilon \in \overline{\mathcal{L}_\Sigma}(E') \cup \overline{\mathcal{L}_\Sigma}(F') = \overline{\mathcal{L}_\Sigma}(E' + F')$$

The case for $\bar{o}(F') = 1$ is symmetric.

$E = E'F'$: Assume $\bar{o}(E'F') = 1$, ie. $\bar{o}(E') = \bar{o}(F') = 1$. By induction we get that $\epsilon \in \overline{\mathcal{L}_\Sigma}(E') \wedge \epsilon \in \overline{\mathcal{L}_\Sigma}(F')$, and hence

$$\epsilon = \epsilon\epsilon \in \overline{\mathcal{L}_\Sigma}(E')\overline{\mathcal{L}_\Sigma}(F') = \overline{\mathcal{L}_\Sigma}(E'F')$$

$E = E'^*$. By definition we get that

$$\begin{aligned} \epsilon &\in \{\epsilon\} \\ &= \overline{\mathcal{L}_\Sigma}(E')^0 \\ &\subseteq \bigcup_{i=0}^{\infty} \overline{\mathcal{L}_\Sigma}(E')^i \\ &= \overline{\mathcal{L}_\Sigma}(E')^* \\ &= \overline{\mathcal{L}_\Sigma}(E'^*) \end{aligned}$$

which completes the first part of the proof.

“ \Leftarrow ”: We show that

$$\bar{o}(E) \neq 1 \Rightarrow \epsilon \notin \overline{\mathcal{L}_\Sigma}(E)$$

which is equivalent to (since $\bar{o}(E) \in \{0, 1\}$)

$$\bar{o}(E) = 0 \Rightarrow \epsilon \notin \overline{\mathcal{L}_\Sigma}(E) \tag{9}$$

We prove (9) by structural induction on E .

$E = 0$: Ok, since $\epsilon \notin \emptyset = \overline{\mathcal{L}_\Sigma}(0)$.

$E = 1$: Trivially holds because $\overline{\mathcal{L}_\Sigma}(1) = 1$.

$E = a$: Ok, since $\epsilon \notin \{a\} = \overline{\mathcal{L}_\Sigma}(a)$.

$E = E' + F'$: Assume $\overline{\mathcal{L}_\Sigma}(E' + F') = 0$, ie. $\overline{\mathcal{L}_\Sigma}(E') = \overline{\mathcal{L}_\Sigma}(F') = 0$. By induction we get that $\epsilon \notin \overline{\mathcal{L}_\Sigma}(E') \wedge \epsilon \notin \overline{\mathcal{L}_\Sigma}(F')$. Hence

$$\epsilon \notin \overline{\mathcal{L}_\Sigma}(E') \cup \overline{\mathcal{L}_\Sigma}(F') = \overline{\mathcal{L}_\Sigma}(E' + F')$$

$E = E'F'$: Assume $\overline{\mathcal{L}_\Sigma}(E'F') = 0$. Hence $\overline{\mathcal{L}_\Sigma}(E') = 0 \vee \overline{\mathcal{L}_\Sigma}(F') = 0$. Assume $\overline{\mathcal{L}_\Sigma}(E') = 0$. By induction we get that $\epsilon \notin \overline{\mathcal{L}_\Sigma}(E')$, and hence

$$\epsilon \notin \{w_1w_2 \mid w_1 \in \overline{\mathcal{L}_\Sigma}(E') \wedge w_2 \in \overline{\mathcal{L}_\Sigma}(F')\} = \overline{\mathcal{L}_\Sigma}(E'F')$$

The case for $\overline{\mathcal{L}_\Sigma}(F') = 0$ is symmetric.

$E = E'^*$: Since $\overline{\mathcal{L}_\Sigma}(E'^*) = 1$ there is nothing to show.

This completes the proof. ■

Corollary 2.12 *For all $A \in \mathbf{Regs}(\Sigma)$*

$$o(A) = 1 \quad \Leftrightarrow \quad \epsilon \in \mathcal{L}_\Sigma(A)$$

Proof: Assume $A \in \mathbf{Regs}(\Sigma)$.

“ \Rightarrow ”:
Assume $o(A) = 1$, ie. there exists $E \in A$ such that $\overline{\mathcal{L}_\Sigma}(E) = 1$. By Lemma 2.11 we get that $\epsilon \in \overline{\mathcal{L}_\Sigma}(E)$, and hence

$$\epsilon \in \overline{\mathcal{L}_\Sigma}(E) \subseteq \bigcup_{E \in A} \overline{\mathcal{L}_\Sigma}(E) = \mathcal{L}_\Sigma(A)$$

which completes the first part of the proof.

“ \Leftarrow ”:
Assume $\epsilon \in \mathcal{L}_\Sigma(A) = \bigcup_{E \in A} \overline{\mathcal{L}_\Sigma}(E)$. Hence there exists $E \in A$ such that $\epsilon \in \overline{\mathcal{L}_\Sigma}(E)$. By Lemma 2.11 we get that $\overline{\mathcal{L}_\Sigma}(E) = 1$, and hence $o(A) = 1$, which completes the proof. ■

3 Differentiation of regular expressions

Now we are ready to introduce the *derivative* of a regular expression with respect to some symbol in the alphabet.

Say we have $a \in \Sigma$. The intuition behind the derivative is that the derivative of a regular expression E with respect to a , is a set of regular expressions, where the language of the *derivative*, is all the sequences starting with a from the language of E , but where a is removed. The formal definition can be seen below.

3.1 Definition of differentiation (derivative)

Definition 3.1 *Differentiation (derivative)*

We define the differentiation of a set of regular expressions with respect to a (where $a \in \Sigma$) by $\overline{D}_a : \mathbf{Reg}(\Sigma) \rightarrow \mathbf{Regs}(\Sigma)$ and $D_a : \mathbf{Regs}(\Sigma) \rightarrow \mathbf{Regs}(\Sigma)$

$$\begin{aligned} \overline{D}_a(0) &= \emptyset \\ \overline{D}_a(1) &= \emptyset \\ \overline{D}_a(b) &= \begin{cases} \{1\} & a = b \\ \emptyset & a \neq b \end{cases} \\ \overline{D}_a(E + F) &= \overline{D}_a(E) \cup \overline{D}_a(F) \\ \overline{D}_a(EF) &= \overline{D}_a(E) F \cup \overline{\sigma}(E) \overline{D}_a(F) \\ \overline{D}_a(E^*) &= \overline{D}_a(E) E^* \\ \\ D_a(A) &= \bigcup_{E \in A} \overline{D}_a(E) \end{aligned}$$

In order to ensure that the definition of the derivative maps makes sense we need to prove the following lemmas.

Lemma 3.2

$$\forall a \in \Sigma \forall E \in \mathbf{Reg}(\Sigma). \quad \overline{D}_a(E) \in \mathbf{Regs}(\Sigma)$$

Proof:

Assume $a \in \Sigma$. The proof is made by structural induction on E .

$E = 0, 1, b$: Trivial.

$E = E' + F'$: By induction we get that $\overline{D}_a(E')$ and $\overline{D}_a(F')$ are finite. Hence

$$\overline{D}_a(E' + F') = \overline{D}_a(E') \cup \overline{D}_a(F') \in \mathbf{Regs}(\Sigma)$$

which completes this case.

$E = E'F'$: By induction $\overline{D}_a(E')$ and $\overline{D}_a(F')$ are finite. Hence

$$\overline{D}_a(E'F') = \overline{D}_a(E') F' \cup \overline{\sigma}(E') \overline{D}_a(F') \in \mathbf{Regs}(\Sigma)$$

$E = E'^*$: By induction $\overline{D}_a(E')$ is finite. Hence

$$\overline{D}_a(E'^*) = \overline{D}_a(E') E'^* \in \mathbf{Regs}(\Sigma)$$

which completes the proof. ■

We now extend the notation for differentiation, such that we can differentiate with respect to a sequence of letters in our alphabet.

Definition 3.3 *Differentiation with respect to a sequence.*

Assume $\pi \in \Sigma^+$ and $a \in \Sigma$ and let πa denote the concatenation of the two. We now define

$$D_{\pi a} = D_a \circ D_\pi$$

ie. $D_\pi : \mathbf{Regs}(\Sigma) \rightarrow \mathbf{Regs}(\Sigma)$ for any $\pi \in \Sigma^+$.

Lemma 3.4

$$\forall \pi \in \Sigma^+ \forall A \in \mathbf{Regs}(\Sigma). \quad D_\pi(A) \in \mathbf{Regs}(\Sigma)$$

Proof:

Induction on $|\pi|$ (where $|\pi|$ denotes the number of symbols in π).

$|\pi| = 1$. Assume $A \in \mathbf{Regs}(\Sigma)$. Since $|\pi| = 1$ we get that $\pi = a$ for some $a \in \Sigma$. Hence

$$\begin{aligned} D_\pi(A) &= D_a(A) \\ &= \bigcup_{E \in A} \overline{D}_a(E) \end{aligned}$$

By Lemma 3.2 we get that $\overline{D}_a(E)$ is finite for all E . Since A is finite we have a finite union of finite sets, which then must be finite. This completes the first case.

$|\pi| > 1$: Hence $\pi = \sigma a$ for some $a \in \Sigma$, where $1 \leq |\sigma| < |\pi|$. Since

$$\begin{aligned} D_\pi(A) &= D_{\sigma a}(A) \\ &= D_a(D_\sigma(A)) \end{aligned}$$

By induction we get that $D_\sigma(A) \in \mathbf{Regs}(\Sigma)$, and by the basic case in this lemma, we get that $D_a(D_\sigma(A)) \in \mathbf{Regs}(\Sigma)$. ■

The definition of differentiation of regular expressions is different from the definition, that is given in [5], and which is used in the coinductive axiomatization in [4]. But the semantics of the results when differentiating with our definition of the derivative and differentiating with the definition from [5] is the same. Why we have chosen a different definition will be made clear in Section 7, which shows the completeness of the axiomatization (if we had used the definition from [5] the axiomatization would not be complete).

Let us see an example of the derivative.

Example 3.5 *We now differentiate $a(ba)^*b$ with respect to a and b , and see what the languages of the results are.*

$$\begin{aligned} D_a(\{a(ba)^*b\}) &= \overline{D}_a(a(ba)^*b) \\ &= \overline{D}_a(a)(ba)^*b \cup \overline{\partial}(a)\overline{D}_a((ba)^*b) \\ &= \{1\}(ba)^*b \cup 0\overline{D}_a((ba)^*b) \\ &= \{(ba)^*b\} \end{aligned}$$

$$\begin{aligned}
D_b(\{a(ba)^*b\}) &= \overline{D}_b(a)(ba)^*b \cup \overline{o}(a)\overline{D}_b((ba)^*b) \\
&= \emptyset(ba)^*b \cup \emptyset\overline{D}_b((ba)^*b) \\
&= \emptyset
\end{aligned}$$

The languages of the derivatives are

$$\begin{aligned}
\mathcal{L}_\Sigma(D_a(a(ba)^*b)) &= \overline{\mathcal{L}}_\Sigma(ba)^*b \\
&= \left(\bigcup_{i=0}^{\infty} ba^i\right)\{b\} \\
&= \{b, bab, babab, \dots\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}_\Sigma(D_b(a(ba)^*b)) &= \mathcal{L}_\Sigma(\emptyset) \\
&= \emptyset
\end{aligned}$$

Notice that the derivate has the properties that we want in this case, ie. that the derivate of $a(ba)^*b$ is a set of regular expressions A , where the language of A is the language of $a(ba)^*b$, where we only have the sequences starting with a , and with a removed.

In the second case, we see that we get an empty set of regular expressions, ie. the language is empty. This is not surprising, because we see that there is no sequence in the language of $a(ba)^*b$ that starts with b .

Later we will show that this property holds in general.

3.2 Simple properties of the differentiation map

In order to prove that the derivative behaves like we want, we first need to prove some simple properties about the differentiation map.

Lemma 3.6 For all $a \in \Sigma$, for all $E \in \mathbf{Reg}(\Sigma)$ and for all $A, B \in \mathbf{Regs}(\Sigma)$ the following properties hold

- 1 D_a is distributive, ie. $D_a(\bigcup_{i \in I} A_i) = \bigcup_{i \in I} D_a(A_i)$
- 2 D_a is monotone, ie. $A \subseteq B \Rightarrow D_a(A) \subseteq D_a(B)$
- 3 $D_a(AE) = D_a(A)E \cup o(A)D_a(\{E\})$ and $D_a(EA) = D_a(\{E\})A \cup o(\{E\})D_a(A)$.

Proof: The first two things we need to prove are fairly obvious.

ad 1: We make the following rewritings

$$\begin{aligned}
D_a\left(\bigcup_{i \in I} A_i\right) &= \bigcup_{E \in \bigcup_{i \in I} A_i} \overline{D}_a(E) \\
&= \bigcup_{i \in I} \bigcup_{E \in A_i} \overline{D}_a(E) \\
&= \bigcup_{i \in I} D_a(A_i)
\end{aligned}$$

which completes the first part of the proof.

ad 2: Assume $A \subseteq B$ then

$$D_a(A) = \bigcup_{E \in A} \overline{D}_a(E) \quad (10)$$

$$\subseteq \bigcup_{E \in B} \overline{D}_a(E) \quad (11)$$

$$= D_a(B) \quad (12)$$

(11): Since $A \subseteq B$.

This completes this case.

ad 3: We split the proof depending on E .

$E = 0$: Since $D_a(A0) = D_a(A)0 = o(A)D_a(\{0\}) = \emptyset$ we see that both sides of the equation is the empty set.

$E = 1$: Since $A1 = A$, $D_a(A)1 = D_a(A)$ and $D_a(1) = \emptyset$ we see that both sides of the equation yields $D_a(A)$.

$E \neq 0, 1$: Since A is a set we can write $A = \bigcup_{i \in I} \{E_i\}$ for some index set I . We now make

the following rewritings:

$$D_a(AE) = D_a\left(\left(\bigcup_{i \in I} \{E_i\}\right) E\right) \quad (13)$$

$$= D_a\left(\bigcup_{i \in I} (\{E_i\} E)\right) \quad (14)$$

$$= D_a\left(\left(\bigcup_{i \in I \wedge E_i \neq 0,1} \{E_i E\}\right) \cup \{E \mid \exists i \in I. E_i = 1\}\right) \quad (15)$$

$$= \left(\bigcup_{i \in I \wedge E_i \neq 0,1} \overline{D}_a(E_i E)\right) \cup \{\overline{D}_a(E) \mid \exists i \in I. E_i = 1\} \quad (16)$$

$$= \left(\bigcup_{i \in I \wedge E_i \neq 0,1} \overline{D}_a(E_i) E \cup \overline{\sigma}(E_i) \overline{D}_a(E)\right) \cup \{\overline{D}_a(E) \mid \exists i \in I. E_i = 1\} \quad (17)$$

$$= \left(\bigcup_{i \in I \wedge E_i \neq 0,1} \overline{D}_a(E_i) E\right) \cup \left(\bigcup_{i \in I \wedge E_i \neq 0,1} \overline{\sigma}(E_i) \overline{D}_a(E)\right) \cup \{\overline{D}_a(E) \mid \exists i \in I. E_i = 1\} \quad (18)$$

$$= \left(\bigcup_{i \in I} \overline{D}_a(E_i) E\right) \cup \left(\bigcup_{i \in I \wedge E_i \neq 1} \overline{\sigma}(E_i) \overline{D}_a(E)\right) \cup \{\overline{D}_a(E) \mid \exists i \in I. E_i = 1\} \quad (19)$$

$$= \left(\bigcup_{i \in I} \overline{D}_a(E_i) E\right) \cup \left(\bigcup_{i \in I} \overline{\sigma}(E_i) \overline{D}_a(E)\right) \quad (20)$$

$$= \left(\bigcup_{i \in I} \overline{D}_a(E_i) E\right) \cup o(A) \overline{D}_a(E) \quad (21)$$

$$= \left(\bigcup_{i \in I} \overline{D}_a(E_i)\right) E \cup o(A) \overline{D}_a(E) \quad (22)$$

$$= D_a\left(\bigcup_{i \in I} \{E_i\}\right) E \cup o(A) \overline{D}_a(E) \quad (23)$$

$$= D_a(A) E \cup o(A) D_a(\{E\}) \quad (24)$$

(14): Clearly concatenation of regular expressions on sets is distributive over set union.

(15): Follows from the definition of concatenation of elements different from 0 and 1 on a set (we have assumed that $E \neq 0, 1$).

(16): By ad 1, and by the definition of D_a .

(17): By the definition of \overline{D}_a .

(19): Since $D_a(0) E = D_a(1) E = \emptyset$ we can remove the condition that $E_i \neq 0, 1$, and since $\overline{\sigma}(0) \overline{D}_a(E) = \emptyset$ we can remove the condition $E_i \neq 0$ in the second union.

(20): If there exists $i \in I$ such that $E_i = 1$ then

$$\begin{aligned}
\bigcup_{i \in I} \bar{o}(E_i) \bar{D}_a(E) &= \bigcup_{i \in I \wedge E_i \neq 1} \bar{o}(E_i) \bar{D}_a(E) \cup \bar{o}(1) \bar{D}_a(E) \\
&= \bigcup_{i \in I \wedge E_i \neq 1} \bar{o}(E_i) \bar{D}_a(E) \cup \bar{D}_a(E) \\
&= \bigcup_{i \in I \wedge E_i \neq 1} \bar{o}(E_i) \bar{D}_a(E) \cup \{\bar{D}_a(E) \mid \exists i \in I. E_i = 1\}
\end{aligned}$$

If there does not exist $i \in I$ such that $E_i = 1$ then the equation trivially holds.

(21): Since $o(E_i) \in \{0, 1\}$ and since $0D_a(E) = \emptyset$ and $1D_a(E) = D_a(E)$ we get that $(\bigcup_{i \in I} \bar{o}(E_i) \bar{D}_a(E)) = o(A) \bar{D}_a(E)$.

(23): Follows from ad 1, and the definition of D_a .

The symmetric case can be proved similarly.

This completes the proof of the theorem. ■

3.3 Language of a derivate

We are now ready to prove that the differentiation map satisfies the property we inspected in Example 3.5

Theorem 3.7 *For all $E \in \mathbf{Reg}(\Sigma)$ and for all $a \in \Sigma$ then*

$$\mathcal{L}_\Sigma(D_a(\{E\})) = \{w \mid aw \in \mathcal{L}_\Sigma(\{E\})\}$$

Proof:

By structural induction on E .

$E = 0$: In this case we get that

$$\begin{aligned}
\mathcal{L}_\Sigma(D_a(\{0\})) &= \mathcal{L}_\Sigma(\emptyset) \\
&= \emptyset \\
&= \{w \mid aw \in \emptyset\} \\
&= \{w \mid aw \in \mathcal{L}_\Sigma(\{0\})\}
\end{aligned}$$

which completes this case.

$E = 1$: In this case we get that

$$\begin{aligned}
\mathcal{L}_\Sigma(D_a(\{1\})) &= \mathcal{L}_\Sigma(\emptyset) \\
&= \emptyset \\
&= \{w \mid aw \in \{\epsilon\}\} \\
&= \{w \mid aw \in \mathcal{L}_\Sigma(\{1\})\}
\end{aligned}$$

which completes this case.

$E = b$: We split the proof depending on b .

$a \neq b$: Hence

$$\begin{aligned}
\mathcal{L}_\Sigma(D_a(\{b\})) &= \mathcal{L}_\Sigma(\emptyset) \\
&= \emptyset \\
&= \{w \mid aw \in \{b\}\} \\
&= \{w \mid aw \in \mathcal{L}_\Sigma(\{b\})\}
\end{aligned}$$

$a = b$: Hence

$$\begin{aligned}\mathcal{L}_\Sigma(D_a(\{a\})) &= \mathcal{L}_\Sigma(\{1\}) \\ &= \{\epsilon\} \\ &= \{w \mid aw \in \{a\}\} \\ &= \{w \mid aw \in \mathcal{L}_\Sigma(\{a\})\}\end{aligned}$$

which completes this case.

$E = E' + F'$: Hence

$$\mathcal{L}_\Sigma(D_a(\{E' + F'\})) = \mathcal{L}_\Sigma(D_a(\{E'\}) \cup D_a(\{F'\})) \quad (25)$$

$$= \mathcal{L}_\Sigma(D_a(\{E'\})) \cup \mathcal{L}_\Sigma(D_a(\{F'\})) \quad (26)$$

$$= \{w \mid aw \in \mathcal{L}_\Sigma(\{E'\})\} \cup \{w \mid aw \in \mathcal{L}_\Sigma(\{F'\})\} \quad (27)$$

$$= \{w \mid aw \in \mathcal{L}_\Sigma(\{E'\} \cup \mathcal{L}_\Sigma(\{E'\}))\} \quad (28)$$

$$= \{w \mid aw \in \mathcal{L}_\Sigma(\{E' + F'\})\} \quad (29)$$

(25): Follows from the definition of the derivative.

(26): Follows from Lemma 2.7.

(27): Follows by induction.

(29): Follows from the definition of the semantics.

$E = E'F'$: Hence

$$\mathcal{L}_\Sigma(D_a(\{E'F'\})) = \mathcal{L}_\Sigma(D_a(\{E'\})F' \cup o(\{E'\})D_a(\{F'\})) \quad (30)$$

$$= \mathcal{L}_\Sigma(D_a(\{E'\})F') \cup \mathcal{L}_\Sigma(o(\{E'\})D_a(\{F'\})) \quad (31)$$

$$= \mathcal{L}_\Sigma(D_a(\{E'\}))\mathcal{L}_\Sigma(\{F'\}) \cup \mathcal{L}_\Sigma(\{o(\{E'\})\})\mathcal{L}_\Sigma(D_a(\{F'\})) \quad (32)$$

$$= \{w_1w_2 \mid w_1 \in \mathcal{L}_\Sigma(D_a(\{E'\})) \wedge w_2 \in \mathcal{L}_\Sigma(\{F'\})\} \cup \quad (33)$$

$$\{w_1w_2 \mid w_1 \in \mathcal{L}_\Sigma(\{o(\{E'\})\}) \wedge w_2 \in \mathcal{L}_\Sigma(D_a(\{F'\}))\} \quad (34)$$

$$= \{w_1w_2 \mid aw_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge w_2 \in \mathcal{L}_\Sigma(\{F'\})\} \cup \quad (35)$$

$$\{w \mid o(\{E'\}) = 1 \wedge aw \in \mathcal{L}_\Sigma(\{F'\})\} \quad (36)$$

(30): Follows from the definition of the derivative.

(31): Follows from Lemma 2.7.

(32): Follows from Lemma 2.7.

(36): Follows by induction and since $\mathcal{L}_\Sigma(\{o(\{E'\})\}) \subseteq \{\epsilon\}$, and hence by Corollary 2.12 we get that $o(\{E'\}) = 1$ if and only if $\{\epsilon\} = \mathcal{L}_\Sigma(\{o(\{E'\})\})$.

We denote the above set by H . Define

$$\begin{aligned}\overline{H} &= \{w \mid aw \in \mathcal{L}_\Sigma(\{E'F'\})\} \\ &= \{w \mid aw \in \{w_1w_2 \mid w_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge w_2 \in \mathcal{L}_\Sigma(\{F'\})\}\}\end{aligned}$$

In order to complete the proof for this case, we need to prove that $H = \overline{H}$.

$H \subseteq \overline{H}$: Assume $w \in H$. Then we must be in one of the following cases.

i $w = t_1t_2$ where $at_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge t_2 \in \mathcal{L}_\Sigma(\{F'\})$.

ii $o(\{E'\}) = 1$ and $aw \in \mathcal{L}_\Sigma(\{F'\})$

ad i: Choose $w_1 = at_1$ and $w_2 = t_2$. Hence $w = t_1t_2 \in \overline{H}$.

ad ii: Since $o(\{E'\}) = 1$ we can choose $w_1 = \epsilon \in \mathcal{L}_\Sigma(\{E'\})$ (by Corollary 2.12). Furthermore choose $w_2 = aw$. Then $w_1w_2 = aw$ and hence $w \in \overline{H}$.

$H \supseteq \overline{H}$: Assume $w \in \overline{H}$. Ie. there exists $w_1 \in \mathcal{L}_\Sigma(\{E'\})$ and $w_2 \in \mathcal{L}_\Sigma(\{F'\})$ such that $aw = w_1w_2$. We now split the proof in the following two cases.

i $w = \epsilon$

ii $w \neq \epsilon$

ad i: Since $\epsilon = w_1 \in \mathcal{L}_\Sigma(\{E'\})$ and since $w_1w_2 = aw$ we get that $o(\{E'\}) = 1$ (by Corollary 2.12) and $w_2 = aw$. Hence $w \in H$.

ad ii: Since $w_1w_2 = aw$ and since $w_1 \neq \epsilon$ we get that $w_1 = aw'$ for some $w' \in \Sigma^*$. Hence $aw = w_1w_2 = aw'w_2$ and therefore $w = w'w_2$. Since $w_1 = aw' \in \mathcal{L}_\Sigma(\{E'\})$ and $w_2 \in \mathcal{L}_\Sigma(\{F'\})$ we get that $w \in H$.

Which completes this part of the proof.

$E = E'^*$: First we make some rewritings.

$$\mathcal{L}_\Sigma(D_a(\{E'^*\})) = \mathcal{L}_\Sigma(D_a(\{E'\})E'^*) \quad (37)$$

$$= \mathcal{L}_\Sigma(D_a(\{E'\}))\mathcal{L}_\Sigma(\{E'^*\}) \quad (38)$$

$$= \{w_1w_2 \mid w_1 \in \mathcal{L}_\Sigma(D_a(\{E'\})) \wedge w_2 \in \mathcal{L}_\Sigma(\{E'^*\})\} \quad (39)$$

$$= \{w_1w_2 \mid aw_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge w_2 \in \mathcal{L}_\Sigma(\{E'^*\})\} \quad (40)$$

$$= \{w_1w_2 \mid aw_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge w_2 \in \mathcal{L}_\Sigma(\{E'\})^*\} \quad (41)$$

(40): Follows by induction.

(41): Follows from Lemma 2.7.

We denote the above set by H . Define

$$\begin{aligned} \overline{H} &= \{w \mid aw \in \mathcal{L}_\Sigma(\{E'^*\})\} \\ &= \{w \mid aw \in \mathcal{L}_\Sigma(\{E'\})^*\} \end{aligned}$$

In order to complete the proof we need to prove that $H = \overline{H}$.

$H \subseteq \overline{H}$: Assume $w = w_1w_2 \in H$. Hence $aw_1 \in \mathcal{L}_\Sigma(\{E'\})$ and $w_2 \in \mathcal{L}_\Sigma(\{E'\})^* = \cup_{i=0}^\infty \mathcal{L}_\Sigma(\{E'\})^i$, and therefore there exists $i \geq 0$ such that $w_2 \in \mathcal{L}_\Sigma(\{E'\})^i$. This means that

$$\begin{aligned} aw_1w_2 &\in \mathcal{L}_\Sigma(\{E'\})\mathcal{L}_\Sigma(\{E'\})^i \\ &= \mathcal{L}_\Sigma(\{E'\})^{i+1} \\ &\subseteq \mathcal{L}_\Sigma(\{E'\})^* \end{aligned}$$

Hence $w = w_1w_2 \in \overline{H}$.

$H \supseteq \overline{H}$: Assume that $w \in \overline{H}$. This means that $aw \in \mathcal{L}_\Sigma(\{E'\})^* = \cup_{i=0}^\infty \mathcal{L}_\Sigma(\{E'\})^i$, ie. there exists a *least* $i \geq 1$ such that

$$\begin{aligned} aw &\in \mathcal{L}_\Sigma(\{E'\})^i \\ &= \mathcal{L}_\Sigma(\{E'\})\mathcal{L}_\Sigma(\{E'\})^{i-1} \\ &= \{w_1w_2 \mid w_1 \in \mathcal{L}_\Sigma(\{E'\}) \wedge w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1}\} \end{aligned}$$

(Note that $i \geq 1$ because $\mathcal{L}_\Sigma(\{E'\})^0 = \{\epsilon\}$). We now show that

$$\exists w_1 \in \mathcal{L}_\Sigma(\{E'\}) \exists w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1} \exists w' \in \Sigma^*. \quad aw = w_1 w_2 \wedge w_1 = aw' \quad (42)$$

We prove (42) by contradiction. So assume that

$$\forall w_1 \in \mathcal{L}_\Sigma(\{E'\}) \forall w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1} \forall w' \in \Sigma^*. \quad aw \neq w_1 w_2 \vee w_1 \neq aw'$$

which is equivalent to

$$\forall w_1 \in \mathcal{L}_\Sigma(\{E'\}) \forall w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1} \forall w' \in \Sigma^*. \quad aw = w_1 w_2 \Rightarrow w_1 \neq aw' \quad (43)$$

Assume that $w_1 \in \mathcal{L}_\Sigma(\{E'\})$, $w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1}$ and $aw = w_1 w_2$. From (43) we get that $\forall w' \in \Sigma^*$. $w_1 \neq aw'$. Since $aw = w_1 w_2$ we must have $w_1 = \epsilon$, and hence $aw = w_2 \in \mathcal{L}_\Sigma(\{E'\})^{i-1}$, but this gives us a contradiction, since i was the least i such that $aw \in \mathcal{L}_\Sigma(\{E'\})^i$. This proves (42).

Hence there exist $w_1 \in \mathcal{L}_\Sigma(\{E'\})$, $w_2 \in \mathcal{L}_\Sigma(\{E'\})^i$ and $w' \in \Sigma^*$ such that $aw = w_1 w_2$ and $w_1 = aw'$. Since $w_2 \in \mathcal{L}_\Sigma(\{E'\})^i \subseteq \mathcal{L}_\Sigma(\{E'\})^*$ we get from the definition of H that $w \in H$, which completes the proof. \blacksquare

Corollary 3.8 *For all $a \in \Sigma$ and for all $A \in \mathbf{Regs}(\Sigma)$ then*

$$\mathcal{L}_\Sigma(D_a(A)) = \{w \mid aw \in \mathcal{L}_\Sigma(A)\}$$

Proof: Assume $a \in \Sigma$ and assume that $A \in \mathbf{Regs}(\Sigma)$. We can write $A = \bigcup_{i \in I} \{E_i\}$ for some index set I . Hence

$$\mathcal{L}_\Sigma(D_a(A)) = \mathcal{L}_\Sigma(D_a\left(\bigcup_{i \in I} \{E_i\}\right)) \quad (44)$$

$$= \mathcal{L}_\Sigma\left(\bigcup_{i \in I} D_a(\{E_i\})\right) \quad (45)$$

$$= \bigcup_{i \in I} \mathcal{L}_\Sigma(D_a(\{E_i\})) \quad (46)$$

$$= \bigcup_{i \in I} \{w \mid aw \in \mathcal{L}_\Sigma(\{E_i\})\} \quad (47)$$

$$= \left\{ w \mid aw \in \bigcup_{i \in I} \mathcal{L}_\Sigma(\{E_i\}) \right\} \quad (48)$$

$$= \left\{ w \mid aw \in \mathcal{L}_\Sigma\left(\bigcup_{i \in I} \{E_i\}\right) \right\} \quad (49)$$

$$= \{w \mid aw \in \mathcal{L}_\Sigma(A)\} \quad (50)$$

(45): Follows from Lemma 3.6.

(46): Follows from Lemma 2.7.

(47): Follows from Theorem 3.7. \blacksquare

4 Bi-simulations

In this section we will define bi-simulations on sets of regular expressions, and we will give an example of a bi-simulation.

Later we will prove that the set of pairs of semantically equal sets of regular expressions is a bi-simulation.

The notion of bi-simulations is widely used, and bi-simulations for regular *behaviors* are introduced in [6], but because of the differences in semantics the definition below is significantly different.

Definition 4.1 *Bi-simulation*

We define that $\mathcal{R} \subseteq \mathbf{Rregs}(\Sigma) \times \mathbf{Rregs}(\Sigma)$ is a bi-simulation if and only if the following two properties are fulfilled.

$$\begin{aligned} A_1 \mathcal{R} A_2 &\Rightarrow \forall \alpha \in \Sigma : D_\alpha(A_1) \mathcal{R} D_\alpha(A_2) \\ A_1 \mathcal{R} A_2 &\Rightarrow o(A_1) = o(A_2) \end{aligned}$$

The intention is that there is a connection between when two sets of regular terms are semantically equivalent, and when there is a bi-simulation that relates the two sets.

We will prove in the completeness section, that there is a finite bi-simulation relating two sets of regular expressions, if the two sets are semantically equivalent.

For now, we will just give an example of a finite bi-simulation relating $\{a(ba)^*\}$ and $\{ab(ab)^*\}$.

Example 4.2 We will now give a bi-simulation, which relates $\{a(ba)^*b\}$ and $\{ab(ab)^*\}$.

$$\begin{aligned} \mathcal{R} = \{ & (\{a(ba)^*b\}, \{ab(ab)^*\}), \\ & (\{(ba)^*b\}, \{b(ab)^*\}), \\ & (\{a(ba)^*b, 1\}, \{(ab)^*\}), \\ & (\{\}, \{\}) \\ & \} \end{aligned}$$

In order to justify that \mathcal{R} is a bi-simulation, we need to compute the following derivatives.

$$\begin{array}{ll} D_a(\{a(ba)^*b\}) &= \{(ba)^*b\} & D_a(\{ab(ba)^*\}) &= \{b(ab)^*\} \\ D_b(\{a(ba)^*b\}) &= \{\} & D_b(\{ab(ba)^*\}) &= \{\} \\ D_a(\{(ba)^*b\}) &= \{\} & D_a(\{b(ab)^*\}) &= \{\} \\ D_b(\{(ba)^*b\}) &= \{a(ba)^*b, 1\} & D_b(\{b(ab)^*\}) &= \{(ab)^*\} \\ D_a(\{a(ba)^*b, 1\}) &= \{(ba)^*b\} & D_a(\{(ab)^*\}) &= \{b(ab)^*\} \\ D_b(\{a(ba)^*b, 1\}) &= \{\} & D_b(\{(ab)^*\}) &= \{\} \\ D_a(\{\}) &= \{\} & D_b(\{\}) &= \{\} \end{array}$$

Using this, the first property of bi-simulations can be written as below.

$$\begin{aligned}
\{a(ba)^*b\}\mathcal{R}\{ab(ab)^*\} &\Rightarrow \{(ba)^*b\}\mathcal{R}\{b(ab)^*\} \\
\{a(ba)^*b\}\mathcal{R}\{ab(ab)^*\} &\Rightarrow \{\}\mathcal{R}\{\} \\
\{(ba)^*b\}\mathcal{R}\{b(ab)^*\} &\Rightarrow \{\}\mathcal{R}\{\} \\
\{(ba)^*b\}\mathcal{R}\{b(ab)^*\} &\Rightarrow \{a(ba)^*b, 1\}\mathcal{R}\{(ab)^*\} \\
\{a(ba)^*b, 1\}\mathcal{R}\{(ab)^*\} &\Rightarrow \{(ba)^*b\}\mathcal{R}\{b(ab)^*\} \\
\{a(ba)^*b, 1\}\mathcal{R}\{(ab)^*\} &\Rightarrow \{\}\mathcal{R}\{\} \\
\{\}\mathcal{R}\{\} &\Rightarrow \{\}\mathcal{R}\{\}
\end{aligned}$$

We can see, that the first property is fulfilled. Now we only need to check the second property of bi-simulations, and this is done below.

$$\begin{aligned}
o(\{a(ba)^*b\}) &= 0 = o(\{ab(ab)^*\}) \\
o(\{(ba)^*b\}) &= 0 = o(\{b(ab)^*\}) \\
o(\{a(ba)^*b, 1\}) &= 1 = o(\{(ab)^*\}) \\
o(\{\}) &= 0 = o(\{\})
\end{aligned}$$

Therefore we can conclude that \mathcal{R} is a bi-simulation, and $\{a(ba)^*b\}\mathcal{R}\{ab(ab)^*\}$.

5 The Axiomatization

In this section we will give a co-inductive axiomatization of equality of sets of regular expressions, and we will give an example of a proof in this axiomatization.

We will prove in the following sections, that this system is sound and complete with respect to the semantic notion of equality.

We will in the following sections assume that the alphabet is finite and of the following form.

$$\Sigma = \{a_1, a_2, \dots, a_n\}$$

Definition 5.1 $\Gamma \vdash A_1 = A_2$

The judgment $\Gamma \vdash A_1 = A_2$ where $\Gamma \subseteq \mathbf{Regs}(\Sigma) \times \mathbf{Regs}(\Sigma)$ is a finite set, is given by the following inference rules.

$$\text{Eq - Diff} \frac{\forall i = 1, 2, \dots, n : \Gamma \cup \{(A_1, A_2)\} \vdash D_{a_i}(A_1) = D_{a_i}(A_2)}{\Gamma \vdash A_1 = A_2} (o(A_1) = o(A_2))$$

$$\text{Eq - Hyp} \frac{(A_1, A_2) \in \Gamma}{\Gamma \vdash A_1 = A_2}$$

We notice, that there are only two rules, and we don't need any symmetry, reflexivity or transitivity rules.

In the following examples we will for simplicity assume that there are only two items in our alphabet, so $\Sigma = \{a, b\}$.

We will prove that $\{\} \vdash \{a(ba)^*b\} = \{ab(ab)^*\}$, but let us start by proving that $\{\} \vdash \{\} = \{\}$.

Example 5.2 $\{\} \vdash \{\} = \{\}$

Since we start with the empty set of assumptions, we can't use the Eq-Hyp rule directly, so the root of the proof tree must be of the following form

$$\text{Eq - Diff} \frac{\forall i = 1, 2, \dots, n : \{\} \cup \{(\{\}, \{\})\} \vdash D_{a_i}(\{\}) = D_{a_i}(\{\})}{\{\} \vdash \{\} = \{\}} (o(\{\}) = o(\{\}))$$

Since o is a function, we get that $o(\{\}) = o(\{\})$, so we only need to prove that

$\{(\{\}, \{\})\} \vdash D_a(\{\}) = D_a(\{\})$ and $\{(\{\}, \{\})\} \vdash D_b(\{\}) = D_b(\{\})$.

But by using the definition of D we get that $D_a(\{\}) = D_b(\{\}) = \{\}$, so we only need to prove that $\{(\{\}, \{\})\} \vdash \{\} = \{\}$ and we can do this using the Eq-Hyp rule.

This means that the entire derivation of $\{\} \vdash \{\} = \{\}$ looks like this,

$$\text{Eq - Diff} \frac{\text{Eq - Hyp} \frac{(\{\}, \{\}) \in \{(\{\}, \{\})\}}{\{(\{\}, \{\})\} \vdash \{\} = \{\}} \quad \text{Eq - Hyp} \frac{(\{\}, \{\}) \in \{(\{\}, \{\})\}}{\{(\{\}, \{\})\} \vdash \{\} = \{\}}}{\{\} \vdash \{\} = \{\}} (o(\{\}) = o(\{\})) \quad (51)$$

This was a rather trivial example, but now we are ready to prove that $\{\} \vdash \{a(ba)^*b\} = \{ab(ab)^*\}$.

Example 5.3 $\{\} \vdash \{a(ba)^*b\} = \{ab(ab)^*\}$

Just like in the previous example we need to start with the Eq-Diff rule, so the root of the proof-tree must look like this (where we define $N = \{1, 2, \dots, n\}$)

$$\frac{\forall i \in N : \{\} \cup \{(\{a(ba)^*b\}, \{ab(ab)^*\})\} \vdash D_{a_i}(\{a(ba)^*b\}) = D_{a_i}(\{ab(ab)^*\})}{\{\} \vdash \{a(ba)^*b\} = \{ab(ab)^*\}} (o(\{a(ba)^*b\}) = o(\{ab(ab)^*\}))$$

Since $o(\{a(ba)^*b\}) = 0 = o(\{ab(ab)^*\})$ we only need to prove that

$\Gamma_1 \vdash D_a(\{a(ba)^*b\}) = D_a(\{ab(ab)^*\})$ and $\Gamma_1 \vdash D_b(\{a(ba)^*b\}) = D_b(\{ab(ab)^*\})$,
if we let $\Gamma_1 = \{(\{a(ba)^*b\}, \{ab(ab)^*\})\}$.

Now we can use the definition of D to reduce these equalities to

$\Gamma_1 \vdash \{(ba)^*b\} = \{b(ab)^*\}$ and $\Gamma_1 \vdash \{\} = \{\}$.

By generalizing proof (51) we can obtain the second equality, but to obtain the first equality, we need to use the Eq-Diff rule again. (where we define $N = \{1, 2, \dots, n\}$)

$$\frac{\forall i \in N : \Gamma_1 \cup \{(\{(ba)^*b\}, \{b(ab)^*\})\} \vdash D_{a_i}(\{(ba)^*b\}) = D_{a_i}(\{b(ab)^*\})}{\Gamma_1 \vdash \{(ba)^*b\} = \{b(ab)^*\}} (o(\{(ba)^*b\}) = o(\{b(ab)^*\}))$$

Since $o(\{(ba)^*b\}) = 0 = o(\{b(ab)^*\})$ we only need to prove that

$\Gamma_2 \vdash D_a(\{(ba)^*b\}) = D_a(\{b(ab)^*\})$ and $\Gamma_2 \vdash D_b(\{(ba)^*b\}) = D_b(\{b(ab)^*\})$,
if we let $\Gamma_2 = \{(\{a(ba)^*b\}, \{ab(ab)^*\}), (\{(ba)^*b\}, \{b(ab)^*\})\}$.

Now we can again use the definition of D to reduce the desired equalities to

$\Gamma_2 \vdash \{\} = \{\}$ and $\Gamma_2 \vdash \{a(ba)^*b, 1\} = \{(ab)^*\}$.

By generalizing proof (51) we can obtain the first equality, but to obtain the second equality, we need to use the Eq-Diff rule one last time. (where we define $N = \{1, 2, \dots, n\}$)

$$\frac{\forall i \in N : \Gamma_2 \cup \{(\{a(ba)^*b, 1\}, \{(ab)^*\})\} \vdash D_{a_i}(\{a(ba)^*b, 1\}) = D_{a_i}(\{(ab)^*\})}{\Gamma_2 \vdash \{a(ba)^*b, 1\} = \{(ab)^*\}} (o(\{a(ba)^*b, 1\}) = o(\{(ab)^*\}))$$

Since $o(\{a(ba)^*b, 1\}) = 1 = o(\{(ab)^*\})$ we only need to prove that

$\Gamma_3 \vdash D_a(\{a(ba)^*b, 1\}) = D_a(\{(ab)^*\})$ and $\Gamma_3 \vdash D_b(\{a(ba)^*b, 1\}) = D_b(\{(ab)^*\})$,
if we let $\Gamma_3 = \{(\{a(ba)^*b\}, \{ab(ab)^*\}), (\{(ba)^*b\}, \{b(ab)^*\}), (\{a(ba)^*b, 1\}, \{(ab)^*\})\}$.

Now we can again use the definition of D to reduce the desired equalities to

$\Gamma_3 \vdash \{(ba)^*b\} = \{b(ab)^*\}$ and $\Gamma_3 \vdash \{\} = \{\}$.

By generalizing proof (51) we can obtain the second equality, and since

$((ba)^*b, b(ab)^*) \in \Gamma_3 = \{(\{a(ba)^*b\}, \{ab(ab)^*\}), (\{(ba)^*b\}, \{b(ab)^*\}), (\{a(ba)^*b, 1\}, \{(ab)^*\})\}$,
we can prove the first equality using rule Eq-Hyp like this

$$\text{Eq - Hyp} \frac{(\{(ba)^*b\}, \{b(ab)^*\}) \in \Gamma_3}{\Gamma_3 \vdash \{(ba)^*b\} = \{b(ab)^*\}}$$

Now we can derive $\{\} \vdash \{a(ba)^*b\} = \{ab(ab)^*\}$ by collecting the proof-parts as described above.

6 Soundness

Before we can prove soundness of the axiomatization we need to show a theorem about, how the language of the derivatives of a set of regular expressions are related to the derivatives. As the following theorem shows, and as seen in Section 3.3 the language of the derivative is closely related to the language of the original expressions.

Theorem 6.1 *Specialized version of the “Fundamental Theorem of languages”.*

For all $A \in \mathbf{Regs}(\Sigma)$

$$\mathcal{L}_\Sigma(A) = \mathcal{L}_\Sigma(\{o(A)\}) \cup \bigcup_{i=1}^n a_i \mathcal{L}_\Sigma(D_{a_i}(A))$$

Proof:

“ \supseteq ”: By Corollary 2.12 we get that $o(A) = 1$ if and only if $\epsilon \in \mathcal{L}_\Sigma(A)$, hence $\mathcal{L}_\Sigma(\{o(A)\}) \subseteq \mathcal{L}_\Sigma(A)$. Now we only need to prove that

$$\forall i \ 1 \leq i \leq n. \quad a_i \mathcal{L}_\Sigma(D_{a_i}(A)) \subseteq \mathcal{L}_\Sigma(A)$$

Assume $1 \leq i \leq n$. By Corollary 3.8 we get that

$$\begin{aligned} a_i \mathcal{L}_\Sigma(D_{a_i}(A)) &= a_i \{w \mid a_i w \in \mathcal{L}_\Sigma(A)\} \\ &= \{a_i w \mid a_i w \in \mathcal{L}_\Sigma(A)\} \\ &\subseteq \mathcal{L}_\Sigma(A) \end{aligned}$$

which completes the first part of the proof.

“ \subseteq ”: Assume $w \in \mathcal{L}_\Sigma(A)$. We split the proof depending on w .

$w = \epsilon$: By Corollary 2.12 we get that $o(A) = 1$, and hence $w = \epsilon \in \mathcal{L}_\Sigma(\{o(A)\})$.

$w \neq \epsilon$: Hence $w = a_i w'$ for some $1 \leq i \leq n$ and some $w' \in \Sigma^*$. By Corollary 3.8 we get that

$$\begin{aligned} a_i \mathcal{L}_\Sigma(D_{a_i}(A)) &= a_i \{w \mid a_i w \in \mathcal{L}_\Sigma(A)\} \\ &= \{a_i w \mid a_i w \in \mathcal{L}_\Sigma(A)\} \end{aligned}$$

Since $w = a_i w' \in \mathcal{L}_\Sigma(A)$ we get that $w \in a_i \mathcal{L}_\Sigma(D_{a_i}(A))$, which completes the proof. \blacksquare

6.1 Level stratified interpretation

In order to make the soundness proof work, we make the same trick as seen in [1] and [2], which is to introduce another semantical interpretation of sequents (*level stratified interpretation*). Before we can define the level stratified interpretation we need to define *approximation* of a language.

Definition 6.2 *Approximation of a language.*

Assume L is a language then we define

$$L|_n = \{w \mid |w| \leq n \wedge w \in L\}, \quad \forall n \geq 0$$

Approximations of languages are related to the original languages in the following way.

Lemma 6.3 *Assume L and L' are languages then*

$$(\forall k \geq 0. L|_k = L'|_k) \Leftrightarrow L = L'$$

Proof:

\Leftarrow : Trivial, since $\cdot|_k$ is a map.

\Rightarrow : We first show that $L \subseteq L'$. Assume $w \in L$ then $w \in L|_{|w|}$, and hence $L'|_{|w|}$ and thus $w \in L'$. The case for $L \supseteq L'$ is symmetric. \blacksquare

Now we are ready to define level stratified interpretation of sequents.

Definition 6.4 *Level stratified interpretation.*

For all $A, B \in \mathbf{Reqs}(\Sigma)$ we define

$$\begin{aligned} \vDash_k A = B &\Leftrightarrow \mathcal{L}_\Sigma(A)|_k = \mathcal{L}_\Sigma(B)|_k \\ \vDash_k \Gamma &\Leftrightarrow \forall (A, B) \in \Gamma. \vDash_k A = B \\ \Gamma \vDash_k A = B &\Leftrightarrow \vDash_k \Gamma \Rightarrow \vDash_k A = B \\ \Gamma \vDash^s A = B &\Leftrightarrow \forall k \in \mathbb{N}_0. \Gamma \vDash_k A = B \end{aligned}$$

Intuitively, if we have $\Gamma \vDash^s A = B$ it says that for all k , if we approximate the languages in the assumptions Γ with k , then the approximations of the languages of A and B are the same, and this holds for all k .

6.2 The axiomatization is sound

Now we can prove that the axiomatization is sound with respect to the level stratified interpretation.

Theorem 6.5 *Soundness with respect to the level stratified interpretation.*

$$\Gamma \vdash A = B \Rightarrow \Gamma \vDash^s A = B$$

Proof:

By rule induction.

Eq-Hyp: Assume that $(A, B) \in \Gamma$. We need to prove that $\Gamma \vDash^s A = B$, ie. we need to prove that

$$\forall k \in \mathbb{N}_0. \vDash_k \Gamma \Rightarrow \vDash_k A = B$$

Assume that $k \in \mathbb{N}_0$ and assume that $\vDash_k \Gamma$. Since we have assumed that $(A, B) \in \Gamma$ we get $\vDash_k A = B$, which completes the proof in this case.

Eq-Diff: Assume by induction that

$$\forall 1 \leq i \leq n. \Gamma \cup \{(A, B)\} \vDash^s D_{a_i}(A) = D_{a_i}(B) \tag{52}$$

and $o(A) = o(B)$. We need to show that $\Gamma \vDash^s A = B$, ie. we need to show

$$\forall k \in \mathbb{N}_0. \vDash_k \Gamma \Rightarrow \vDash_k A = B$$

We prove this by induction on k .
 $k = 0$: Since

$$\mathcal{L}_\Sigma(A)|_0 = \{w \mid |w| \leq 0 \wedge w \in \mathcal{L}_\Sigma(A)\} \quad (53)$$

$$= \begin{cases} \{\epsilon\} & o(A) = 1 \\ \emptyset & o(A) \neq 1 \end{cases} \quad (54)$$

$$= \begin{cases} \{\epsilon\} & o(B) = 1 \\ \emptyset & o(B) \neq 1 \end{cases} \quad (55)$$

$$= \{w \mid |w| \leq 0 \wedge w \in \mathcal{L}_\Sigma(B)\} \quad (56)$$

$$= \mathcal{L}_\Sigma(B)|_0 \quad (57)$$

(55): Since $o(A) = o(B)$.

which completes this case.

$k > 0$: By induction we get that

$$\vDash_{k-1} \Gamma \quad \Rightarrow \quad \vDash_{k-1} A = B \quad (58)$$

Assume $\vDash_k \Gamma$. Hence $\vDash_{k-1} \Gamma$. By (58) we get that $\vDash_{k-1} A = B$ and therefore $\vDash_{k-1} \Gamma \cup \{(A, B)\}$.
 By (52) we get that

$$\forall 1 \leq i \leq n. \quad \vDash_{k-1} D_{a_i}(A) = D_{a_i}(B) \quad (59)$$

Since $o(A) = o(B)$ we get that

$$L(A)|_k = (\mathcal{L}_\Sigma(\{o(A)\}) \cup a_i \mathcal{L}_\Sigma(D_{a_i}(A)))|_k \quad (60)$$

$$= \mathcal{L}_\Sigma(\{o(A)\})|_k \cup a_i (\mathcal{L}_\Sigma(D_{a_i}(A)))|_{k-1} \quad (61)$$

$$= \mathcal{L}_\Sigma(\{o(B)\})|_k \cup a_i (\mathcal{L}_\Sigma(D_{a_i}(B)))|_{k-1} \quad (62)$$

$$= (\mathcal{L}_\Sigma(\{o(B)\}) \cup a_i \mathcal{L}_\Sigma(D_{a_i}(B)))|_k \quad (63)$$

$$= \mathcal{L}_\Sigma(B)|_k \quad (64)$$

(60): Follows from Theorem 6.1.

(61): Follows easily from the definition of approximation.

(62): Follows from (59) and since $o(A) = o(B)$.

(64): Follows from Theorem 6.1. Hence $\vDash_k A = B$ which completes the proof. \blacksquare

From Theorem 6.5 we are now able to show, that the axiomatization is sound with respect to the usual interpretation.

Corollary 6.6 *Soundness with respect to the usual interpretation.*

For all $A, B \in \mathbf{Regs}(\Sigma)$ we have

$$\vdash A = B \quad \Rightarrow \quad \vDash A = B$$

Proof:

Assume $A, B \in \mathbf{Regs}(\Sigma)$ and assume that $\vdash A = B$. By Theorem 6.5 we get that $\vDash^s A = B$, ie.

$$\forall k \in \mathbb{N}_0. \quad \vDash_k A = B$$

which means that

$$\forall k \in \mathbb{N}_0. \quad \mathcal{L}_\Sigma(A)|_k = \mathcal{L}_\Sigma(B)|_k$$

By Lemma 6.3 we get that $\mathcal{L}_\Sigma(A) = \mathcal{L}_\Sigma(B)$ and hence $\vDash A = B$. \blacksquare

7 Completeness

In this section we will prove that the co-inductive axiomatization of equality is complete.

We will prove the completeness in three steps.

First we will prove that the set of pairs of semantically equal sets of regular expressions is a bi-simulation, and therefore for all pairs of semantically equal sets of regular expressions there is a bi-simulation relating them.

Then we will prove that if there is a bi-simulation relating two sets of regular expressions, then there is a finite bi-simulation relating those sets.

Finally we will prove that if there is a finite bi-simulation relating two sets of regular expressions, then there is a derivation in the co-inductive axiomatization of equality concluding that the two sets are equal.

Before we prove completeness, let us investigate the problem with the co-inductive axiomatization from [4].

7.1 The problem with Grabmeyers axiomatization

Let us see why the axiomatization would not be complete, if we had used the definition of differentiation from [5] as suggested in [4].

In [5, p. 43] the derivate is defined by $(\cdot)_a : \mathbf{Reg}(\Sigma) \rightarrow \mathbf{Reg}(\Sigma)$:

$$\begin{aligned} (0)_a &= 0 \\ (1)_a &= 0 \\ (b)_a &= \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases} \\ (E + F)_a &= (E)_a + (F)_a \\ (EF)_a &= (E)_a F + \bar{o}(E) (F)_a \\ (E^*)_a &= (E)_a E^* \end{aligned}$$

where \bar{o} is as defined earlier.

Furthermore the axiomatization in [4] is not defined on sets of regular expressions, and has the judgment form $\Gamma \vdash E = F$, where Γ is a set of equations on regular expressions, but otherwise the rules look the same.

Example 7.1 *With the above definition of differentiation the axiomatization is not complete. We now try to differentiate $(a^*)^*$.*

$$\begin{aligned} ((a^*)^*)_a &= (a^*)_a (a^*)^* \\ &= (1a^*) (a^*)^* \end{aligned}$$

Let us differentiate the result with respect to a

$$\begin{aligned} ((1a^*) (a^*)^*)_a &= ((1a^*))_a (a^*)^* + \bar{o}(1a^*) ((a^*)^*)_a \\ &= ((1a^*))_a (a^*)^* + 1 ((a^*)^*)_a \\ &= ((1a^*))_a (a^*)^* + 1 (1a^*) (a^*)^* \end{aligned}$$

Notice that the original expressions now occurs twice in the second derivative with respect to a , and if we keep on differentiating we will get an ever expanding expression. Hence the

axiomatization can not be complete, because if we tried to prove that $\vdash (a^*)^* = a^*$, we would continuously add new equations to the set of assumptions using the Eq – Diff rule.

From Example 7.1 we see that using the definition of derivative from [5] we get problems with ever expanding expressions. Let us now see why our definition of differentiation can handle this problem.

Example 7.2 We now prove that $\vdash \{(a^*)^*\} = \{a^*\}$.

Since

$$\begin{aligned} D_a(\{(a^*)^*\}) &= \overline{D}_a(a^*)(a^*)^* \\ &= ((\overline{D}_a(a)) a^*)(a^*)^* \\ &= (\{1\} a^*)(a^*)^* \\ &= \{a^*\} (a^*)^* \\ &= \{a^* (a^*)^*\} \end{aligned}$$

and the second derivative is (this will not be differentiated in detail)

$$D_a(\{a^* (a^*)^*\}) = \{a^* (a^*)^*, (a^*)^*\}$$

and the third derivative will then be

$$D_a(\{a^* (a^*)^*, (a^*)^*\}) = \{a^* (a^*)^*, (a^*)^*\}$$

Since $D_a(\{a^*\}) = \{a^*\}$ we see that using three applications of the Eq – Diff rule and one application of the Eq – Hyp rule will complete the proof.

The problem in Example 7.1 is that the definition of the derivative from [5] does not eliminate 0 and 1 where possible, and it does not collapse identical subexpressions. These problems should be fixed by our definition, and later we will prove, that given a set of regular expressions, then there are only finitely many different derivatives, and we will see why this is sufficient to prove that the axiomatization is complete.

7.2 The equality bi-simulation

First we will define the set of semantically equal sets of regular expressions.

Definition 7.3 The set of semantically equal sets of regular expressions: $\mathcal{R}_=$

We define

$$\mathcal{R}_= = \{(A_1, A_2) \in \mathbf{Regs}(\Sigma) \times \mathbf{Regs}(\Sigma) \mid \models A_1 = A_2\}$$

Lemma 7.4 $\mathcal{R}_=$ is a bi-simulation.

We prove this, by proving the two properties of a bi-simulation are fulfilled by $\mathcal{R}_=$.

The first property states that $A_1 \mathcal{R}_= A_2 \Rightarrow \forall a \in \Sigma : D_a(A_1) \mathcal{R}_= D_a(A_2)$.

Assume that $A_1 \mathcal{R}_= A_2$. This means by definition that $\models A_1 = A_2$, and therefore $\mathcal{L}(A_1) = \mathcal{L}(A_2)$. Choose an arbitrary $a \in \Sigma$.

Now Corollary 3.8 yields that

$\mathcal{L}(D_a(A_1)) = \{w \mid aw \in \mathcal{L}(A_1)\} = \{w \mid aw \in \mathcal{L}(A_2)\} = \mathcal{L}(D_a(A_2))$, so $\models D_a(A_1) = D_a(A_2)$.

We can therefore conclude that $\forall a \in \Sigma : D_a(A_1)\mathcal{R}=D_a(A_2)$.

The second property states that $A_1\mathcal{R}=A_2 \Rightarrow o(A_1) = o(A_2)$.

Assume that $A_1\mathcal{R}=A_2$. This means by definition that $\vDash A_1 = A_2$, and therefore $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

Now Corollary 2.12 yields that $o(A_1) = \begin{cases} 1 & \varepsilon \in \mathcal{L}(A_1) \\ 0 & \text{otherwise} \end{cases} = \begin{cases} 1 & \varepsilon \in \mathcal{L}(A_2) \\ 0 & \text{otherwise} \end{cases} = o(A_2)$.

We can therefore conclude that $o(A_1) = o(A_2)$. ■

Now we know that if two sets of regular expressions are semantically equal, then there is a bi-simulation (namely $\mathcal{R}=_$) that relates them.

We will now prove that this means that there is also a finite bi-simulation relating them. But before we can do this we need to prove, that there are only finitely many different derivatives.

7.3 There are only finitely many derivatives

In this section we will prove that there are only finitely many derivatives. This is needed if we want to ensure, that if two sets A and B of regular expressions are equal, then there exists a finite bi-simulation \mathcal{R} such that $A \mathcal{R} B$.

Definition 7.5 *Restriction of a sequence.*

For $\pi \in \Sigma^+$ we define

$$\pi_i = \begin{cases} \text{The last } i \text{ symbols in the sequence } \pi & 0 < i < |\pi| \\ \text{All symbols in the sequence } \pi & |\pi| \leq i \end{cases}$$

Lemma 7.6 *For all $\pi \in \Sigma^+$ and for all $E \in \mathbf{Reg}(\Sigma)$ we have*

$$D_\pi(\{E^*\}) \subseteq \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{E\}) \right) E^*$$

Proof:

By induction on the length of the sequence.

$|\pi| = 1$: Assume that $E \in \mathbf{Reg}(\Sigma)$. Since $|\pi| = 1$ we get that $\pi = a$ for some $a \in \Sigma$. Hence

$$\begin{aligned} D_\pi(\{E^*\}) &= D_a(\{E^*\}) \\ &= \overline{D}_a(E^*) \\ &= \overline{D}_a(E) E^* \\ &= D_a(\{E\}) E^* \\ &= \left(\bigcup_{i=1}^{|\pi|} D_{a_i}(\{E\}) \right) E^* \\ &= \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{E\}) \right) E^* \end{aligned}$$

so the lemma holds in this case.

$|\pi| > 1$: Ie. we can write $\pi = \sigma a$ for some $a \in \Sigma$ and $\sigma \in \Sigma^+$, where $1 \leq |\sigma| < |\pi|$ and $|\sigma| + 1 = |\pi|$. Hence

$$D_\pi(\{E^*\}) = D_{\sigma a}(\{E^*\}) \quad (65)$$

$$= D_a(D_\sigma(\{E^*\})) \quad (66)$$

$$\subseteq D_a\left(\left(\bigcup_{i=1}^{|\sigma|} D_{\sigma_i}(\{E\})\right) E^*\right) \quad (67)$$

$$= D_a\left(\bigcup_{i=1}^{|\sigma|} (D_{\sigma_i}(\{E\}) E^*)\right) \quad (68)$$

$$= \bigcup_{i=1}^{|\sigma|} D_a(D_{\sigma_i}(\{E\}) E^*) \quad (69)$$

$$= \bigcup_{i=1}^{|\sigma|} D_a(D_{\sigma_i}(\{E\})) E^* \cup o(D_{\sigma_i}(\{E\})) D_a(\{E^*\}) \quad (70)$$

$$\subseteq \bigcup_{i=1}^{|\sigma|} D_a(D_{\sigma_i}(\{E\})) E^* \cup D_a(\{E^*\}) \quad (71)$$

$$= \left(\bigcup_{i=1}^{|\sigma|} D_a(D_{\sigma_i}(\{E\})) E^*\right) \cup D_a(\{E^*\}) \quad (72)$$

$$= \left(\bigcup_{i=1}^{|\sigma|} D_{\sigma_i a}(\{E\}) E^*\right) \cup D_a(\{E^*\}) \quad (73)$$

$$= \left(\bigcup_{i=1}^{|\sigma|} D_{(\sigma a)_{i+1}}(\{E\}) E^*\right) \cup D_a(\{E^*\}) \quad (74)$$

$$= \left(\bigcup_{i=2}^{|\sigma|+1} D_{(\sigma a)_i}(\{E\}) E^*\right) \cup D_a(\{E^*\}) \quad (75)$$

$$= \left(\bigcup_{i=2}^{|\sigma|+1} D_{(\sigma a)_i}(\{E\}) E^*\right) \cup D_a(\{E\}) E^* \quad (76)$$

$$= \bigcup_{i=1}^{|\sigma|+1} D_{(\sigma a)_i}(\{E\}) E^* \quad (77)$$

$$= \bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{E\}) E^* \quad (78)$$

$$(79)$$

(67): Since $\pi = \sigma a$ we get that $|\sigma| < |\pi|$. By induction we get that $D_\sigma(\{E^*\}) \subseteq \left(\bigcup_{i=0}^{|\sigma|} D_{\sigma_i}(\{E\})\right) E^*$, and since D_a is monotone (Lemma 3.6) the inclusion holds.

(69): By Lemma 3.6.

(70): By Lemma 3.6.

(71): Since $o(D_{\sigma_i}(\{E\})) = 0, 1$.

This completes the proof. ■

Lemma 7.7 *For all $\pi \in \Sigma^+$ and for all $E, F \in \mathbf{Reg}(\Sigma)$ we have*

$$D_\pi(\{EF\}) \subseteq D_\pi(\{E\})F \cup \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F\}) \right)$$

Proof:

This proof is similar to the proof of Lemma 7.6, so some of the details will be omitted.

Induction on $|\pi|$.

$|\pi| = 1$: Hence $\pi = a \in \Sigma$ for some a . Assume $E, F \in \mathbf{Reg}(\Sigma)$. Since

$$\begin{aligned} D_\pi(\{EF\}) &= D_a(\{EF\}) \\ &= D_a(\{E\})F \cup o(\{E\})D_a(\{F\}) \\ &\subseteq D_a(\{E\})F \cup D_a(\{F\}) \\ &= D_a(\{E\})F \cup \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F\}) \right) \\ &= D_\pi(\{E\})F \cup \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F\}) \right) \end{aligned}$$

which completes this case.

$|\pi| > 1$: Assume $E, F \in \mathbf{Reg}(\Sigma)$. In this case we have $\pi = \sigma a$ for some $a \in \Sigma$ and $\sigma \in \Sigma^+$,

where $1 \leq |\sigma| < |\pi|$ and $|\sigma| + 1 = |\pi|$. Since

$$D_\pi(\{EF\}) = D_{\sigma a}(\{EF\}) \quad (80)$$

$$= D_a(D_\sigma(\{EF\})) \quad (81)$$

$$\subseteq D_a\left(D_\sigma(\{E\})F \cup \left(\bigcup_{i=1}^{|\sigma|} D_{\sigma_i}(\{F\})\right)\right) \quad (82)$$

$$= D_a(D_\sigma(\{E\})F) \cup \left(\bigcup_{i=1}^{|\sigma|} D_a(D_{\sigma_i}(\{F\}))\right) \quad (83)$$

$$= D_{\sigma a}(\{E\})F \cup o(D_\sigma(\{E\}))D_a(\{F\}) \cup \left(\bigcup_{i=1}^{|\sigma|} D_{\sigma_i a}(\{F\})\right) \quad (84)$$

$$\subseteq D_{\sigma a}(\{E\})F \cup D_a(\{F\}) \cup \left(\bigcup_{i=1}^{|\sigma|} D_{\sigma_i a}(\{F\})\right) \quad (85)$$

$$= D_{\sigma a}(\{E\})F \cup D_a(\{F\}) \cup \left(\bigcup_{i=2}^{|\sigma|+1} D_{(\sigma a)_i}(\{F\})\right) \quad (86)$$

$$= D_{\sigma a}(\{E\})F \cup \left(\bigcup_{i=1}^{|\sigma|+1} D_{(\sigma a)_i}(\{F\})\right) \quad (87)$$

$$= D_\pi(\{E\})F \cup \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F\})\right) \quad (88)$$

(82): Follows by induction and since D_a is monotone (Lemma 3.6).

(83): Since D_a is distributive over set union (Lemma 3.6).

This completes the proof of the lemma. ■

Theorem 7.8 *There are only finitely many differentials.*

$$\forall E \in \mathbf{Reg}(\Sigma). \quad \{D_\pi(\{E\}) \mid \pi \in \Sigma^+\} \text{ is finite}$$

Proof:

By structural induction on E .

$E = 0, 1$: Trivial, since $D_a(\{0\}) = D_a(\{1\}) = \emptyset$ for all $a \in \Sigma$, and hence

$$\{D_\pi(\{0\}) \mid \pi \in \Sigma^+\} = \{D_\pi(\{1\}) \mid \pi \in \Sigma^+\} = \{\emptyset\}$$

which completes this case.

$E = b$: Since $D_a(\{b\}) = \begin{cases} \{1\} & a = b \\ \emptyset & a \neq b \end{cases}$, we get that

$$\{D_\pi(\{b\}) \mid \pi \in \Sigma^+\} \subseteq \{\{1\}, \emptyset\}$$

which completes this case.

$E = E' + F'$: By induction we get that $\{D_\pi(\{E'\}) \mid \pi \in \Sigma^+\}$ and $\{D_\pi(\{F'\}) \mid \pi \in \Sigma^+\}$ are finite. Hence

$$\{D_\pi(\{E' + F'\}) \mid \pi \in \Sigma^+\} = \{D_\pi(\{E'\}) \cup D_\pi(\{F'\}) \mid \pi \in \Sigma^+\}$$

must be finite, since we can only generate finitely many different elements.

$E = E'^*$: By induction we get that $\{D_\pi(\{E\}) \mid \pi \in \Sigma^+\}$ is finite, and by Lemma 3.4 we get that $D_\pi(\{E\})$ is finite for all $\pi \in \Sigma^+$. Hence

$$M = \bigcup_{\pi \in \Sigma^+} D_\pi(\{E\}) \text{ is finite}$$

Since

$$\forall \pi \in \Sigma^+. \bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{E\}) \subseteq M$$

(This holds because $\pi_i \in \Sigma^+$ for all $i \geq 1$) we get that

$$\forall \pi \in \Sigma^+. \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{E\}) \right) E^* \subseteq ME^* \quad (89)$$

Since M is finite, then so is ME^* . From Lemma 7.6 and from (89) we get that

$$\forall \pi \in \Sigma^+. D_\pi(\{E^*\}) \subseteq ME^*$$

Hence

$$\{D_\pi(\{E^*\}) \mid \pi \in \Sigma^+\} \subseteq \mathcal{P}(ME^*)$$

Since ME^* is finite then so is $\mathcal{P}(ME^*)$, which completes the proof in this case.

$E = E'F'$: By induction we get that

$$\{D_\pi(\{E'\}) \mid \pi \in \Sigma^+\} \quad \text{is finite} \quad (90)$$

$$\{D_\pi(\{F'\}) \mid \pi \in \Sigma^+\} \quad \text{is finite} \quad (91)$$

Define

$$M = \bigcup_{\pi \in \Sigma^+} \left(D_\pi(\{E'\}) \cup \bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F'\}) \right)$$

We first prove that M is finite.

$$\begin{aligned} M &= \bigcup_{\pi \in \Sigma^+} \left(D_\pi(\{E'\}) \cup \bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F'\}) \right) \\ &= \bigcup_{\pi \in \Sigma^+} (D_\pi(\{E'\})) \cup \bigcup_{\pi \in \Sigma^+} \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F'\}) \right) \end{aligned}$$

By Lemma 3.4 we get that for all $\pi \in \Sigma$ that $D_\pi(\{E'\})$ is finite, so from (90) we get that $\bigcup_{\pi \in \Sigma^+} (D_\pi(\{E'\}))$ is a union of finitely many finite sets, and hence it must itself be finite.

By Lemma 3.4 we get that for all $\pi \in \Sigma$ that $D_\pi(\{F'\})$ is finite, and by (91) we get that $\{D_\pi(\{F'\}) \mid \pi \in \Sigma^+\}$ is finite. Hence $\left\{\bigcup_{i=1}^{|\pi|} D_\pi(\{F'\}) \mid \pi \in \Sigma^+\right\}$ is a finite set of finite sets. This means that

$$\bigcup_{\pi \in \Sigma^+} \left(\bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F'\}) \right) \text{ is finite} \quad (92)$$

Hence M is a union of finitely many finite sets, and therefore M is finite.

By the definition of M and by Lemma 7.7 we get that

$$\forall \pi \in \Sigma^+. D_\pi(\{E'F'\}) \subseteq D_\pi(\{E'\}) \cup \bigcup_{i=1}^{|\pi|} D_{\pi_i}(\{F'\}) \subseteq M$$

and hence

$$\{D_\pi(\{E'F'\}) \mid \pi \in \Sigma^+\} \subseteq \mathcal{P}(M)$$

Since M is finite then so is $\mathcal{P}(M)$, and this completes the proof. \blacksquare

Corollary 7.9 *There are only finitely many differentials.*

$$\forall A \in \mathbf{Regs}(\Sigma). \quad \{D_\pi(A) \mid \pi \in \Sigma^+\} \text{ is finite}$$

Proof: Since $A \in \mathbf{Regs}(\Sigma)$ we can write $A = \bigcup_{i \in I} \{E_i\}$ for some finite index set I . Assume $\pi \in \Sigma^*$ then using $|\pi|$ applications of Lemma 3.6 we get that

$$\begin{aligned} D_\pi(A) &= D_\pi\left(\bigcup_{i \in I} \{E_i\}\right) \\ &= \bigcup_{i \in I} D_\pi(\{E_i\}) \end{aligned}$$

By Theorem 7.8 we get that

$$\forall i \in I. \quad M_i = \{D_\pi(\{E_i\}) \mid \pi \in \Sigma^+\} \text{ is finite}$$

Hence

$$\begin{aligned} |\{D_\pi(A) \mid \pi \in \Sigma^+\}| &= \left| \left\{ \bigcup_{i \in I} D_\pi(\{E_i\}) \mid \pi \in \Sigma^+ \right\} \right| \\ &\leq \prod_{i \in I} |M_i| \\ &< \infty \end{aligned}$$

where the last inequality follows since I is finite. \blacksquare

7.4 Completeness theorem

Lemma 7.10 *If \mathcal{R} is a bi-simulation such that $A_1\mathcal{R}A_2$ then there is a finite bi-simulation \mathcal{R}' such that $A_1\mathcal{R}'A_2$.*

Given a bi-simulation \mathcal{R} such that $A_1\mathcal{R}A_2$, we define

$$\mathcal{R}' = \{(A'_1, A'_2) \in \{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \mid A'_1\mathcal{R}A'_2\} \cup \{(A_1, A_2)\}.$$

We will now prove, that \mathcal{R}' is a finite bi-simulation such that $A_1\mathcal{R}'A_2$.

- \mathcal{R}' is finite.

Corollary 7.9 yields that $\{D_\pi(A_1) \mid \pi \in \Sigma^+\}$ and $\{D_\pi(A_2) \mid \pi \in \Sigma^+\}$ are finite sets, and since $\mathcal{R}' \subseteq \{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\}$, we can conclude that \mathcal{R}' is finite.

- \mathcal{R}' is a bi-simulation.

Assume that $A'_1\mathcal{R}'A'_2$. Since $A'_1\mathcal{R}'A'_2 \Rightarrow A'_1\mathcal{R}A'_2$, the definition of bi-simulations yields that $A'_1\mathcal{R}'A'_2 \Rightarrow o(A'_1) = o(A'_2)$.

Since $\mathcal{R}' \subseteq \{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\}$ there are two cases for $A'_1\mathcal{R}'A'_2$.

The first case is that $A'_1 = A_1$ and $A'_2 = A_2$. We have that $A_1\mathcal{R}A_2$ and therefore $\forall a \in \Sigma : D_a(A_1)\mathcal{R}D_a(A_2)$, and therefore $\forall a \in \Sigma : D_a(A_1)\mathcal{R}'D_a(A_2)$.

The second case is that $A'_1 = D_{\pi_1}(A_1)$ and $A'_2 = D_{\pi_2}(A_2)$ for some π_1 and π_2 . We have that $A'_1\mathcal{R}'A'_2 \Rightarrow A'_1\mathcal{R}A'_2 \Rightarrow \forall a \in \Sigma : D_a(A'_1)\mathcal{R}D_a(A'_2)$, and since $D_a(A'_1) = D_a(D_{\pi_1}(A_1)) = D_{\pi_1 a}(A_1)$ and since $D_a(A'_2) = D_a(D_{\pi_2}(A_2)) = D_{\pi_2 a}(A_2)$ we get that $A'_1\mathcal{R}'A'_2 \Rightarrow \forall a \in \Sigma : D_a(A'_1)\mathcal{R}'D_a(A'_2)$.

- $A_1\mathcal{R}'A_2$.

Since we directly add this pair to \mathcal{R}' , the definition yields that $A_1\mathcal{R}'A_2$.

■

Now we are ready to prove the completeness theorem, because we know that if two sets of regular expressions are semantically equal, then there is a finite bi-simulation relating them. Now we only need to prove that this means that there is a derivation of their equality in the co-inductive axiomatization.

Theorem 7.11 *If $\models A_1 = A_2$ then $\forall \Gamma. \Gamma \vdash A_1 = A_2$.*

Since $\models A_1 = A_2$ we have that $A_1\mathcal{R}_=A_2$, and Lemma 7.4 yields that $\mathcal{R}_=$ is a bi-simulation.

Therefore Lemma 7.10 yields that there is a finite bi-simulation \mathcal{R} such that $A_1\mathcal{R}A_2$.

We will now prove that this means that $\Gamma \vdash A_1 = A_2$ for all Γ by induction on $|\mathcal{R} \setminus \Gamma|$.

Start: $|\mathcal{R} \setminus \Gamma| = 0$

In this case $(A_1, A_2) \in \mathcal{R} \subseteq \Gamma$, so we can prove the desired as below.

$$\text{Eq} - \text{Hyp} \frac{(A_1, A_2) \in \Gamma}{\Gamma \vdash A_1 = A_2}$$

Induction Hypothesis:

If $|\mathcal{R} \setminus \Gamma'| < |\mathcal{R} \setminus \Gamma|$ and $A'_1\mathcal{R}A'_2$ then we can assume that there is a derivation of $\Gamma' \vdash A'_1 = A'_2$.

Step: $|\mathcal{R} \setminus \Gamma| > 0$

If $(A_1, A_2) \in \Gamma$ then we can simply use the Eq-Hyp rule.

If $(A_1, A_2) \notin \Gamma$ we define $\Gamma' = \Gamma \cup \{(A_1, A_2)\}$, and we observe that $|\mathcal{R} \setminus \Gamma'| < |\mathcal{R} \setminus \Gamma|$.

Since $A_1 \mathcal{R} A_2$ we get that $o(A_1) = o(A_2)$ and $\forall i = 1 \dots n : D_{a_i}(A_1) \mathcal{R} D_{a_i}(A_2)$ so the induction hypothesis yields that $\forall i = 1 \dots n : \Gamma' \vdash D_{a_i}(A_1) = D_{a_i}(A_2)$. Therefore we can create the derivation

$$\text{Eq - Diff} \frac{\forall i = 1, 2, \dots, n : \Gamma \cup \{(A_1, A_2)\} \vdash D_{a_i}(A_1) = D_{a_i}(A_2)}{\Gamma \vdash A_1 = A_2} (o(A_1) = o(A_2))$$

■

We will now proceed by giving an algorithm to decide whether two sets of regular expressions are semantically equal, and return a proof if they are.

8 Proof searching

We will now introduce an algorithm that given A_1 and A_2 searches for a derivation of $\vdash A_1 = A_2$. We will later prove that this search terminates either with a derivation of $\vdash A_1 = A_2$ or with an error proving that $\vdash A_1 \neq A_2$.

8.1 Algorithm

RegEq (Γ, A_1, A_2, w)	
▶	if $o(A_1) \neq o(A_2)$
▶	then raise NotEq (w)
▶	else if $(A_1, A_2) \in \Gamma$
▶	then Eq-Hyp
▶	else let $\Gamma' = \Gamma \cup \{(A_1, A_2)\}$
▶	in let
▶	$P_{a_1} = \text{RegEq} (\Gamma', D_{a_1}(A_1), D_{a_1}(A_2), wa_1),$
▶	$\dots,$
▶	$P_{a_n} = \text{RegEq} (\Gamma', D_{a_n}(A_1), D_{a_n}(A_2), wa_n)$
▶	where $\Sigma = \{a_1, a_2, \dots, a_n\}$
▶	in Eq-Diff (P_{a_1}, \dots, P_{a_n})
▶	end
▶	end

In **RegEq** Γ is the set of assumptions, which is initially set to the empty set, A_1 and A_2 are the sets of regular expressions we wish to test for equality, and w is an accumulating parameter which is initially set to the empty string.

8.2 Termination

The first thing we will prove is that **RegEq** terminates, and we do this by limiting the call-depth.

Lemma 8.1 *For all $A \in \mathbf{Rregs}(\Sigma)$ and $a \in \Sigma$ then $\{D_\pi(D_a(A)) \mid \pi \in \Sigma^+\} \subseteq \{D_\pi(A) \mid \pi \in \Sigma^+\}$.*

Let $A \in \mathbf{Rregs}(\Sigma)$ and $a \in \Sigma$ be chosen arbitrarily.

Assume that $A' \in \{D_\pi(D_a(A)) \mid \pi \in \Sigma^+\}$.

This means that $A' = D_w(D_a(A)) = D_{aw}(A)$ and therefore $A' \in \{D_\pi(A) \mid \pi \in \Sigma^+\}$. ■

Lemma 8.2 *For all Γ, A_1, A_2 and w the execution of **RegEq** (Γ, A_1, A_2, w) terminates.*

Theorem 7.9 yields that $\{D_\pi(A_1) \mid \pi \in \Sigma^+\}$ and $\{D_\pi(A_2) \mid \pi \in \Sigma^+\}$ are finite, so we prove this by induction on $|\{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \setminus \Gamma|$.

Start: $|\{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \setminus \Gamma| = 0$

*If $o(A_1) \neq o(A_2)$ then **RegEq** (Γ, A_1, A_2, w) raises an exception, otherwise since $(A_1, A_2) \in \{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \subseteq \Gamma$, **RegEq** (Γ, A_1, A_2, w) returns **Eq-Hyp** directly.*

Induction Hypothesis:

If $|\{D_\pi(A'_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A'_2) \mid \pi \in \Sigma^+\} \cup \{(A'_1, A'_2)\} \setminus \Gamma'| <$

$|\{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \setminus \Gamma|$
then the execution of **RegEq** $(\Gamma', A'_1, A'_2, w')$ terminates.

Step: $|\{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \setminus \Gamma| > 0$

If $o(A_1) \neq o(A_2)$ then **RegEq** (Γ, A_1, A_2, w) raises an exception.

If $(A_1, A_2) \in \Gamma$ then **RegEq** (Γ, A_1, A_2, w) returns **Eq-Hyp** directly.

If $(A_1, A_2) \notin \Gamma$ then Lemma 8.1 yields that

if $|\{D_\pi(D_a(A_1)) \mid \pi \in \Sigma^+\} \times \{D_\pi(D_a(A_2)) \mid \pi \in \Sigma^+\} \cup \{(D_a(A_1), D_a(A_2))\} \setminus \Gamma \cup \{(A_1, A_2)\}| < |\{D_\pi(A_1) \mid \pi \in \Sigma^+\} \times \{D_\pi(A_2) \mid \pi \in \Sigma^+\} \cup \{(A_1, A_2)\} \setminus \Gamma|$ for all $a \in \Sigma$, and therefore the induction hypothesis yields that $P_{a_i} = \mathbf{RegEq}(\Gamma', D_{a_i}(A_1), D_{a_i}(A_2), wa_i)$ terminates for all $i = 1 \dots n$, and therefore **RegEq** (Γ, A_1, A_2, w) returns **Eq-Diff** $(P_{a_1}, \dots, P_{a_n})$. ■

8.3 Correctness

We will now prove that if the given sets are semantically equal, then **RegEq** returns a proof of their equality. This is done by contraposition, so we actually prove that if **RegEq** doesn't return a proof of their equality then the given terms are not semantically equal.

Because we have already proved that **RegEq** terminates, it is sufficient to prove that if **RegEq** raises an exception then the given terms are not semantically equal.

Lemma 8.3 If **RegEq** (Γ, A_1, A_2, w) raises **NotEq** (w') then $w' = ww''$ and if $w'' = \varepsilon$ then $o(A_1) \neq o(A_2)$ and otherwise $o(D_{w''}(A_1)) \neq o(D_{w''}(A_2))$.

We prove this by induction on the depth of the calltree **RegEq** (Γ, A_1, A_2, w) .

Start: **RegEq** raises **NotEq** (w') directly.

This can only be done if $o(A_1) \neq o(A_2)$ and since w is raised, we get that $w'' = \varepsilon$, so the lemma is true.

Induction Hypothesis:

If the recursive call $P_{a_i} = \mathbf{RegEq}(\Gamma', A'_1, A'_2, w''')$ raises **NotEq** (w') then we can assume that $w' = w'''w''''$ and if $w'''' = \varepsilon$ then $o(A'_1) \neq o(A'_2)$ and otherwise $o(D_{w''''}(A'_1)) \neq o(D_{w''''}(A'_2))$.

Step: **RegEq** raises **NotEq** (w') in a recursive call.

This has to occur in one of the recursive calls $P_{a_i} = \mathbf{RegEq}(\Gamma', D_{a_i}(A_1), D_{a_i}(A_2), wa_i)$.

Therefore the induction hypothesis yields that $w' = wa_iw''''$ and if $w'''' = \varepsilon$ then $o(D_{a_i}(A_1)) \neq o(D_{a_i}(A_2))$ and otherwise $o(D_{w''''}(D_{a_i}(A_1))) \neq o(D_{w''''}(D_{a_i}(A_2)))$.

Since $w' = wa_iw''''$ and $w' = ww''$ we get that $w'' = a_iw''''$.

Therefore we only need to prove that $o(D_{a_iw''''}(A_1)) \neq o(D_{a_iw''''}(A_2))$.

If $w'''' = \varepsilon$ then we have that $o(D_{a_iw''''}(A_1)) = o(D_{a_i}(A_1)) \neq o(D_{a_i}(A_2)) = o(D_{a_iw''''}(A_2))$.

Otherwise we have that $o(D_{a_iw''''}(A_1)) = o(D_{w''''}(D_{a_i}(A_1))) \neq o(D_{w''''}(D_{a_i}(A_2))) = o(D_{a_iw''''}(A_2))$. ■

Theorem 8.4 Completeness of **RegEq**

If **RegEq** $(\{\}, A_1, A_2, \varepsilon)$ raises **NotEq** (w) then $\vDash A_1 \neq A_2$.

Assume that **RegEq** $(\{\}, A_1, A_2, \varepsilon)$ raises **NotEq** (w) .

If $w = \varepsilon$ then Lemma 8.3 yields that $o(A_1) \neq o(A_2)$ and therefore Corollary 2.12 yields that $\mathcal{L}(A_1) \neq \mathcal{L}(A_2)$ so $\vDash A_1 \neq A_2$.

If $w \neq \varepsilon$ then Lemma 8.3 yields that $o(D_w(A_1)) \neq o(D_w(A_2))$ and therefore Corollary 2.12

yields that $\mathcal{L}(D_w(A_1)) \neq \mathcal{L}(D_w(A_2))$ and therefore Corollary 3.8 yields that $\mathcal{L}(A_1) \neq \mathcal{L}(A_2)$ so $\vDash A_1 \neq A_2$. ■

Now we only need to prove that if **RegEq** returns a proof of equality, then the given sets are semantically equal. Because we have proved soundness of the proof-system, we only need to prove that the returned proof is a correct proof, and that it concludes the desired.

Theorem 8.5 *Soundness of **RegEq*** If **RegEq** (Γ, A_1, A_2, w) returns with a proof, then that proof is a correct proof of $\Gamma \vdash A_1 = A_2$.

We prove this by induction on the depth of the calltree **RegEq** (Γ, A_1, A_2, w) .

Start: **RegEq** (Γ, A_1, A_2, w) returns directly.

This can only happen if $(A_1, A_2) \in \Gamma$ and therefore **Eq-Hyp** is a correct proof of $\Gamma \vdash A_1 = A_2$.

Induction Hypothesis:

If **RegEq** (Γ, A_1, A_2, w) calls $P = \mathbf{RegEq}(\Gamma', A'_1, A'_2, w')$, then P is set to a correct proof of $\Gamma' \vdash A'_1 = A'_2$.

Step: **RegEq** (Γ, A_1, A_2, w) makes recursive calls.

In the case where $(A_1, A_2) \notin \Gamma$ then Γ' is set to $\Gamma \cup \{(A_1, A_2)\}$ and the recursive calls $P_{a_i} = \mathbf{RegEq}(\Gamma', D_{a_i}(A_1), D_{a_i}(A_2), w_{a_i})$ are made for $i = 1, 2, \dots, n$ so the induction hypothesis yields that P_{a_i} is a correct proof of $\Gamma' \vdash D_{a_i}(A_1) = D_{a_i}(A_2)$ and therefore **Eq-Diff** $(P_{a_1}, P_{a_2}, \dots, P_{a_n})$ is a correct proof of $\Gamma \vdash A_1 = A_2$. ■

9 Conclusion

In this section, we will briefly sum up what we have done in this project, and how this work could be extended.

9.1 What has been solved

In this project, we have defined the basic theory of regular expressions and their semantics, and extended it to sets of regular expressions.

We have used this to define semantical equality on sets of regular expressions.

We have introduced a co-inductive axiomatization of equality of sets of regular expressions.

By using a stratified interpretation of soundness, we have proved that the axiomatization is sound.

By using bi-simulations, we have proved that the axiomatization is complete.

This yields an algorithm which we have documented, implemented in SML and tested.

9.2 Future work

We have used differentiation of sets of regular expressions to create an axiomatization of equality, but it could probably be used for DFA generation as well.

The idea is that if two sets are semantically equal, they can be represented by the same DFA's, and a proof in the axiomatization represents one of these DFA's. That DFA could be obtained by converting the proof in the following way.

The DFA has a node for every sub derivation in the proof, and an edge from each node to all of the nodes representing its sub derivations. The letter for each edge is the letter that is used for the differentiation, and the trick is that all edges pointing to a node representing an Eq-Hyp rule must be re-directed to the node representing the sub derivation where the used assumption is added to the Γ .

It should be noted, that this method doesn't necessarily produce a minimal DFA.

Even though we have proved the axiomatization to be complete, we have no upper limit on the size of the proofs. The analysis we use to prove that the axiomatization is complete could probably be improved to yield an upper limit, but only a very poor one. The proofs we have found in this axiomatization have had an acceptable size, so there is reason to believe that it is possible to find a good upper limit. One way of pursuing this result could be to limit the size of the DFA found by the above method.

We have defined an axiomatization of equality, but it should be straightforward to convert it to an axiomatization of ordering, representing the semantics that $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ simply by changing the side condition of the Eq-Diff rule to $o(A_1) = 1 \Rightarrow o(A_2) = 1$.

Usually when considering a co-inductive axiomatization it leads to an interpretation as coercions, but it is not obvious how this should be done in this case. One way could be that a word in the language of a regular expression should be paired with a path describing how the expression is used to produce that word, and the coercions could then transform a path for one regular expression to a path for the other regular expression producing the same word.

A Program

Since the result of the analysis is a rather simple program, we have had the opportunity to actually implement and test this program.

We have chosen to implement the program in SML, because it has the necessary features such as exceptions and easy implementation of sets, while allowing the code to be compact and easy to read.

If we had chosen to write the program in twelf, the superior typing system would have been able to confirm that the resulting proofs are correct proofs concluding the desired, but it is unlikely that it would be able to conclude that the function actually terminates, without implementing most of our project in twelf.

A.1 The implementation

```

(* The signature of a set *)
signature SetSig =
sig
  type element
5   type set
    val empty : set
    val member : element -> set -> bool
    val subset : set -> set -> bool
    val equal : set * set -> bool
10  val insert : element * set -> set
    val union : set -> set -> set
    val toList : set -> element list
    val fromList : element list -> set
    val toString : set -> string
15 end
(* A functor for setstructure generation *)
signature EQ =
sig
  type element
20  val eq : element * element -> bool
    val elmToString : element -> string
end
functor SetFct(s : EQ) :> SetSig where type element = s.element =
struct
25  type element = s.element
    type set = element list
    val empty = []
    fun member a = List.exists (fn y => s.eq(a,y))
    fun subset xs ys = List.all (fn x => member x ys) xs
30  fun equal (s1,s2) = subset s1 s2 andalso subset s2 s1
    fun insert (a,s) = if member a s
                        then s
                        else a::s

```

```

    val union = foldl insert
35  val fromList = foldl insert empty
    fun toList s = s
    fun toString xs = "{ "
                        ^ (foldl (fn (x,b) => s.elmToString x ^ ", " ^ b) "" xs)
                        ^ " }"
40  end
    (* Defining regular expressions *)
    datatype reg = R0
                | R1
                | Rletter of char
45  | Rseq of reg * reg
                | Rsum of reg * reg
                | Rstar of reg
    (* Pretty printing of regular expressions *)
    fun regToString R0 = "0"
50  | regToString R1 = "1"
    | regToString (Rletter ch) = Char.toString ch
    | regToString (Rseq (a,b)) = "("
                                ^ (regToString a)
                                ^ (regToString b)
55  ^ ")"
    | regToString (Rsum (a,b)) = "("
                                ^ (regToString a)
                                ^ " + "
                                ^ (regToString b)
60  ^ ")"
    | regToString (Rstar a) = "("
                                ^ (regToString a)
                                ^ ")*"
    (* Defining sets of regular expressions *)
65  structure RegEq : EQ =
    struct
        type element = reg
        val eq = op =
        val elmToString = regToString
70  end
    (* Set structure for regular expression *)
    structure SetReg = SetFct(RegEq)
    structure RegEnv : EQ =
    struct
75  type element = SetReg.set * SetReg.set
        fun eq ((A,A'),(B,B')) = SetReg.equal (A,B)
                                andalso SetReg.equal (A',B')
        fun elmToString (A,B) = "("
                                ^ (SetReg.toString A)
80  ^ ", "

```

```

      ^ (SetReg.toString B)
      ^ ")”
end
(* Set structure for environments, ie.
85   set structure for pairs of sets of regular expressions *)
structure SetEnv = SetFct(RegEnv)
(* The Empty Word Property *)
fun ewp R1 = R1
    | ewp R0 = R0
90   | ewp (Rseq (a,b)) = if ewp a = R1 andalso ewp b = R1
                        then R1
                        else R0
    | ewp (Rsum (a,b)) = if ewp a = R1 orelse ewp b = R1
                        then R1
95   else R0
    | ewp (Rstar _) = R1
    | ewp (Rletter _) = R0
fun EWP A = if List.exists (fn E => ewp E = R1) (SetReg.toList A)
            then R1
100          else R0
(* Combine a reg exp with a set of reg exps *)
fun SEQ1 R1 s = s
    | SEQ1 R0 _ = SetReg.empty
    | SEQ1 E s = foldl (fn (R0,A) => A
105   | (R1,A) => SetReg.insert (E ,A)
    | (E' ,A) => SetReg.insert (Rseq(E, E') ,A))
                        SetReg.empty
                        (SetReg.toList s)
(* Combine a set of reg exps with a reg exp *)
110 fun SEQ2 s R1 = s
    | SEQ2 s R0 = SetReg.empty
    | SEQ2 s E = foldl (fn (R0,A) => A
    | (R1,A) => SetReg.insert (E ,A)
    | (E' ,A) => SetReg.insert (Rseq(E' , E) ,A))
115   SetReg.empty (SetReg.toList s)
(* Differentiation *)
local
    fun diff _ R0 = SetReg.empty
    | diff _ R1 = SetReg.empty
120   | diff c (Rletter b) = if b = c
                        then SetReg.insert (R1, SetReg.empty)
                        else SetReg.empty
    | diff c (Rsum (a,b)) = SetReg.union (diff c a) (diff c b)
    | diff c (Rseq (a,b)) = SetReg.union (SEQ2 (diff c a) b)
125   | diff c (Rstar a) = SEQ2 (diff c a) (Rstar a)
in

```

```

    fun DIFF c s = foldl (fn (E,A) => SetReg.union (diff c E) A)
                        SetReg.empty (SetReg.toList s)
130 end
    exception NotEq of string
    datatype proof = EqHyp | EqDiff of (char * proof) list

    (* The algorithm *)
135 fun RegEq G A1 A2 w =
        if EWP A1 <> EWP A2
        then raise NotEq w
        else if SetEnv.member (A1,A2) G
            then EqHyp
140         else let val G' = SetEnv.insert ((A1,A2), G)
                in
                    let val Pa = RegEq G' (DIFF #"a" A1)
                        (DIFF #"a" A2)
                        (implode ((explode w) @ [#"a"]))
145                 val Pb = RegEq G' (DIFF #"b" A1)
                        (DIFF #"b" A2)
                        (implode ((explode w) @ [#"b"]))
                    in EqDiff [(#"a",Pa), (#"b",Pb)]
                    end
                end
150         end

    (* Defining constructors and equality-operator *)
    infix 6 ++
    fun A ++ B = Rsum (A, B)
155 infix 7 ^^
    fun A ^^ B = Rseq (A, B)
    fun ** A = Rstar A
    infix 4 ==
    fun A == B = let
160         val x = RegEq SetEnv.empty
                (SetReg.insert (A, SetReg.empty))
                (SetReg.insert (B, SetReg.empty))
                ""
            in
165         true
            end
        handle NotEq w => false

    (* Defining the regular expressions *)
    val a = (Rletter #"a") (* a *)
170 val b = (Rletter #"b") (* b *)
    val ab = a ^^ b (* ab *)
    val ba = b ^^ a (* ba *)
    val E1 = a ++ a (* a+a *)
    val E2 = a ++ b (* a+b *)

```

```

175 val E3 = b ++ a (* b+a *)
    val E4 = a ^^ b ^^ ** (a ^^ b) (* ab(ab)* *)
    val E5 = a ^^ ** (b ^^ a) ^^ b (* a(ba)*b *)
    val E6 = ** (a ^^ b) (* (ab)* *)
    val bb = b ^^ b
180 val bbb = b ^^ b ^^ b
    val E7 = a ^^ (b ++ ** bb ++ ** bbb) ^^ a
          (* a(b+(bb)*+(bbb)* )a *)
    val E8 = a ^^ ** b ^^ a (* ab*a *)
    val E9 = a ^^ ** (bb ++ bbb) ^^ a (* a(bb+bbb)*a *)
185 val E10 = a ^^ (R1 ++ b ^^ b ^^ ** b) ^^ a (* a(1+bbb* )a *)
    (* Test execution *)
    val test0 = (R0 == R1) = false (* w = "" *)
    val test1 = (a == a) = true
    val test2 = (b == b) = true
190 val test3 = (a == b) = false (* w = "a" *)
    val test4 = (b == a) = false (* w = "a" *)
    val test5 = (ab == ab) = true
    val test6 = (ba == ab) = false (* w = "ab" *)
    val test7 = (E1 == a) = true
195 val test8 = (E2 == E3) = true
    val test9 = (E4 == E5) = true
    val test10 = (R1 ++ E5 == E6) = true
    val test11 = (E5 == E6) = false (* w = "" *)
    val test12 = (E7 == E8) = false (* w = "abbbba" *)
200 val test13 = (E9 == E10) = true;
    quit();

```

A.2 The test results

The test cases, and the expected results are found in the final part of the code.

The tests are of the form

$$\text{val result} = (A == B) = C$$

where A and B are the expressions to test, and C is the expected result. If we don't expect the given terms to be found equal, we have written in a comment, which string we expect `RegEq` to raise in an exception.

All the cases return the expected result.

B Literature

References

- [1] [MBFH98] Coinductive Axiomatization of Recursive Type Equality and Subtyping, *Fundamenta Informaticae* 33
Michael Brandt and Fritz Henglein
ISSN: 0169-2968
Printed in the Netherlands
- [2] [MN06] Coinductive Axiomatizations
Michael Nissen
<http://ln.omk.dk/download/nissen.pdf>
- [3] [LN06] Coinductive Axiomatizations Continued
Lasse Nielsen
<http://ln.omk.dk/download/nielsen.pdf>
- [4] [CG05] Relating Proof Systems for Recursive Types
Clemens Grabmeyer
ISBN: 90-9019086-4
PrintPartners Ipskamp, Enschede, the Netherlands
- [5] [JHC71] Regular Algebra and Finite Machines
J. H. Conway
ISBN: 0-412-10620-5
Printed in GB by William Clowes & Sons Ltd
- [6] [RM82] A Complete Inference System for a Class of Regular Behaviors
Robin Milner
Department of Computer Science, University of Edinburgh
- [7] [AS66] Two Complete Axiom Systems for the Algebra of Regular Events
Arto Salomaa
University of Turku, Finland
- [8] [BCP02] Types and programming languages
Benjamin C. Pierce
ISBN: 0-262-16209-1
The MIT Press
- [9] [GW93] The formal semantics of programming languages
Glynn Winskel
ISBN: 0-262-23169-7
The MIT press