The Inclusion Problem for Regular Expressions☆

Dag Hovland[1]

Department of Informatics, University of Bergen, Norway

Abstract

This paper presents a polynomial-time algorithm for the inclusion problem for a large class of regular expressions. The algorithm is not based on construction of finite automata, and can therefore be faster than the lower bound implied by the Myhill-Nerode theorem. The algorithm automatically discards irrelevant parts of the right-hand expression. The irrelevant parts of the right-hand expression might even be 1-ambiguous. For example, if r is a regular expression such that any DFA recognizing r is very large, the algorithm can still, in time independent of r, decide that the language of ab is included in that of $(a + r)b$. The algorithm is based on a syntax-directed inference system. It takes arbitrary regular expressions as input. If the 1-ambiguity of the right-hand expression becomes a problem, the algorithm will report this. Otherwise, it will decide the inclusion problem for the input.

Keywords: formal languages, regular expressions, inclusion, 1-unambiguity

1. Introduction

The inclusion problem for regular expressions was shown PSPACE-complete in Meyer & Stockmeyer [1]. The input to the problem consists of two expressions, the *left-hand* expression and the *right-hand* expression, respectively. The question is whether the language of the left-hand expression is included in the language of the right-hand expression. The classical algorithm starts with constructing non-deterministic finite automata (NFAs) for each of the expressions, then constructs a DFA from the NFA recognizing the language of the right-hand expression, and a DFA recognizing the complement of this language. Then an NFA recognizing the intersection of the language of the left-hand expression with the complement of the language of the right-hand expression is constructed. Finally, the algorithm checks that no final state is reachable in the latter NFA. The super-polynomial blowup occurs when constructing a DFA from the NFA recognizing the right-hand expression. A lower bound to this blowup is given by the Myhill-Nerode theorem [2, 3]. All the other steps, seen separately, are polynomial-time.

1-Unambiguous regular expressions were first used in SGML [4], and first formalized and studied by Brüggemann-Klein & Wood [5, 6]. The latter show a polynomial-time construction of DFAs from 1-unambiguous regular expressions. The classical algorithm can therefore be modified to solve the inclusion problem in polynomial time when the right-hand expression is 1-unambiguous. This paper presents an alternative algorithm for inclusion of 1-unambiguous regular expressions. As in the modified classical algorithm, the left-hand expression can be an arbitrary regular expression. If the right-hand expression is 1-unambiguous, the algorithm is guaranteed to decide the inclusion problem, while if it is not 1-unambiguous (i.e., the expression is *1-ambiguous*), it might either decide the problem correctly, or report that the 1-ambiguity is a problem. An implementation of the algorithm is available from the website of the

☆An extended abstract of this paper appeared in the Proceedings of the 4th International Conference on Language and Automata Theory and Applications (LATA 2010). An earlier version of this paper appeared in the PhD thesis "Feasible Algorithms for Semantics — Employing Automata and Inference Systems", Dag Hovland, University of Bergen, 2010.

doi:10.1016/j.jcss.2011.12.003

Email address: hovlanddag@gmail.com (Dag Hovland)

[1]Present address: Department of Informatics, University of Oslo, Postboks 1080 Blindern, N-0316 Oslo, Norway

author. The algorithm can of course also be run twice to test whether the languages of two 1-unambiguous regular expressions are equal.

A consequence of the Myhill-Nerode theorem is that for many regular expressions, the minimal DFA recognizing this language, is of super-polynomial size. For example, there are no polynomial-size DFAs recognizing expressions of the form $(b + c)^*c(b + c)\cdots(b + c)$. An advantage of the algorithm presented in this paper is that it only treats the parts of the right-hand expression which are necessary; it is therefore sufficient that these parts of the expression are 1-unambiguous. For some expressions, it can therefore be faster than the modified classical algorithm above. For example, the algorithm described in this paper will (in polynomial time) decide that the language of ab is included in that of $(a + (b + c)^*c(b + c)\cdots(b + c))b$, and the sub-expression $(b + c)^*c(b + c)\cdots(b + c)$ will be discarded. The polynomial-time version of the classical algorithm cannot easily be modified to handle expressions like this, without adding complex and ad hoc pre-processing.

To summarize: Our algorithm always terminates in polynomial time. If the right-hand expression is 1-unambiguous, the algorithm will return a positive answer if and only if the expressions are in an inclusion relation, and a negative answer otherwise. If the right-hand expression is 1-ambiguous, three outcomes are possible: The algorithm might return a positive or negative answer, which is then guaranteed to be correct, or the algorithm might also decide that the 1-ambiguity of the right-hand expression is a problem, report this, and terminate.

Section 2 defines operations on regular expressions and properties of these. Section 3 describes the algorithm for inclusion, and Section 4 shows some important properties of the algorithm. The last section covers related work and a conclusion.

2. Regular Expressions

Fix an *alphabet* Σ of *letters*. Assume a, b, and c are members of Σ. $l, l_1, l_2, \ldots$ are used as variables for members of Σ.

Definition 2.1 (Regular Expressions)**.** *The* regular expressions *over the language* Σ *are denoted* R_Σ *and defined in the following inductive manner:*

$$R_\Sigma ::= R_\Sigma + R_\Sigma \mid R_\Sigma \cdot R_\Sigma \mid R_\Sigma^* \mid \Sigma \mid \epsilon$$

We use $r, r_1, r_2, \ldots$ *as variables for regular expressions. Concatenation is right-associative, such that, e.g.,* $r_1 \cdot r_2 \cdot r_3 = r_1 \cdot (r_2 \cdot r_3)$*. The sign for concatenation,* $\cdot$*, will often be omitted. The star has highest precedence, followed, in order, by concatenation and choice. A regular expression denoting the empty language is not included, as this is irrelevant to the results in this paper. We denote the set of letters from* Σ *occurring in* r *by* $\mathsf{sym}(r)$*.*

The semantics of regular expressions is defined in terms of sets of words over the alphabet Σ. We lift concatenation of words to sets of words, such that if $L_1, L_2 \subseteq \Sigma^*$, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}$. ϵ denotes the *empty word* of zero length, such that for all $w \in \Sigma^*$, $\epsilon \cdot w = w \cdot \epsilon = w$. Integer exponents are short-hand for repeated concatenation of the same set, such that for a set L of words, e.g., $L^2 = L \cdot L$, and we define $L^0 = \{\epsilon\}$.

Definition 2.2 (Language of a Regular Expression)**.** *The* language *of a regular expression* r *is denoted* $\|r\|$ *and is defined by the following inductive rules:* $\|r_1 + r_2\| = \|r_1\| \cup \|r_2\|$, $\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\|$, $\|r^*\| = \bigcup_{0 \leq i} \|r\|^i$ *and for* $a \in \Sigma \cup \{\epsilon\}$, $\|a\| = \{a\}$.

All subexpressions of the forms $\epsilon \cdot r$, $\epsilon + \epsilon$ or ϵ^* can be simplified to r, ϵ, or ϵ respectively, in linear time, working bottom up. We will often tacitly assume there are no subexpressions of these forms. Furthermore, we use r^i as a short-hand for r concatenated with itself i times.

Definition 2.3 (Nullable Expressions)**.** *[7, 8] The* nullable *regular expressions are denoted* $\mathfrak{N}_\Sigma$ *and are defined inductively as follows:*

$$\mathfrak{N}_\Sigma ::= \mathfrak{N}_\Sigma + R_\Sigma \mid R_\Sigma + \mathfrak{N}_\Sigma \mid \mathfrak{N}_\Sigma \cdot \mathfrak{N}_\Sigma \mid R_\Sigma{}^* \mid \epsilon$$

The nullable expressions are exactly those denoting a language containing the empty word. Proofs of the following lemma, and other lemmas in this section, can be found in Appendix A.

Table 1: The first-set of a regular expression

$\mathsf{first}(\epsilon) = \varnothing$,	$r \in \Sigma \Rightarrow \mathsf{first}(r) = \{r\}$
$r = r_1 + r_2 \Rightarrow \mathsf{first}(r) = \mathsf{first}(r_1) \cup \mathsf{first}(r_2)$	
$r = r_1 \cdot r_2 \wedge r_1 \in \mathfrak{N}_\Sigma \Rightarrow \mathsf{first}(r) = \mathsf{first}(r_1) \cup \mathsf{first}(r_2)$	
$r = r_1 \cdot r_2 \wedge r_1 \notin \mathfrak{N}_\Sigma \Rightarrow \mathsf{first}(r) = \mathsf{first}(r_1)$	
$r = r_1^* \Rightarrow \mathsf{first}(r) = \mathsf{first}(r_1)$	

Lemma 2.4. *For all regular expressions $r \in R_\Sigma$, $\epsilon \in \|r\| \Leftrightarrow r \in \mathfrak{N}_\Sigma$.*

Intuitively, the first-set of a regular expression is the set of letters that can occur first in a word in the language. An inductive definition of the first-set is given in Table 1. Similar definitions have been given by many others, e.g., Glushkov [7] and Yamada & McNaughton [8].

Lemma 2.5 (first)**.** *[7, 8] For any regular expression r,* $\mathsf{first}(r) = \{l \in \Sigma \mid \exists w : lw \in \|r\|\}$ *and* $\mathsf{first}(r)$ *can be calculated in time $O(|r| \cdot |\Sigma|)$. (Where $|r|$ is the length of r, and $|\Sigma|$ is the size of the alphabet.)*

The followLast-set of a regular expression is the set of letters which can follow a word in the language.

Definition 2.6 (followLast)**.** *[5]*

$$\mathsf{followLast}(r) = \{l \in \mathsf{sym}(r) \mid \exists u, v \in \mathsf{sym}(r)^* : (u \in \|r\| \wedge ulv \in \|r\|)\}$$

To limit the number of rules in the inference system explained in Section 3, we will put regular expressions on *header-form*.

Definition 2.7 (Header-form)**.** *A regular expression is in header-form if it is of the form ϵ, $l \cdot r_1$, $(r_1 + r_2) \cdot r_3$ or $r_1^* \cdot r_2$, where $l \in \Sigma$ and $r_1, r_2, r_3 \in R_\Sigma$.*

A regular expression can in linear time be put in header-form without changing the denoted language by applying the mapping hdf. We need the auxiliary mapping header, which maps a pair of regular expressions to a single regular expression. It is defined by the following inductive rules:

$$\mathsf{header}(\epsilon, r) = r$$

$$\mathsf{header}(r_1, r_2) = \begin{cases} (\text{write } r_1 \text{ as } r_1' \cdots r_n' \text{ for } n \geq 1, \text{ where } r_n' \text{ is not a concatenation}) & \\ r_1 \cdot r_2 & \text{if } n = 1 \\ \mathsf{header}(r_1', r_2) & \text{if } n = 2, r_2' = \epsilon \\ \mathsf{header}(r_1', r_2' \cdots r_{n-1}' \cdot r_2) & \text{if } n > 2, r_n' = \epsilon \\ \mathsf{header}(r_1', r_2' \cdots r_n' \cdot r_2) & \text{if } n \geq 2, r_n' \neq \epsilon \end{cases}$$

We can now define $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$.

Example 2.8. $\mathsf{hdf}(a) = a\epsilon$, $\mathsf{hdf}((cd)e) = c(d(e\epsilon))$, $\mathsf{hdf}((ab)((cd)e)) = a(b((cd)(e\epsilon)))$.

The reader may wonder why we do not simply require all expressions to be written in the right-associative form. That is, why do we not simply assume some linear-time transformation which gets rid of all subexpressions of the form $(r_1 \cdot r_2) \cdot r_3$? The problem is that the main algorithm shown below will construct new expressions by concatenating subexpressions of the original expression. The constructed expressions might not be on the required right-associative form, and we would need to run the transformation again. This transformation would then change the expressions, which creates problems for proving the polynomial runtime of the algorithm. The proofs of polynomial runtime of the algorithm are sensitive to the details of the mapping hdf, and we must therefore treat hdf in some detail. We summarize the basic properties of the mappings hdf and header in the following lemma:

Lemma 2.9. *For any regular expression r:*

1. $\mathsf{hdf}(r)$ *is in header-form,*
2. $\|\mathsf{hdf}(r)\| = \|r\|$,
3. $\exists n \geq 0, r_1, \ldots, r_n \in R_\Sigma - \{\epsilon\} : \mathsf{hdf}(r) = r_1 \cdots r_n \cdot \epsilon$.
4. $\mathsf{hdf}(\mathsf{hdf}(r)) = \mathsf{hdf}(r)$.

2.1. Term Trees and Positions

Given a regular expression r, we follow Terese [9] and define the term tree of r as the tree where the root is labeled with the main operator (choice, concatenation, or star) and the subtrees are the term trees of the subexpression(s). If $a \in \Sigma \cup \{\epsilon\}$ the term tree is a single root-node with a as label.

We use $\langle n_1, \ldots, n_k \rangle$, a possibly empty sequence of natural numbers, to denote a position in a term tree. We let p, q, including subscripted variants, be variables for such possibly empty sequences of natural numbers. The position of the root is $\langle\rangle$. If $r = r_1 \cdot r_2$ or $r = r_1 + r_2$, and $n_1 \in \{1, 2\}$, the position $\langle n_1, \ldots, n_k \rangle$ in r is the position $\langle n_2, \ldots, n_k \rangle$ in the subtree of child n_1, that is, in the term tree of r_{n_1}. If $r = {r_1}^*$, the position $\langle 1, n_1, \ldots, n_k \rangle$ in r is the position $\langle n_1, \ldots, n_k \rangle$ in the term tree of r_1. Let $\mathsf{pos}(r)$ be the set of positions in r.

For two positions $p = \langle m_1, \ldots, m_k \rangle$ and $q = \langle n_1, \ldots, n_l \rangle$, the notation $p \downarrow q$ will be used for the concatenated position $\langle m_1, \ldots, m_k, n_1, \ldots, n_l \rangle$. We will also use this notation for lists of positions, so if $p, p_1, \ldots, p_n$ are positions, then $p \downarrow (p_1 \cdot \cdots \cdot p_n) = (p \downarrow p_1) \cdot \cdots \cdot (p \downarrow p_n)$. Further, we use the notation for concatenating a position with each element of a set consisting of lists of positions, such that if p is a position, and S is a set of lists of positions, then $p \downarrow S = \{p \downarrow q \mid q \in S\}$.

Below we will encounter regular expressions whose alphabet are sets of positions. Concatenating a position with such an expression is defined by concatenating the position with all the positions occurring in the expression. Note that the language of such a regular expression is a set of lists of positions. Hence, for p a position, $r_1 \in R_\Sigma$, and $r \in R_{\mathsf{pos}(r_1)}$, $\|p \downarrow r\| = p \downarrow \|r\|$. Concatenation with a position has highest precedence, such that, e.g., $p \downarrow r_1 r_2 = (p \downarrow r_1) r_2$. Whenever concatenating with a position of length one, we will often omit the angular braces, such that for example $p \downarrow 1 = p \downarrow \langle 1 \rangle$, $2S = \langle 2 \rangle \downarrow S$, $i \downarrow r = \langle i \rangle \downarrow r$, etc.

For a position p in r we will denote the subexpression rooted at this position by $r[p]$. Note that $r[\langle\rangle] = r$. $r[]$ can be seen as a mapping from positions to regular expressions. There is an easy way to lift this into a mapping from strings of positions to regular expressions: Given $w \in \mathsf{pos}(r)^*$, put $r[w] = \epsilon$ if $w = \epsilon$, and otherwise put $r[w] = r[p_1] \cdot \cdots \cdot r[p_n]$, where $w = p_1 \cdots p_n$ for some $p_1, \ldots, p_n \in \mathsf{pos}(r)$. Lastly, we lift $r[]$ to sets of string, such that if $S \subseteq \mathsf{pos}(r)^*$, then $r[S] = \{r[w] \mid w \in S\}$.

Note that for $r \in R_\Sigma$, $p \in \mathsf{pos}(r)$, and $q \in \mathsf{pos}(r[p])$, we have $r[p \downarrow q] = r[p][q]$. This can be shown by induction on $r[p]$ (see, e.g., Terese [9]). For example in the case of $r[p] = r_1 \cdot r_2$, we have that q is a position in either r_1 or r_2. Assume it is in r_1, then $q = 1 \downarrow q'$ for some $q' \in \mathsf{pos}(r_1)$. As $r[p][\langle 1 \rangle] = r_1 = r[p \downarrow 1]$ we get that $r[p][1 \downarrow q'] = r[p \downarrow 1][q']$, and by the induction hypothesis $r[p \downarrow 1][q'] = r[p \downarrow 1 \downarrow q']$.

The concept of *marked expressions* will be important in this paper. It was first used in a similar context by Brüggemann-Klein & Wood [6]. The intuition is that the marked expression is the expression where every instance of any symbol from Σ is substituted by its position in the expression.

Example 2.10. *Consider* $\Sigma = \{a, b, c\}$ *and* $r = a + bc$. *Then* $\mu(r) = \langle 1 \rangle + \langle 2, 1 \rangle \cdot \langle 2, 2 \rangle$. *The term trees of r and* $\mu(r)$ *are shown in Fig. 1.*

Definition 2.11 (Marked Expressions)**.** *If* $r \in R_\Sigma$ *is a regular expression,* $\mu(r) \in R_{\mathsf{pos}(r)}$ *is the marked expression, defined in the following inductive manner:*

- $\mu(\epsilon) = \epsilon$
- *for* $l \in \Sigma$, $\mu(l) = \langle\rangle$
- $\mu(r_1 + r_2) = 1 \downarrow \mu(r_1) + 2 \downarrow \mu(r_2)$

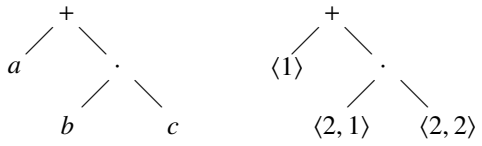

Figure 1: Term trees for $a + bc$ and $\mu(a + bc) = \langle 1\rangle + \langle 2, 1\rangle\langle 2, 2\rangle$

- $\mu(r_1 \cdot r_2) = 1 \downarrow \mu(r_1) \cdot 2 \downarrow \mu(r_2)$
- $\mu(r_1^*) = (1 \downarrow \mu(r_1))^*$

Note that, e.g., $\mu(b) = \mu(a) = \langle\rangle$, which shows that marking is not injective. Furthermore $\|\mu(r_1 \cdot r_2)\| = 1 \downarrow \|\mu(r_1)\| \cdot 2 \downarrow \|\mu(r_2)\|$, $\|\mu(r_1 + r_2)\| = 1 \downarrow \|\mu(r_1)\| \cup 2 \downarrow \|\mu(r_2)\|$, and $\|\mu(r^*)\| = 1 \downarrow \|\mu(r)^*\|$. The following lemma will often be used tacitly.

Lemma 2.12. *For any regular expression r,*

1. $\|r\| = r[\|\mu(r)\|]$
2. *For any* $p \in \mathsf{sym}(\mu(r))$, $\mu(r)[p] = p$.
3. *For any* $p \in \mathsf{pos}(r)$, $r[p] \in \Sigma$ *iff* $p \in \mathsf{sym}(\mu(r))$.

2.2. 1-Unambiguous Regular Expressions

Definition 2.13 (Star Normal Form)**.** *[5, 6]: A regular expression is in* star normal form *iff for all subexpressions* r^**:* $r \notin \mathfrak{N}_\Sigma$ *and* $\mathsf{first}(\mu(r)) \cap \mathsf{followLast}(\mu(r)) = \varnothing$.

Brüggemann-Klein & Wood described also in [5, 6] a linear time algorithm mapping a regular expression to an equivalent expression in star normal form. We will therefore often tacitly assume that all regular expressions are in star normal form.

It is almost immediate that hdf preserves star normal form, as starred subexpressions are not altered.

Definition 2.14. *[5, 6] A regular expression r is 1-unambiguous if for any two* $upv, uqw \in \|\mu(r)\|$*, where* $p, q \in \mathsf{sym}(\mu(r))$ *(i.e.,* $r[p], r[q] \in \Sigma$*) and* $u, v, w \in \mathsf{sym}(\mu(r))^*$ *such that* $r[p] = r[q]$*, we have* $p = q$.

Examples of 1-unambiguous regular expressions are aa^* and $b^*a(b^*a)^*$, while $(\epsilon + a)a$ and $(a + b)^*a$ are not 1-unambiguous. The languages denoted by 1-unambiguous regular expressions will be called *1-unambiguous regular languages*. An expression which is not 1-unambiguous is called 1-ambiguous. Brüggemann-Klein & Wood [6] showed that there exist regular languages that are not 1-unambiguous regular languages, e.g. $\|(a + b)^*(ac + bd)\|$. However, the reverse of $(a + b)^*(ac + bd)$, namely $(ca + db)(a + b)^*$ is 1-unambiguous. There are of course also expressions like $(a + b)^*(ac + bd)(c + d)^*$, which denotes a 1-ambiguous language, read both backwards and forwards.

Brüggemann-Klein & Wood characterized the 1-unambiguous regular expressions in [6, Lemma 3.2]. The latter lemma implies that all subexpressions of a 1-unambiguous regular expression (in star normal form) are 1-unambiguous. Another important consequence is that if r_1 and r_2 are 1-unambiguous, and $\mathsf{first}(r_1) \cap \mathsf{first}(r_2) = \varnothing$, then $r_1 + r_2$ is 1-unambiguous. Lastly, $r_1 \cdot r_2$ is 1-unambiguous if r_1 and r_2 are 1-unambiguous, $r_1 \in \mathfrak{N}_\Sigma \Rightarrow \mathsf{first}(r_1) \cap \mathsf{first}(r_2) = \varnothing$, and $\mathsf{followLast}(r_1) \cap \mathsf{first}(r_2) = \varnothing$. The latter three facts will be used several times.

Lemma 2.15. *For a 1-unambiguous regular expression r,* $\mathsf{hdf}(r)$ *is also 1-unambiguous.*

1-unambiguity is different from, though related with, *unambiguity*, as used to classify grammars in language theory, and studied for regular expressions by Book et al. [10]. From [10]: "A regular expression is called unambiguous if every tape in the event can be generated from the expression in one way only"[2] It is not hard to see that the class of 1-unambiguous regular expressions is included in the class of unambiguous regular expressions.

[2] In modern language, "tape" is "word" and "event" is "language".

Lemma 2.16. *A 1-unambiguous regular expression is also unambiguous*

The inclusion is strict, as for example the expression $(a + b)^*a$ is both unambiguous and 1-ambiguous. See also [5, 6] for comparisons of unambiguity and 1-unambiguity.

3. Rules for Inclusion

The algorithm is based on an inference system described in Table 2, inductively defining a binary relation $\sqsubseteq$ between regular expressions. The core of the algorithm is a goal-directed, depth-first search using this inference system. We will show later that a pair of regular expressions is in the relation $\sqsubseteq$ if and only if their languages are in the inclusion relation.

We will say that $r_1 \sqsubseteq r_2$ *is sound*, if $\|r_1\| \subseteq \|r_2\|$. Each rule consists of a horizontal line with a conclusion below it, and zero, one, or two premises above the line. All rules but one also have *side-conditions* in square brackets. We only allow rule instances where the side-conditions are satisfied. This means that *matching* the conclusion of a rule implies satisfying the side-conditions. Note that (StarChoice1) and (LetterChoice) each have only one premise.

Input: Two regular expressions r_1 and r_2
Output: "Yes", "No" or "1-ambiguous"
Initialize stack T and set S as empty ;
push $(\mathsf{hdf}(r_1), \mathsf{hdf}(r_2))$ on T;
while T *not empty* **do**
  pop (r_3, r_4) from T;
  if $(r_3, r_4) \notin$ S **then**
    if $\mathsf{first}(r_3) \not\subseteq \mathsf{first}(r_4)$ *or* $r_3 \in \mathfrak{N}_\Sigma \wedge r_4 \notin \mathfrak{N}_\Sigma$ *or* $r_4 = \epsilon \wedge r_3 \neq \epsilon$ **then**
      return *"No"*;
    end
    if $r_3 \sqsubseteq r_4$ *matches conclusion of more than one rule instance* **then**
      return *"1-ambiguous"*;
    end
    add (r_3, r_4) to S;
    for *all premises* $r_5 \sqsubseteq r_6$ *of the rule instance where* $r_3 \sqsubseteq r_4$ *matches the conclusion* **do**
      push $(\mathsf{hdf}(r_5), \mathsf{hdf}(r_6))$ on T;
    end
  end
end
return *"Yes"*;

Figure 2: Algorithm for inclusion of regular expressions

Figure 2 describes the algorithm for deciding inclusion of regular expressions. The algorithm takes a pair of regular expressions as input, and if it returns "Yes" they are in an inclusion relation, if it returns "No" they are not, and if it returns "1-ambiguous", the right-hand expression is 1-ambiguous. The stack T is used for a depth-first search, while the set S keeps track of already treated pairs of regular expressions. Both S and T consist of pairs of regular expressions.

Figures 3, 4, 5, and 6 show examples of how to use the inference rules. The example noted in the introduction, deciding whether $\|ab\| \subseteq \|(a + (b + c)^*c(b + c) \cdots (b + c))b\|$ is shown in Fig. 6. Note that branches end either in an instance of the rule (Axm), usage of the store of already treated relations, or a failure. In addition to correctness of the algorithm, termination is of course of paramount importance. It is natural to ask how the algorithm possibly can terminate, when the rules (LetterStar), (LeftStar), and (StarChoice2) have more complex premises than conclusions. This will be answered in the next section.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\textit{Store}(1)}}{\textsf{(Letter)}\ \ 5 : a^*b^* \sqsubseteq (a+b)^*}}{\textsf{(LetterChoice)}\ \ 4 : aa^*b^* \sqsubseteq a(a+b)^*}}{\textsf{(LetterStar)}\ \ 3 : aa^*b^* \sqsubseteq (a+b)(a+b)^*}}{\textsf{(LeftStar)}\ \ 2 : aa^*b^* \sqsubseteq (a+b)^*} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\textit{Store}(6)}}{\textsf{(Letter)}\ \ 10 : b^* \sqsubseteq (a+b)^*}}{\textsf{(LetterChoice)}\ \ 9 : bb^* \sqsubseteq b(a+b)^*}}{\textsf{(LetterStar)}\ \ 8 : bb^* \sqsubseteq (a+b)(a+b)^*} \quad \dfrac{}{\textsf{(Axm)}}}{\textsf{(LeftStar)}\ \ 7 : bb^* \sqsubseteq (a+b)^*} \quad \dfrac{}{11 : \epsilon \sqsubseteq (a+b)^*}}{6 : b^* \sqsubseteq (a+b)^*}}{1 : a^*b^* \sqsubseteq (a+b)^*}$$

Figure 3: Example usage of the inference rules to decide $a^*b^* \sqsubseteq (a+b)^*$. Note that this and the following examples are written bottom-up, therefore the input is at the bottom.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\textit{Store}(3)}}{\textsf{(Letter)}\ \ 7 : b(ab)^*a \sqsubseteq (ba)^*} \quad \dfrac{}{\textsf{(Axm)}}}{\textsf{(Letter)}\ \ 9 : \epsilon \sqsubseteq (ba)^*}}{\textsf{(LeftStar)}\ \ 6 : ab(ab)^*a \sqsubseteq a(ba)^*} \quad 8 : a \sqsubseteq a(ba)^*}{\textsf{(Letter)}\ \ 5 : (ab)^*a \sqsubseteq a(ba)^*}}{\textsf{(LetterStar)}\ \ 4 : b(ab)^*a \sqsubseteq ba(ba)^*} \quad \dfrac{}{\textit{Store}(9)}}{\textsf{(Letter)}\ \ 3 : b(ab)^*a \sqsubseteq (ba)^*} \quad \dfrac{}{\textsf{(Letter)}\ \ 11 : \epsilon \sqsubseteq (ba)^*}}{\textsf{(LeftStar)}\ \ 2 : ab(ab)^*a \sqsubseteq a(ba)^*} \quad 10 : a \sqsubseteq a(ba)^*}{1 : (ab)^*a \sqsubseteq a(ba)^*}$$

Figure 4: Example usage of the inference rules to decide $(ab)^*a \sqsubseteq a(ba)*$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Fail because } \textsf{first}((ab)^*) \not\subseteq \textsf{first}(b^*)}{\textsf{(Letter)}\ \ 7 : (ab)^* \sqsubseteq b^*}}{\textsf{(LetterStar)}\ \ 6 : b(ab)^* \sqsubseteq bb^*}}{\textsf{(ElimCat)}\ \ 5 : b(ab)^* \sqsubseteq b^*}}{\textsf{(Letter)}\ \ 4 : b(ab)^* \sqsubseteq a^*b^*}}{\textsf{(LetterStar)}\ \ 3 : ab(ab)^* \sqsubseteq aa^*b^*} \quad \dfrac{}{\textsf{(Axm)}}}{\textsf{(LeftStar)}\ \ 2 : ab(ab)^* \sqsubseteq a^*b^*} \quad \dfrac{}{8 : \epsilon \sqsubseteq a^*b^*}}{1 : (ab)^* \sqsubseteq a^*b^*}$$

Figure 5: Example usage of the inference rules to decide that $(ab)^* \sqsubseteq a^*b^*$ is not sound

Table 2: The rules for the relation $\sqsubseteq$.

$$\text{(Axm)}\quad \frac{}{\epsilon \sqsubseteq r}\ [r \in \mathfrak{N}_\Sigma] \qquad \text{(Letter)}\quad \frac{r_1 \sqsubseteq r_2}{l \cdot r_1 \sqsubseteq l \cdot r_2} \qquad \text{(LetterStar)}\quad \frac{l \cdot r_1 \sqsubseteq r_2 r_2^* r_3}{l \cdot r_1 \sqsubseteq r_2^* r_3}\ [l \in \mathsf{first}(r_2)]$$

$$\text{(LetterChoice)}\quad \frac{l \cdot r_1 \sqsubseteq r_i r_4}{l \cdot r_1 \sqsubseteq (r_2 + r_3) r_4}\ \left[\begin{array}{c} i \in \{2,3\} \\ l \in \mathsf{first}(r_i) \end{array}\right] \qquad \text{(LeftChoice)}\quad \frac{r_1 r_3 \sqsubseteq r_4 \quad r_2 r_3 \sqsubseteq r_4}{(r_1 + r_2) r_3 \sqsubseteq r_4}$$

$$\text{(LeftStar)}\quad \frac{r_1 r_1^* r_2 \sqsubseteq r_3 r_4 \quad r_2 \sqsubseteq r_3 r_4}{r_1^* r_2 \sqsubseteq r_3 r_4}\ \left[\begin{array}{c} \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) \neq \varnothing \\ \exists l, r_5 : r_3 = l \vee r_3 = r_5^* \end{array}\right]$$

$$\text{(StarChoice1)}\quad \frac{r_1^* r_2 \sqsubseteq r_i r_5}{r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5}\ \left[\begin{array}{c} i \in \{3,4\} \\ \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_i) \neq \varnothing \\ \mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_i r_5) \\ r_2 \notin \mathfrak{N}_\Sigma \vee r_i \in \mathfrak{N}_\Sigma \end{array}\right]$$

$$\text{(StarChoice2)}\quad \frac{r_1 r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5 \quad r_2 \sqsubseteq (r_3 + r_4) r_5}{r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5}\ \left[\begin{array}{c} \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 + r_4) \neq \varnothing \\ \left(\begin{array}{l} (r_4 \notin \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 r_5) \neq \varnothing \\ \vee\ \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) \neq \varnothing \\ \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_4 \notin \mathfrak{N}_\Sigma) \end{array}\right) \\ \left(\begin{array}{l} (r_3 \notin \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4 r_5) \neq \varnothing \\ \vee\ \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4) \neq \varnothing \\ \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_3 \notin \mathfrak{N}_\Sigma) \end{array}\right) \end{array}\right]$$

$$\text{(ElimCat)}\quad \frac{r_1 \sqsubseteq r_3}{r_1 \sqsubseteq r_2 r_3}\ \left[\begin{array}{c} \exists l, r_4, r_5 : r_1 = l \cdot r_4 \ \vee\ r_1 = r_4^* r_5 \\ r_2 \in \mathfrak{N}_\Sigma \\ \mathsf{first}(r_1) \subseteq \mathsf{first}(r_3) \end{array}\right]$$

4. Properties of the Algorithm

To help understanding the algorithm and the rules, Table 3 shows which rules might apply for each combination of header-forms of the left-hand and right-hand expressions. The following lemma implies that if the second "if" inside the main loop of the algorithm fails, then there is always at least one rule matching the pair. Note also that the conditions in the lemma hold for all pairs which are in the inclusion relation.

Lemma 4.1. *For any regular expressions r_1 and r_2 in header normal form, where* $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_2)$, $r_1 \notin \mathfrak{N}_\Sigma \vee r_2 \in \mathfrak{N}_\Sigma$, *and $r_1 = \epsilon \vee r_2 \neq \epsilon$, there is at least one rule instance with conclusion $r_1 \sqsubseteq r_2$.*

Proof. By a case distinction on r_1 and r_2, using Tables 2 and 3, Definition 2.2, and Lemma 2.5. The only combinations that are never matched are when the right-hand expression is ϵ while the left-hand expression is not (5, 9, and 13 in Table 3), and the combinations where the left-hand expression is ϵ while the right-hand is of the form $l \cdot r$ (2 in Table 3). The former cannot occur under the assumptions of the lemma since subexpressions of the forms $\epsilon \cdot r'$, $\epsilon + \epsilon$ and ϵ^* are assumed removed, while the latter combinations follow from that $l \cdot r \notin \mathfrak{N}_\Sigma$.

The cases when $r_1 = \epsilon$ (1, 2, 3, and 4 in Table 3) the pair matches (Axm), as the only side condition, $r_2 \in \mathfrak{N}_\Sigma$, is true by assumption. When both expressions start with a letter (6 in Table 3), the pair matches (Letter), which has no

side-conditions.

In the cases where $r_1 = lr'$ and $r_2 = (r_3 + r_4)r_5$ (7 in Table 3) we have by assumption either that $l \in \mathsf{first}(r_3 + r_4)$, such that (LetterChoice) matches, or we have $r_3 + r_4 \in \mathfrak{N}_\Sigma$ and $l \in \mathsf{first}(r_5)$ such that (ElimCat) matches the pair.

In the cases where $r_1 = lr'$ and $r_2 = r_3^* r_4$ (8 in Table 3) we have by assumption either that $l \in \mathsf{first}(r_3)$, such that (LetterStar) matches, or we have $l \in \mathsf{first}(r_4)$ such that (ElimCat) matches the pair.

The cases where $r_1 = (r_3 + r_4)r_5$ (9, 10, 11, and 12 in Table 3) match (LeftChoice) which has no side-conditions.

The cases where $r_1 = r_3^* r_4$ and $r_2 = l \cdot r_5$ (14 in Table 3) are matched by (LeftStar). The first side-condition holds by the assumptions in the lemma, and the second by the form of r_2.

For the cases where $r_1 = r_3^* r_4$ and $r_2 = r_5^* r_6$ (16 in Table 3), note that from the assumptions in the lemma, $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_5^* r_6)$. There are two cases to treat. Firstly, we can have $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_6)$ such that the pair matches (ElimCat). Otherwise, we have $\mathsf{first}(r_1) \cap \mathsf{first}(r_5) \neq \varnothing$ which implies that the pair matches (LeftStar).

We now treat the hardest case (15 in Table 3). For expository reasons we stick to the notation in (StarChoice2) and take the left hand side ("r_1") to be $r_1^* r_2$ and the right hand side ("r_2") to be $(r_3 + r_4)r_5$. The pair can possibly match (ElimCat), (StarChoice1), or (StarChoice2). We will treat this case by assuming that the pair does not match (ElimCat) or (StarChoice1), and proceeding to show that it is then matched by (StarChoice2). We only need to show that all the side-conditions of (StarChoice2) hold. For the first side-condition, note that one assumption in the lemma is that $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}((r_3 + r_4)r_5)$. If $r_3 + r_4 \notin \mathfrak{N}_\Sigma$, we have $\mathsf{first}((r_3 + r_4)r_5) = \mathsf{first}(r_3 + r_4)$ and therefore get $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_3 + r_4)$, so the first side-condition holds. Otherwise, if $r_3 + r_4 \in \mathfrak{N}_\Sigma$, we get from the fact that (ElimCat) does not match the conclusion that $\mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_5)$, so we must also have the first side-condition. For the two remaining side-conditions of (StarChoice2), note first that since (StarChoice1) does not match, we get the following two facts:

$$\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) = \varnothing \vee \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5) \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_3 \notin \mathfrak{N}_\Sigma) \tag{1}$$

$$\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4) = \varnothing \vee \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5) \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_4 \notin \mathfrak{N}_\Sigma) \tag{2}$$

From the fact that (ElimCat) did not match, we get that $r_3 \in \mathfrak{N}_\Sigma \Rightarrow \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_5)$ and that $r_4 \in \mathfrak{N}_\Sigma \Rightarrow \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_5)$. Therefore, $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) = \varnothing \Rightarrow \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5)$ and $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4) = \varnothing \Rightarrow \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5)$. Hence (1) and (2) can be simplified to

$$\mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5) \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_3 \notin \mathfrak{N}_\Sigma) \tag{3}$$

$$\mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5) \vee (r_2 \in \mathfrak{N}_\Sigma \wedge r_4 \notin \mathfrak{N}_\Sigma) \tag{4}$$

Applying standard operations in propositional logic to (3) and (4) we get

$$\begin{pmatrix} (r_3 \notin \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5)) \\ \vee\ (r_3 \in \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5)) \\ \vee\ (r_2 \in \mathfrak{N}_\Sigma \wedge r_3 \notin \mathfrak{N}_\Sigma) \end{pmatrix} \tag{5}$$

$$\begin{pmatrix} (r_4 \notin \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5)) \\ \vee\ (r_4 \in \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5)) \\ \vee\ (r_2 \in \mathfrak{N}_\Sigma \wedge r_4 \notin \mathfrak{N}_\Sigma) \end{pmatrix} \tag{6}$$

We use again that $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}((r_3 + r_4)r_5) = \mathsf{first}(r_3 r_5) \cup \mathsf{first}(r_4 r_5)$ to get the following implications: $\mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5) \Rightarrow \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4 r_5) \neq \varnothing$, $\mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5) \Rightarrow \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 r_5) \neq \varnothing$, $(r_3 \in \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_3 r_5)) \Rightarrow \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4) \neq \varnothing$, and $(r_4 \in \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \nsubseteq \mathsf{first}(r_4 r_5)) \Rightarrow \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) \neq \varnothing$. Applying these implications to (5) and (6) gives exactly the two last side-conditions of (StarChoice2). □

4.1. 1-Unambiguity and the Rules

We must make sure that the rules given in Table 2 preserve 1-unambiguity for the *right-hand* expressions.

Lemma 4.2 (Preservation of 1-unambiguity)**.** *For any rule instance, if the right-hand expression in the conclusion is 1-unambiguous, then also the right-hand expressions in all the premises are 1-unambiguous.*

Table 3: The rules that might apply for any combination of header-forms of the left-hand and right-hand expressions

Left \ Right	ϵ	$l \cdot r$	$(r_1 + r_2) \cdot r_3$	$r_1^* \cdot r_2$
ϵ	1 : (Axm)	2 :$\not\subseteq$	3 : (Axm)	4 : (Axm)
$l \cdot r$	5 :$\not\subseteq$	6 : (Letter)	7 : (ElimCat) (LetterChoice)	8 : (ElimCat) (LetterStar)
$(r_1 + r_2) \cdot r_3$	9 :$\not\subseteq$	10 : (LeftChoice)	11 : (LeftChoice)	12 : (LeftChoice)
$r_1^* \cdot r_2$	13 :$\not\subseteq$	14 : (LeftStar)	15 : (ElimCat) (StarChoice1) (StarChoice2)	16 : (ElimCat) (LeftStar)

Proof. For most rules we either have that the right-hand expression is the same in the premise and the conclusion, or we can use the fact that all subexpressions of a 1-unambiguous regular expression are 1-unambiguous. The latter fact was shown by Brüggemann-Klein & Wood [6, Lemma 3.2]. The remaining cases can also be shown by using [6, Lemma 3.2]. For the convenience of the reader, we show it in an alternative way using Definition 2.14.

For the rule (LetterStar), the right-hand expression of the premise is of the form $r_1 r_1^* r_2$ and we know that $r_1^* r_2$ is 1-unambiguous. We will prove that $r_1 r_1^* r_2$ is 1-unambiguous given that $r_1^* r_2$ is 1-unambiguous. We must use the fact that all expressions are in star normal form (see Definition 2.13), thus $r_1 \notin \mathfrak{N}_\Sigma$, and $\mathsf{first}(\mu(r_1)) \cap \mathsf{followLast}(\mu(r_1)) = \varnothing$. Take $u, v, w \in \mathsf{sym}(\mu(r_1 r_1^* r_2))^*$ and $p, q \in \mathsf{sym}(\mu(r_1 r_1^* r_2))$ as in Definition 2.14, such that $upv, uqw \in \|\mu(r_1 r_1^* r_2)\|$ and $(r_1 r_1^* r_2)[p] = (r_1 r_1^* r_2)[q]$. To prove 1-unambiguity of $r_1 r_1^* r_2$ we must show that $p = q$. For each of the words upv and uqw there are two possibilities to consider:

$$\exists! u_1, u_2, p_1, v_1 : \; u = (1 \downarrow u_1) \cdot (2 \downarrow u_2) \wedge v = 2 \downarrow v_1 \wedge p = 2 \downarrow p_1 \wedge u_1 \in \|\mu(r_1)\| \wedge u_2 p_1 v_1 \in \|\mu(r_1^* r_2)\| \tag{7}$$

$$\exists! u_1, p_1, v_1, v_2 : \; u = 1 \downarrow u_1 \wedge p = 1 \downarrow p_1 \wedge v = (1 \downarrow v_1) \cdot (2 \downarrow v_2) \wedge u_1 p_1 v_1 \in \|\mu(r_1)\| \wedge v_2 \in \|\mu(r_1^* r_2)\| \tag{8}$$

$$\exists! u_1, u_2, q_1, w_1 : \; u = (1 \downarrow u_1) \cdot (2 \downarrow u_2) \wedge w{=}2 \downarrow w_1 \wedge q = 2 \downarrow q_1 \wedge u_1 \in \|\mu(r_1)\| \wedge u_2 q_1 w_1 \in \|\mu(r_1^* r_2)\| \tag{9}$$

$$\exists! u_1, q_1 w_1, w_2 : \; u = 1 \downarrow u_1 \wedge q = 1 \downarrow q_1 \wedge w = (1 \downarrow w_1) \cdot (2 \downarrow w_2) \wedge u_1 q_1 w_1 \in \|\mu(r_1)\| \wedge w_2 \in \|\mu(r_1^* r_2)\| \tag{10}$$

Exactly one of (7) or (8) must hold, and exactly one of (9) or (10) must hold. Firstly, if both (8) and (10) hold, then $p_1 = q_1$ follows from 1-unambiguity of r_1, and thus $p = 1 \downarrow p_1 = 1 \downarrow q_1 = q$. Secondly, if both (7) and (9) hold, the u_1 and u_2 chosen must be the same in both cases, and therefore 1-unambiguity of $r_1^* r_2$ can be used to get $p_1 = q_1$. Thus $p = 2 \downarrow p_1 = 2 \downarrow q_1 = q$. We now show that the two remaining combinations cannot hold. By symmetry, we only treat one case, and assume (by contradiction) that (8) and (9) hold. This implies that $u_2 = \epsilon$, thus $p_1 \in \mathsf{followLast}(\mu(r_1))$ and $q_1 \in \mathsf{first}(\mu(r_1^* \cdot r_2))$. Now we can use $(\langle 1, 1\rangle \downarrow (u_1 \cdot p_1 \cdot v_1)) \cdot v_2 \in \|\mu(r_1^* r_2)\|$ and $(\langle 1, 1\rangle \downarrow u_1) \cdot q_1 \cdot w_1 \in \|\mu(r_1^* r_2)\|$ together with the fact that $(r_1^* r_2)[q] = (r_1^* r_2)[\langle 1, 1\rangle \downarrow p_1]$ in Definition 2.14 to show that $\langle 1, 1\rangle \downarrow p_1 = q_1$. Combined with $q_1 \in \mathsf{first}(\mu(r_1^* \cdot r_2))$ we get that $p_1 \in \mathsf{first}(\mu(r_1))$. But then $p_1 \in \mathsf{first}(\mu(r_1)) \cap \mathsf{followLast}(\mu(r_1))$, which contradicts with the fact that $r_1^* r_2$ is in star normal form.

For (LetterChoice) and (StarChoice1), the right-hand expression in the conclusion is of the form $(r_1 + r_2) r_3$. We can, by symmetry, assume the right-hand expression in the premise is $r_1 r_3$. We assume the right-hand expressions in the conclusion is 1-unambiguous and show that $r_1 r_3$ also is 1-unambiguous. Note now that $\|\mu(r_1 \cdot r_3)\| = (1 \downarrow \|\mu(r_1)\|) \cdot (2 \downarrow \|\mu(r_3)\|)$, and $\|\mu((r_1 + r_2) r_3)\| = (1 \downarrow \|\mu(r_1 + r_2)\|) \cdot (2 \downarrow \|\mu(r_3)\|) = (\langle 1, 1\rangle \downarrow \|\mu(r_1)\|) \cdot (2 \downarrow \|\mu(r_3)\|) \cup (\langle 1, 2\rangle \downarrow \|\mu(r_1)\|) \cdot (2 \downarrow \|\mu(r_3)\|)$. For any $upv, uqw \in \|\mu(r_1 r_3)\|$ as in Definition 2.14 concerning $r_1 r_3$ we therefore have corresponding u', p', q', v', w' concerning $(r_1 + r_2) r_3$ which are obtained by prefixing the positions in u, p, q, v, w starting in 1 with one more 1. Furthermore, $p' = q' \Rightarrow p = q$. Since $(r_1 + r_2) r_3$ is 1-unambiguous we have that $(r_1 + r_2) r_3[p'] = (r_1 + r_2) r_3[q'] \Rightarrow p' = q'$ and therefore also that $r_1 r_3[p] = r_1 r_3[q] \Rightarrow p = q$, such that $r_1 r_3$ is also 1-unambiguous.

□

We must now substantiate the claim that if the side-conditions of more than one applicable rule hold, the right-hand expression is 1-ambiguous.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{(Axm)}}}{\text{(Letter)}\ \ 4 : \epsilon \sqsubseteq \epsilon}}{\text{(Letter)}\ \ 3 : b \sqsubseteq b}}{\text{(LetterChoice)}\ \ 2 : ab \sqsubseteq ab}}{1 : ab \sqsubseteq (a + (b + c)^* c(b + c) \cdots (b + c))b}$$

Figure 6: Example usage of the inference rules

Lemma 4.3. *For any two regular expressions r_1 and r_2, where r_2 is 1-unambiguous, there is at most one rule instance with $r_1 \sqsubseteq r_2$ in the conclusion.*

Proof. This is proved by comparing each pair of rule instances of rules occurring in Table 3 and using Definition 2.14. For each case, we show that the existence of several rule instances with the same conclusion implies that the right-hand expression is 1-ambiguous.

- We first consider the case that one rule has several instances matching the same conclusion. The only rules that can have more than one instance with the same conclusion are (StarChoice1) and (LetterChoice). For (LetterChoice), the conclusion is of the form $l \cdot r_1 \sqsubseteq (r_2 + r_3) \cdot r_4$, and the existence of two instances implies that $l \in \mathsf{first}(r_2) \cap \mathsf{first}(r_3)$. This can only be the case if the right-hand expression is 1-ambiguous. For (StarChoice1), the conclusion is of the form $r_1^* r_2 \sqsubseteq (r_3 + r_4)r_5$, and the existence of two instances of this rule would imply that $\mathsf{first}(r_1^* r_2)$ and $\mathsf{first}(r_4)$ have a non-empty intersection, which furthermore is included in $\mathsf{first}(r_3 r_5)$. The expression $(r_3 + r_4)r_5$ is therefore 1-ambiguous.

- If instances of both (ElimCat) and either (LetterStar) or (LetterChoice) match the pair of expressions (see 7 and 8 in Table 3), then the right-hand expression is of the form $r_2 r_3$. From (LetterStar) and (LetterChoice) the left-hand expression is of the form $l r_1$ and $l \in \mathsf{first}(r_2)$. From (ElimCat) we get that $r_2 \in \mathfrak{N}_\Sigma$ and $l \in \mathsf{first}(r_3)$. But this means that $r_2 r_3$ is 1-ambiguous.

- If instances of both (ElimCat) and either (StarChoice1) or (StarChoice2) had the same conclusion (see 15 in Table 3), then the the conclusion is of the form $r_1^* r_2 \sqsubseteq (r_3 + r_4)r_5$. From (ElimCat), we get that $r_3 + r_4 \in \mathfrak{N}_\Sigma$ and $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_5)$. From the second side-condition of (StarChoice1) or the first side-condition of (StarChoice2) we get that $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 + r_4) \neq \varnothing$. This means that $\mathsf{first}(r_3 + r_4) \cap \mathsf{first}(r_5) \neq \varnothing$. We combine the latter with $r_3 + r_4 \in \mathfrak{N}_\Sigma$ to get that the right-hand expression $(r_3 + r_4)r_5$ is 1-ambiguous.

- For the cases where both an instance of (StarChoice1) and one of (StarChoice2) match the conclusion, we can by symmetry assume the instance of (StarChoice1) has $i = 3$. The last side-condition of (StarChoice1) is then $r_2 \notin \mathfrak{N}_\Sigma \vee r_3 \in \mathfrak{N}_\Sigma$. This is exactly the negation of the third disjunct of the third side condition of (StarChoice2). We must therefore have that the remaining disjunction holds, that is,

$$(r_3 \notin \mathfrak{N}_\Sigma \wedge \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4 r_5) \neq \varnothing) \vee \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_4) \neq \varnothing \tag{11}$$

  Assume first that the left disjunct of (11) holds. Then since $r_3 \notin \mathfrak{N}_\Sigma$, we get $\mathsf{first}(r_3 r_5) = \mathsf{first}(r_3)$. Combined with the third side-condition of (StarChoice1) this implies that $\mathsf{first}(r_4 r_5) \cap \mathsf{first}(r_3) \neq \varnothing$, which means that $(r_3 + r_4)r_5$ is 1-ambiguous. Otherwise, if the right disjunct of (11) holds, we can use the third side-condition of (StarChoice1) to get that $\mathsf{first}(r_4) \cap \mathsf{first}(r_3 r_5) \neq \varnothing$, which also means that $(r_3 + r_4)r_5$ is 1-ambiguous.

- If instances of both (ElimCat) and (LeftStar) match the pair of expressions (see 16 in Table 3), then the conclusion is of the form $r_1^* r_2 \sqsubseteq r_3 r_4$, where $r_3 \in \mathfrak{N}_\Sigma$ and both $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_4)$ and $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) \neq \varnothing$. This can only hold if $r_3 r_4$ is 1-ambiguous.

□

4.2. Invertibility of the Rules

We shall now prove that the rules given in Table 2 are *invertible*. By this we mean that, for each rule instance, assuming that no other rule instance matches the conclusion, then the conclusion is sound if and only if all premises are sound.

Proof. By a case distinction on the rules. For all rules, the fact that the premise(s) implies the conclusion follows almost directly from Definition 2.2. We only treat the converse:

- For (Axm), we only note that the side-condition is that the right-hand expression is nullable, and then $\{\epsilon\}$ is of course a subset of the language.
- For (Letter) we are just removing a single letter prefix from both languages, and this preserves the inclusion relation.
- For (LetterStar), the conclusion is of the form $lr_1 \sqsubseteq r_2^* r_3$. Note that $\|r_2^* r_3\| = \|r_2 r_2^* r_3\| \cup \|r_3\|$. Since (ElimCat) does not match the conclusion, and $r_2^* \in \mathfrak{N}_\Sigma$, we must have that $\mathsf{first}(lr_1) \not\subseteq \mathsf{first}(r_3)$, that is, $l \notin \mathsf{first}(r_3)$. Therefore $\|lr_1\| \cap \|r_3\| = \varnothing$, and thus $\|lr_1\| \subseteq \|r_2 r_2^* r_3\|$ and the premise is sound
- For (LetterChoice), the conclusion is of the form $lr_1 \sqsubseteq (r_2 + r_3)r_4$. Again we depend on the fact that no other instance of (LetterChoice) nor (ElimCat) match the conclusion. We can assume by symmetry that $i = 2$ and the premise is of the form $lr_1 \sqsubseteq r_2 r_4$. Since $i = 3$ does not match we get that $l \notin \mathsf{first}(r_3)$. Note that $\|(r_2 + r_3)r_4\| = \|r_2 r_4\| \cup \|r_3 r_4\|$. Since (ElimCat) does not match the conclusion we get that $(r_2 + r_3) \in \mathfrak{N}_\Sigma \Rightarrow l \notin \mathsf{first}(r_4)$. This implies that $\|lr_1\| \cap \|r_3 r_4\| = \varnothing$, so $\|lr_1\| \subseteq \|r_2 r_4\|$ and we have the premise.
- For (LeftChoice), the implication follows from Definition 2.2.
- (LeftStar) and (StarChoice2) hold by Definition 2.2, as $\|r_1^* r_2\| = \|r_1 r_1^* r_2\| \cup \|r_2\|$.
- For (StarChoice1), the conclusion is of the form $r_1^* r_2 \sqsubseteq (r_3 + r_4)r_5$. Note again that $\|(r_3 + r_4)r_5\| = \|r_3 r_5\| \cup \|r_4 r_5\|$. We can, by symmetry, assume $i = 3$. The second side-condition is then that $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3) \neq \varnothing$. Note that this implies the first side-condition and the middle disjunct of the second side-condition in (StarChoice2). Since (StarChoice2) does not match, we must have the negation of the third side-condition of (StarChoice2). Hence
$$\mathsf{first}(r_4) \cap \mathsf{first}(r_1^* r_2) = \varnothing \wedge (r_3 \in \mathfrak{N}_\Sigma \vee \mathsf{first}(r_4 r_5) \cap \mathsf{first}(r_1^* r_2) = \varnothing) \tag{12}$$
Now, if $r_3 \in \mathfrak{N}_\Sigma$, we get that $\|r_5\| \subseteq \|r_3 r_5\|$, which implies that $\|(r_3 + r_4)r_5\| = \|r_3 r_5\| \cup (\|r_4 r_5\| - \|r_5\|)$. From (12) we have that $\mathsf{first}(r_4) \cap \mathsf{first}(r_1^* r_2) = \varnothing$, which implies that $(\|r_4 r_5\| - \|r_5\|) \cap \|r_1^* r_2\| = \varnothing$. Therefore the premise $r_1^* r_2 \sqsubseteq r_3 r_5$ is sound. On the other hand, if $r_3 \notin \mathfrak{N}_\Sigma$, we get from (12) that $\mathsf{first}(r_4 r_5) \cap \mathsf{first}(r_1^* r_2) = \varnothing$. This implies that $\|r_1^* r_2\| \cap \|r_4 r_5\| = \varnothing$, which implies that the premise $r_1^* r_2 \sqsubseteq r_3 r_5$ is sound.
- For (ElimCat), we have $\|r_2 r_3\| = (\|r_2 r_3\| - \|r_3\|) \cup \|r_3\|$. Therefore it is sufficient to show that $\mathsf{first}(r_1) \cap \mathsf{first}(r_2) = \varnothing$. Note that the left-hand expression is constrained by the first side-condition to be of the form $l \cdot r_4$ or $r_4^* r_5$. The right-hand expression must from Definition 2.3, and the definition of header-form be of the form $r_6^* r_3$ or $(r_6 + r_7)r_3$. We do a case distinction on these forms. If r_1 is of the form $l \cdot r_4$, then since neither (LetterStar) or (LetterChoice) matches the conclusion, we get that $l \notin \mathsf{first}(r_2)$, and the premise $r_1 \sqsubseteq r_3$ must be sound. If the conclusion is of the form $r_4^* r_5 \sqsubseteq r_6^* r_7$, then we must have that the first side-condition of (LeftStar) fails. Thus $\mathsf{first}(r_4^* r_5) \cap \mathsf{first}(r_6^*) = \varnothing$. Lastly, if the conclusion is of the form $r_4^* r_5 \sqsubseteq (r_6 + r_7)r_3$, note that from the second side-condition of (ElimCat) we have $r_6 + r_7 \in \mathfrak{N}_\Sigma$, so we can by symmetry assume $r_6 \in \mathfrak{N}_\Sigma$. We will use that (StarChoice1) with $i = 3$ does not match, and that the third and fourth side-conditions of this instance of (StarChoice1) hold by the assumption that the side-conditions of (ElimCat) hold. This implies that the second side-condition of (StarChoice1) with $i = 3$ does not hold, so we get $\mathsf{first}(r_4^* r_5) \cap \mathsf{first}(r_6) = \varnothing$. If $r_7 \in \mathfrak{N}_\Sigma$, we can use a similar argument to also get $\mathsf{first}(r_4^* r_5) \cap \mathsf{first}(r_7) = \varnothing$. Otherwise, if $r_7 \notin \mathfrak{N}_\Sigma$, we use the fact that (StarChoice2) does not match. The first disjunct of the second side-condition of (StarChoice2) holds, since we have assumed $r_7 \notin \mathfrak{N}_\Sigma$, and we argued above that $\mathsf{first}(r_4^* r_5) \subseteq \mathsf{first}(r_6 r_3)$. Therefore either the first or the third side-condition of (StarChoice2) must fail. Either case implies that $\mathsf{first}(r_4^* r_5) \cap \mathsf{first}(r_7) = \varnothing$, so we are done.

□

Invertibility implies that, at any point during an execution of the algorithm, the pair originally given as input is in the inclusion relation if and only if all the pairs in both the store S and the stack T are in the inclusion relation. These properties are used in the proofs of soundness and completeness below.

4.3. Termination and Polynomial Run-time

To prove that the algorithm always terminates in polynomial time, we will prove that the number of iterations of the main loop where at least one new pair is pushed onto the stack T, has an upper bound in the product of the number of positions in the two regular expressions given as input. This implies that the whole algorithm runs in polynomial time, by the following three observations.

- The number of positions in a regular expression is linear in the length of the regular expression.
- The number of iterations where no new pair is pushed to the stack T, cannot be more than one more than half the total of all iterations. Note that the iterations where no pairs are pushed are those where the first "if"-test "$(r_1, r_2) \in \mathsf{S}$" succeeds, those where the second "if"-test fails, and those where the pair matches (Axm). That these are not more than one more than the half follows from that the other rules never push more than two pairs, and standard arguments on binary trees.
- The time used in each iteration of the loop is polynomial in the length of the regular expressions given as input.

Assume that the algorithm is given r_l and r_r as input. We will prove that there is an injective mapping from each r' occurring on the left-hand or right-hand of a pair in the stack T during the run of the algorithm, to a p in $\mathsf{pos}(r_l)$ or $\mathsf{pos}(r_r)$, respectively. If r is the corresponding input expression, then $r[p]$ is the first factor of r'.

For the purposes of this section, let $\lightning$ be a special *undefined* position, and let $\mathsf{pos}(r)^{\lightning} = \mathsf{pos}(r) \cup \{\lightning\}$. We proceed to define a mapping next_r, which will be used to describe the expressions occurring in a run of the algorithm in terms of subexpressions of the corresponding expression given as input.

Definition 4.4 (next_r)**.** *For a regular expression r, let the mapping* $\mathsf{next}_r : \mathsf{pos}(r) \to \mathsf{pos}(r)^{\lightning}$ *be defined in the following top-down inductive manner:*

- *Put* $\mathsf{next}_r(\langle\rangle) = \lightning$.
- *If* $r[p] = r_1 \cdot r_2$*, put* $\mathsf{next}_r(p \downarrow 1) = p \downarrow 2$ *and put* $\mathsf{next}_r(p \downarrow 2) = \mathsf{next}_r(p)$.
- *If* $r[p] = r_1 + r_2$*, put* $\mathsf{next}_r(p \downarrow 1) = \mathsf{next}_r(p \downarrow 2) = \mathsf{next}_r(p)$.
- *If* $r[p] = r_1^*$*, put* $\mathsf{next}_r(p \downarrow 1) = p$.

We extend next_r *to* next_r^* *which maps a position in r to a list of positions in r:*

$$\mathsf{next}_r^*(p) = \begin{cases} \epsilon & \text{if } \mathsf{next}_r(p) = \lightning \\ \mathsf{next}_r(p) \cdot \mathsf{next}_r^*(\mathsf{next}_r(p)) & \text{otherwise} \end{cases}$$

Example 4.5. *Let* $\Sigma = \{a, b, c, d\}$ *and* $r = ((a \cdot b) \cdot c^*) \cdot d$*. Then*

$$\mathsf{pos}(r) = \{\langle\rangle, \langle 1\rangle, \langle 1, 1\rangle, \langle 1, 1, 1\rangle, \langle 1, 1, 2\rangle, \langle 1, 2\rangle, \langle 1, 2, 1\rangle, \langle 2\rangle\}$$

and next_r *and* next_r^* *have the following values:*

$\mathsf{pos}(r)$	next_r	next_r^*
$\langle\rangle$	$\lightning$	ϵ
$\langle 1\rangle$	$\langle 2\rangle$	$\langle 2\rangle$
$\langle 1, 1\rangle$	$\langle 1, 2\rangle$	$\langle 1, 2\rangle \cdot \langle 2\rangle$
$\langle 1, 1, 1\rangle$	$\langle 1, 1, 2\rangle$	$\langle 1, 1, 2\rangle \cdot \langle 1, 2\rangle \cdot \langle 2\rangle$
$\langle 1, 1, 2\rangle$	$\langle 1, 2\rangle$	$\langle 1, 2\rangle \cdot \langle 2\rangle$
$\langle 1, 2\rangle$	$\langle 2\rangle$	$\langle 2\rangle$
$\langle 1, 2, 1\rangle$	$\langle 1, 2\rangle$	$\langle 1, 2\rangle \cdot \langle 2\rangle$
$\langle 2\rangle$	$\lightning$	ϵ

We now need an auxiliary lemma concerning header.

Lemma 4.6. *For any regular expressions r, $r_1, \ldots, r_n$, $r'_1, \ldots, r'_m$, and r', if* $\mathsf{header}(r, r_1 \cdots r_n \cdot \epsilon) = r'_1 \cdots r'_m \cdot \epsilon$, *then* $\mathsf{header}(r, r_1 \cdots r_n \cdot r') = r'_1 \cdots r'_m \cdot r'$.

Proof. By induction on r. For the base case $r = \epsilon$ we have by definition that $\mathsf{header}(r, r_1 \cdots r_n \cdot \epsilon) = r_1 \cdots r_n \cdot \epsilon$, and $\mathsf{header}(r, r_1 \cdots r_n \cdot r') = r_1 \cdots r_n \cdot r'$, which means the lemma holds. For the cases where r is a letter, choice, or a starred expression, we get by definition that $\mathsf{header}(r, r_1 \cdots r_n \cdot \epsilon) = r \cdot r_1 \cdots r_n$, and $\mathsf{header}(r, r_1 \cdots r_n \cdot r') = r \cdot r_1 \cdots r_n \cdot r'$, which also means the lemma holds. For the induction cases when $r = r''_1 \cdot \epsilon$ we get $\mathsf{header}(r, r_1 \cdots r_n \cdot \epsilon) = \mathsf{header}(r''_1, r_1 \cdots r_n \cdot \epsilon)$ and $\mathsf{header}(r, r_1 \cdots r_n \cdot r') = \mathsf{header}(r''_1, r_1 \cdots r_n \cdot r')$. We now get the result by applying the induction hypothesis for r''_1 to $\mathsf{header}(r''_1, r_1 \cdots r_n \cdot \epsilon) = r'_1 \cdots r'_m \cdot \epsilon$. Lastly, we treat the induction cases where there are $p \geq 2$ and $r''_1, \ldots, r''_p \in R_\Sigma - \{\epsilon\}$ such that either $r = r''_1 \cdots r''_p$ and r''_p is not a concatenation, or $r = r''_1 \cdots r''_p \cdot \epsilon$. We then get

$$\mathsf{header}(r, r_1 \cdots r_n \cdot \epsilon) = \mathsf{header}(r''_1, r''_2 \cdots r''_p \cdot r_1 \cdots r_n \cdot \epsilon)$$
$$\mathsf{header}(r, r_1 \cdots r_n \cdot r') = \mathsf{header}(r''_1, r''_2 \cdots r''_p \cdot r_1 \cdots r_n \cdot r')$$

We get the result by applying the induction hypothesis for r''_1 to $\mathsf{header}(r''_1, r''_2 \cdots r''_p \cdot r_1 \cdots r_n \cdot \epsilon) = r'_1 \cdots r'_m \cdot \epsilon$. □

Corollary 4.7. *For $m > 0$, and regular expressions r, r', r'', $r'_1, \ldots, r'_m$, if* $\mathsf{hdf}(r) = r'' \cdot \mathsf{hdf}(r'_1 \cdots r'_m \cdot \epsilon)$, *then* $\mathsf{hdf}(r \cdot r') = r'' \cdot \mathsf{hdf}(r'_1 \cdots r'_m \cdot r')$.

Proof. If $r = \epsilon$, the corollary holds vacuously. Otherwise, we can assume there are $r''_1, \ldots, r''_n \in R_\Sigma - \{\epsilon\}$ and $n \geq 0$ such that either $r' = r''_1 \cdots r''_n$ where $n \geq 1$ and r''_n is not a concatenation, or that $r' = r''_1 \cdots r''_n \cdot \epsilon$. Put $r''' = r''_1 \cdots r''_n \cdot \epsilon$. Then $\mathsf{hdf}(r \cdot r') = \mathsf{header}(r, r''')$, $\mathsf{hdf}(r'_1 \cdots r'_m \cdot r') = \mathsf{header}(r'_1, r'_2 \cdots r'_m \cdot r''')$, and $\mathsf{header}(r, \epsilon) = r'' \cdot \mathsf{header}(r'_1, r'_2 \cdots r'_m \cdot \epsilon)$. From Lemmas 2.9 and 4.6 we therefore get $\mathsf{header}(r, r''') = r'' \cdot \mathsf{header}(r'_1, r'_2 \cdots r'_m \cdot r''')$. Hence, $\mathsf{hdf}(r \cdot r') = r'' \cdot \mathsf{hdf}(r'_1 \cdots r'_m \cdot r')$. □

We now need an auxiliary lemma concerning the mapping next^*.

Lemma 4.8. *For any regular expressions r, r_1, r_2, and any position $p \in \mathsf{pos}(r)$ such that* $\mathsf{hdf}(r[p]) = r_1 \cdot r_2$, *there is an $n \geq 0$ and positions q, $p_1, \ldots, p_n \in \mathsf{pos}(r)$ such that $p \leq q$,* $\mathsf{next}^*_r(q) = p_1 \cdot \cdots \cdot p_n \cdot \mathsf{next}^*_r(p)$, *and* $r_1 \cdot r_2 = r[q] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot \epsilon)$.

Proof. By induction on the expression $r[p]$. The base case when $r[p] = \epsilon$ holds vacuously. The base cases when $r[p] \in \Sigma$, and the induction cases where $r[p]$ is of the forms $r_1 + r_2$ or r_1^* hold immediately (without using the induction hypothesis), as $\mathsf{hdf}(r[p]) = r[p] \cdot \epsilon = r[p] \cdot \mathsf{hdf}(\epsilon)$ and we can use $q = p$ and $n = 0$. The remaining induction cases are when $r[p]$ is of the form $r[p \downarrow 1] \cdot r[p \downarrow 2]$. By the note after Definition 2.2 $r[p \downarrow 1] \neq \epsilon$ and we have the induction hypothesis for $r[p \downarrow 1]$. Combining this with Definition 4.4, we get that there are $q, p_1, \ldots, p_n$, such that $p \downarrow 1 \leq q$, $\mathsf{next}^*_r(q) = p_1 \cdots p_n \cdot \mathsf{next}^*_r(p \downarrow 1) = p_1 \cdots p_n \cdot p \downarrow 2 \cdot \mathsf{next}^*_r(p)$, and $\mathsf{hdf}(r[p \downarrow 1]) = r[q] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot \epsilon)$. The latter fact applied to Corollary 4.7 implies that $\mathsf{hdf}(r[p]) = r[q] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot r[p \downarrow 2])$, so we have proved the lemma. □

We can now formulate the main lemma of this section, defining the mapping iterPos.

Lemma 4.9. *For each regular expression r given as input to any execution of the algorithm there exists a mapping* $\mathsf{iterPos}_r$ *with the following properties.*

- *The domain of* $\mathsf{iterPos}_r$ *is the set of non-ϵ expressions occurring on the same side as r in any pair on the stack during the execution of the algorithm.*
- *The codomain of* $\mathsf{iterPos}_r$ *is* $\mathsf{pos}(r)$.
- *If* $p = \mathsf{iterPos}_r(r')$, *then* $r' = r[p] \cdot \mathsf{hdf}(r[\mathsf{next}^*_r(p)])$.

Proof. By induction on the number of iterations of the main loop in an execution of the algorithm. We can assume that the lemma holds for the expressions in the pair that is popped from the stack, and show that it holds for the expressions being pushed onto the stack. Remember that hdf is applied to the expressions before they are pushed onto the stack.

The base case is the expressions r_l and r_r given as input. By symmetry we treat only r_l. We can apply Lemma 4.8 to r_l and $\langle\rangle$ to get $q, p_1, \ldots, p_n$ such that $\mathsf{hdf}(r_l) = r_l[q] \cdot \mathsf{hdf}(r_l[p_1] \cdots r_l[p_n] \cdot \epsilon)$, and $\mathsf{next}^*_{r_l}(q) = p_1 \cdots p_n \cdot \mathsf{next}^*_{r_l}(\langle\rangle)$. By Definition 4.4 $\mathsf{next}^*_{r_l}(\langle\rangle) = \epsilon$, so we get $\mathsf{hdf}(r_l) = r_l[q] \cdot \mathsf{hdf}(r_l[\mathsf{next}^*_{r_l}(q)])$, and we can let $\mathsf{iterPos}_{r_l}(r_l) = q$.

The induction case for (Axm), the induction cases where the first if-test "$(r_1, r_2) \notin \mathsf{S}$" fails, and the cases where the second if-test holds (such that "No" is returned) all hold directly by using the induction hypothesis, since the stack is not changed.

In the remaining induction cases, the expressions put on the stack follow five patterns: 1: that $\mathsf{hdf}(r)$ is pushed after popping r, 2: that $\mathsf{hdf}(r)$ is pushed after popping $l \cdot r$, 3: that $\mathsf{hdf}(r_1 r_3)$ is pushed after popping $(r_1 + r_2)r_3$, 4: that $\mathsf{hdf}(r_1 r_1^* r_2)$ is pushed after popping $r_1^* r_2$, and lastly, 5: that r_2 is pushed after popping $r_1 r_2$ where $r_1 \in \mathfrak{N}_\Sigma$. We treat these cases separately.

1. If we push $\mathsf{hdf}(r)$ on the stack after popping r, we get from Lemma 2.9 that $\mathsf{hdf}(r) = r$, so we get the result from the induction hypothesis.

2. The first interesting case is where $l \cdot r_1$ is a member of the pair popped from the stack T, and $\mathsf{hdf}(r_1)$ is a member of the pair pushed. Assume r is the corresponding input expression. By the induction hypothesis we know that there is a p such that $\mathsf{iterPos}_r(lr_1) = p$, $r[p] = l$, and $\mathsf{hdf}(r[\mathsf{next}^*_r(p)]) = r_1$. If $r_1 = \epsilon$, the lemma holds vacuously. Otherwise, we must now calculate the value of $\mathsf{iterPos}_r(r_1)$, that is, a $p' \in \mathsf{pos}(r)$, and show that it has the required properties. We have $r[\mathsf{next}^*_r(p)] \neq \epsilon$, so we get $\mathsf{next}^*_r(p) \neq \epsilon$, thus $\mathsf{next}_r(p) \neq \{\!\}$, and $r_1 = \mathsf{hdf}(r[\mathsf{next}_r(p)] \cdot r[\mathsf{next}^*_r(\mathsf{next}_r(p))])$. We can now apply Lemma 4.8 to r and $\mathsf{next}_r(p)$, and get $p', p_1, \ldots, p_n$ such that $\mathsf{hdf}(r[\mathsf{next}_r(p)]) = r[p'] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot \epsilon)$ and $\mathsf{next}^*_r(p') = p_1 \cdots p_n \cdot \mathsf{next}^*_{r_l}(\mathsf{next}_r(p))$. By applying this to Corollary 4.7 we get that

$$\begin{aligned}&\mathsf{hdf}(r[\mathsf{next}_r(p)] \cdot r[\mathsf{next}^*_r(\mathsf{next}_r(p))])\\ &= r[p'] \cdot \mathsf{hdf}(r[\mathsf{next}^*_r(p')])\end{aligned}$$

   Applying Lemma 2.9 we therefore get

$$\mathsf{hdf}(r_1) = r_1 = r[p'] \cdot \mathsf{hdf}(r[\mathsf{next}_r(p')]),$$

   so the lemma holds.

3. We next treat the case when we push a pair containing an expression of the form $\mathsf{hdf}(r_1 r_3)$ on the stack after popping a pair containing $(r_1 + r_2)r_3$. Assume r is the corresponding input expression. By the induction hypothesis there is a p such that $\mathsf{iterPos}_r((r_1 + r_2)r_3) = p$, $r[p] = (r_1 + r_2)$, and $\mathsf{hdf}(r[\mathsf{next}^*_r(p)]) = r_3$. If $\mathsf{hdf}(r_1 r_3) = \epsilon$ the lemma holds vacuously. Otherwise, since $r_1 = r[p \downarrow 1]$ we get from Lemma 4.8 for r and $p \downarrow 1$ that there are $p', p_1, \ldots, p_n$ such that:

$$\mathsf{hdf}(r_1) = r[p'] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot \epsilon) \tag{13}$$

$$\mathsf{next}^*_r(p') = p_1 \cdots p_n \cdot \mathsf{next}^*_r(p \downarrow 1) \tag{14}$$

   Applying Corollary 4.7 to (13) we get

$$\mathsf{hdf}(r_1 \cdot r_3) = r[p'] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot r_3)$$

   Since $r_3 = r[\mathsf{next}^*_r(p)]$ we get $\mathsf{hdf}(r_1 \cdot r_3) = r[p'] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot r[\mathsf{next}^*_r(p)])$. Furthermore, from Definition 4.4 $\mathsf{next}_r(p \downarrow 1) = \mathsf{next}_r(p)$, and therefore $\mathsf{next}^*_r(p \downarrow 1) = \mathsf{next}^*_r(p)$. Combining the latter with (14) we get $\mathsf{next}^*_r(p') = p_1 \cdots p_n \cdot \mathsf{next}^*_{r_l}(p)$. Finally, we therefore get $\mathsf{hdf}(r_1 \cdot r_3) = r[p'] \cdot \mathsf{hdf}(r[\mathsf{next}^*_r(p')])$, and we can put $\mathsf{iterPos}_r(r_3) = p'$.

4. We treat the case where $r_1^* r_2$ is a member of the pair popped from the stack, and $\mathsf{hdf}(r_1 r_1^* r_2)$ is a member of the pair pushed. Assume r is the corresponding input expression. By the induction hypothesis we have a p such that $\mathsf{iterPos}_r(r_1^* r_2) = p$, where $r[p] = r_1^*$ and $\mathsf{hdf}(r[\mathsf{next}_r^*(p)]) = r_2$. Since $r_1 = r[p \downarrow 1]$, we can apply Lemma 4.8 to r and $p \downarrow 1$, to get $p', p_1, \ldots, p_n$, such that $\mathsf{hdf}(r_1) = r[p'] \cdot \mathsf{hdf}(r[p_1 \cdots p_n])$ and $\mathsf{next}_r^*(p') = p_1 \cdots p_n \cdot \mathsf{next}_r^*(p \downarrow 1)$. By Definition 4.4, we get

$$\mathsf{next}_r^*(p') = p_1 \cdots p_n \cdot p \cdot \mathsf{next}_r^*(p)$$

Thus, applying Corollary 4.7 we get

$$\mathsf{hdf}(r_1 r_1^* r_2) = r[p'] \cdot \mathsf{hdf}(r[\mathsf{next}_r^*(p')])$$

So we can set $\mathsf{iterPos}_r(r_1 r_1^* r_2) = p'$.

5. For the case where $r_1 r_2$ is popped from the stack, $r_1 \in \mathfrak{N}_\Sigma$, and $\mathsf{hdf}(r_2)$ is pushed, assume again that r is the corresponding input expression. From the induction hypothesis there is a p such that $\mathsf{iterPos}_r(r_1 r_2) = p$, $r_1 = r[p]$, and $r_2 = \mathsf{hdf}(r[\mathsf{next}_r^*(p)])$. If $\mathsf{next}_r(p) = \lightning$, then $r_2 = \epsilon$ and the lemma holds vacuously for $\mathsf{hdf}(r_2) = \epsilon$. Otherwise, $r_2 = \mathsf{hdf}(r[\mathsf{next}_r(p)] \cdot r[\mathsf{next}_r^*(\mathsf{next}_r(p))])$. Applying Lemma 4.8 to r and $\mathsf{next}_r(p)$ gives $q, p_1, \ldots, p_n$ such that $\mathsf{hdf}(r[\mathsf{next}_r(p)]) = r[q] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot \epsilon)$ and $\mathsf{next}_r^*(q) = p_1 \cdots p_n \cdot \mathsf{next}_r^*(\mathsf{next}_r(p))$. Applying Corollary 4.7 to this we get $\mathsf{hdf}(r[\mathsf{next}_r(p)] \cdot r[\mathsf{next}_r^*(\mathsf{next}_r(p))]) = r[q] \cdot \mathsf{hdf}(r[p_1] \cdots r[p_n] \cdot r[\mathsf{next}_r^*(\mathsf{next}_r(p))]) = r[q] \cdot \mathsf{hdf}(r[\mathsf{next}_r^*(q)])$. Thus $\mathsf{hdf}(r_2) = r[q] \cdot \mathsf{hdf}(r[\mathsf{next}_r^*(q)])$, and we can set $\mathsf{iterPos}_r(r_2) = q$.

□

Note now that the mappings $\mathsf{iterPos}_{r_l}$ and $\mathsf{iterPos}_{r_r}$ are injective. We show this by letting $r \in \{r_l, r_r\}$, and assuming that for two regular expressions r_1 and r_2 occurring on the same side as r in two pairs in the stack, we have $\mathsf{iterPos}_r(r_1) = \mathsf{iterPos}_r(r_2) = p$ for some p. But from Lemma 4.9 we then have $r_1 = r_2 = r[p] \cdot \mathsf{hdf}(r[\mathsf{next}_r^*(p)])$. So the mapping $\mathsf{iterPos}_r$ is injective.

We are now done with showing termination and polynomial run-time, since Lemma 4.9 implies that the product of the number of positions in the two regular expressions given as input is an upper bound to the number of pairs of members from $R_\Sigma - \{\epsilon\}$ occurring in the stack. The latter number is exactly the number of iterations of the main loop where new pairs are pushed to the stack, since an ϵ on the left-hand side can only be matched by (Axm) and if only the right-hand side is ϵ, this leads to a "No" answer. As argued in the beginning of this section, this means the run-time of the whole algorithm is polynomial.

5. Soundness and Completeness

We need some auxiliary definitions and lemmas before we can prove soundness.

Definition 5.1 (Execution graph)**.** *An execution graph is a directed graph representing a successful run of the algorithm. The nodes correspond to the iterations of the main loop in the algorithm where the test "$(r_1, r_2) \in \mathsf{S}$" fails. Each node is labeled by the name and conclusion of the rule instance matching the pair popped from the stack in the corresponding iteration. There is an edge from each node to the node(s) labeled with the premise(s) of the rule instance applied in the corresponding iteration.*

Every usage of the store corresponds to a loop in the graph. Note that the only nodes without outgoing edges in an execution graph, are those labeled (Axm).

Example 5.2. *Figure 7 shows the execution graph corresponding to a run of the algorithm with input $a^* b^*, (a + b)^*$.*

Let the *size* of a regular expression be the sum of the number of letters and operators $*$ and $+$ occurring in the expression. Note that the concatenation operator and ϵ are not counted. We will label an edge in an execution graph as *left-increasing* or *right-increasing*, respectively, if the left-hand or right-hand expression labeling the start node has smaller size than the corresponding expression in the end node. *Left-decreasing* and *right-decreasing* labels are defined similarly.

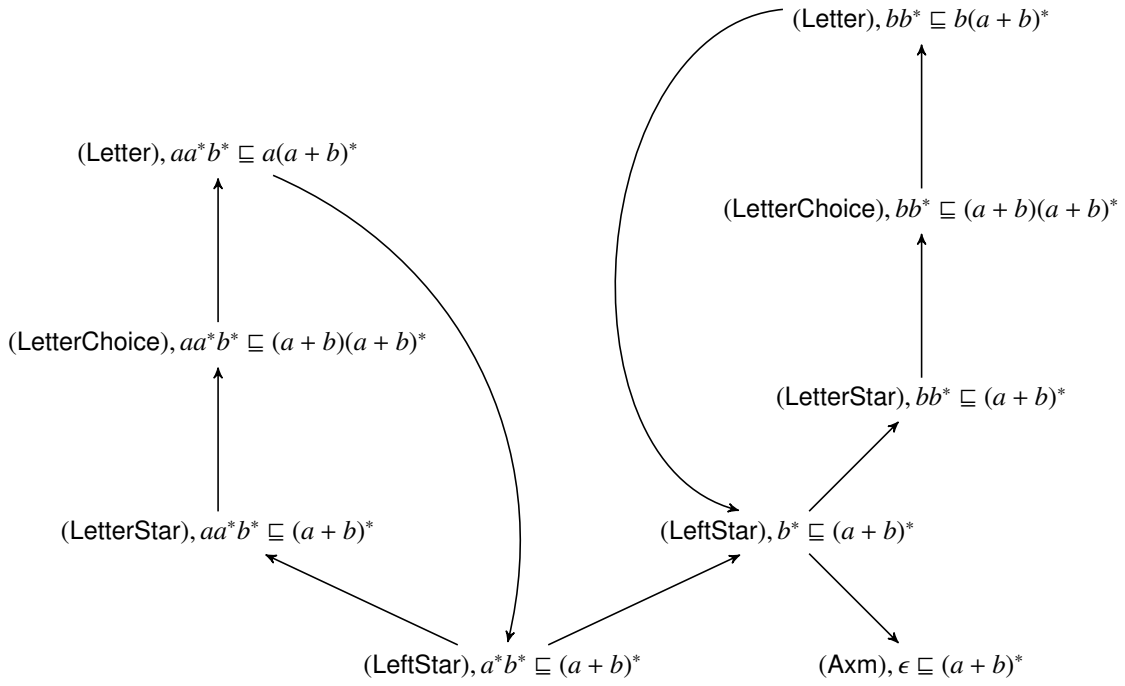

Figure 7: The execution graph corresponding to input a^*b^*, $(a+b)^*$. (cf. Fig. 3).

Nodes labeled (StarChoice2) and (LeftStar) have one left-increasing and left-decreasing outgoing edge. An edge is right-increasing if and only if it starts in a node labeled (LetterStar). Outgoing edges from all other rules are left-decreasing, right-decreasing, or both. If an edge is neither left-increasing nor left-decreasing then the expression on the left-hand side in the start and end node are the same. The similar statement holds for the right-hand side. The edges corresponding to usage of the store S have no labels, since by construction the expressions in the start and end node are the same.

Lemma 5.3. *If there is a left-increasing edge in a loop, then there is also a node labeled* (Letter) *in the loop.*

Proof. We prove a stronger statement, which implies that any path starting with a left-increasing edge, and not containing a node labeled (Letter) cannot be a loop: We show that in a path where there is no node labeled (Letter) and which starts with a left-increasing edge, the left-hand expressions in all nodes except the first node are of the form $r'_1 \cdots r'_n \cdot r_1^* r_2$ for some $r'_1 \cdots r'_n \notin \mathfrak{N}_\Sigma$. This is proved by induction on the length of the path. For the base case, only one edge in the path, note that the start node must be labeled (StarChoice2) or (LeftStar), so the left-hand expression in the first node is of the form $r_1^* r_2$ and the left-hand expression in the last node is $r_1 r_1^* r_2$. Now, $r_1 \notin \mathfrak{N}_\Sigma$ follows from the fact that the expressions are in star normal form.

There is an induction case for each premise of each rule. Note that (Axm) cannot be applied because of the induction hypothesis. For the premises corresponding to edges which are neither left-decreasing nor left-increasing, the left-hand expression is unchanged, and we can just use the induction hypothesis. For the premises corresponding to left-increasing edges, note that by the induction hypothesis the start node is of the form $r'_1 \cdots r'_n \cdot r_1^* r_2$, and the left-hand expression in the last node is $r' \cdot r'_1 \cdots r'_n \cdot r_1^* r_2$ for some r', and $r'_1 \cdots r'_n \notin \mathfrak{N}_\Sigma \Rightarrow r' \cdot r'_1 \cdots r'_n \notin \mathfrak{N}_\Sigma$.

The interesting cases are the premises corresponding to left-decreasing edges. For (LeftChoice), we can apply the induction hypothesis to get that the left-hand expression in the start node is $(r'_1 + r'_2) \cdot r'_3 \cdots r'_n \cdot r_1^* r_2$ where $(r'_1 + r'_2) \cdot r'_3 \cdots r'_n \notin \mathfrak{N}_\Sigma$. The last node has left-hand expression $r'_i \cdot r'_3 \cdots r'_n \cdot r_1^* r_2$ for $i \in \{1, 2\}$. But $(r'_1 + r'_2) \cdot r'_3 \cdots r'_n \notin \mathfrak{N}_\Sigma \Rightarrow r'_i \cdot r'_3 \cdots r'_n \notin \mathfrak{N}_\Sigma$, so the lemma holds also for the new last node.

For a left-decreasing edge, corresponding to a premise of (StarChoice2) or (LeftStar), we get from the induction hypothesis and Definition 2.3 that the left-hand expression in the starting node is of the form $r_1'^* \cdot r'_2 \cdots r'_n \cdot r_1^* r_2$ where

$r'_2 \cdots r'_n \notin \mathfrak{N}_\Sigma$. The left-hand expression in the last node is $r'_2 \cdots r'_n \cdot r_1^* r_2$, so the lemma holds also for this case. □

Lemma 5.4. *If there is a right-increasing edge in a loop, then there is also an instance of* (Letter) *in the loop.*

Proof. We prove a stronger statement which implies that any path starting with a right-increasing edge, and not containing a node labeled (Letter) cannot be a loop: In a path where there is no node labeled (Letter) and which starts with a right-increasing edge, all nodes except the first are of the form $l \cdot r_1 \sqsubseteq r'_1 \cdots r'_n \cdot r_2^* r_3$ for some $r'_1 \cdots r'_n \notin \mathfrak{N}_\Sigma$ where $l \in \mathsf{first}(r'_1 \cdots r'_n)$. This is proved by induction on the length of the path.

- For the base case, only one edge in the path, the first node must be labeled with (LetterStar) and $l \cdot r_1 \sqsubseteq r_2^* r_3$. The last node is then labeled $l \cdot r_1 \sqsubseteq r_2 r_2^* r_3$. That $r_2 \notin \mathfrak{N}_\Sigma$ follows from that the expressions are in star normal form. $l \in \mathsf{first}(r_2)$ is the side conditions on (LetterStar).

There are induction cases for all rules, but the only rules that can match the relation are (LetterStar), (LetterChoice) and (ElimCat).

- For (LetterStar), the conclusion must be of the form

$$l \cdot r_1 \sqsubseteq {r'_1}^* r'_2 \cdots r'_n \cdot r_2^* r_3$$

  where $r'_2 \cdots r'_n \notin \mathfrak{N}_\Sigma$. From the side-condition we get $l \in \mathsf{first}(r'_1)$. Thus the lemma holds for the premise $l \cdot r_1 \sqsubseteq r'_1 {r'_1}^* r'_2 \cdots r'_n r_2^* r_3$.

- For the cases matching (LetterChoice) the conclusion is of the form $l \cdot r_1 \sqsubseteq (r'_1 + r'_2) r'_3 \cdots r'_n r_2^* r_3$, and the premise is $l \cdot r_1 \sqsubseteq r'_i \cdot r'_3 \cdots r'_n r_2^* r_3$ where $i \in \{1, 2\}$, $l \in \mathsf{first}(r'_1)$ and $r'_i \cdot r'_3 \cdots r'_n \notin \mathfrak{N}_\Sigma$.

- For (ElimCat), the conclusion is of the form $l \cdot r_1 \sqsubseteq r'_1 \cdots r'_n \cdot r_2^* r_3$ for some $r'_1 \in \mathfrak{N}_\Sigma$, where $r'_1 \cdots r'_n \notin \mathfrak{N}_\Sigma$. The latter implies that $r'_2 \cdots r'_n \notin \mathfrak{N}_\Sigma$, and the side-conditions ensure that $l \in \mathsf{first}(r'_2 \cdots r'_n \cdot r_2^* r_3)$, which imply that $l \in \mathsf{first}(r'_2 \cdots r'_n)$. So the lemma also holds for the premise $l \cdot r_1 \sqsubseteq r'_2 \cdots r'_n \cdot r_2^* r_3$.

□

Lemma 5.5. *In any loop, there is at least one instance of* (Letter)

Proof. At least one rule instance in a loop is right- or left-increasing or -decreasing. This implies there must be at least one left- or right-increasing instance, and the result follows immediately from Lemmas 5.3 and 5.4. □

Definition 5.6 (Letter-path)**.** *A* letter-path *is a path in an execution graph of the algorithm where the last node is labeled* (Letter) *and there are no other nodes labeled* (Letter)*,*

Lemma 5.7 (Letter-path language conservation)**.** *In every letter-path, if the last node is labeled $lr_1 \sqsubseteq lr_2$ and the first node is labeled $r_3 \sqsubseteq r_4$, then $\|r_2\| \subseteq \{w \mid lw \in \|r_4\|\}$*

Proof. By induction on the length of the letter-path.

- The base case is a path consisting of a single node labeled (Letter). This case is immediate, as we get $r_4 = l \cdot r_2$.

There are induction cases for each of the rules shown in Table 2, except (Axm) and (Letter). The cases where the right-hand expression is unchanged ((LeftChoice), (LeftStar), and (StarChoice2)) hold immediately from the induction hypothesis.

- For (LetterStar), the right-hand expression in the label of the first node is of the form $r_5^* r_6$ and the induction hypothesis is that $\|r_2\| = \{w \mid lw \in \|r_5 r_5^* r_6\|\}$. The inclusion $\|r_5 r_5^* r_6\| \subseteq \|r_5^* r_6\|$ follows from Definition 2.2, thus we also get that $\{w \mid lw \in \|r_5 r_5^* r_6\|\} \subseteq \{w \mid lw \in \|r_5^* r_6\|\}$, so the lemma holds.

- For (LetterChoice) and (StarChoice1), the right-hand expression in the conclusion is of the form $(r_5 + r_6) r_7$, and by symmetry we can assume the right-hand expression in the premise is $r_5 r_7$, so the induction hypothesis is that $\|r_2\| \subseteq \{w \mid lw \in \|r_5 r_7\|\}$. But since $\|r_5 r_7\| \subseteq \|(r_5 + r_6) r_7\|$ follows from Definition 2.2 we also get that $\{w \mid lw \in \|r_5 r_7\|\} \subseteq \{w \mid lw \in \|(r_5 + r_6) r_7\|\}$, so the lemma holds.

- For (ElimCat), the right-hand expression in the conclusion is of the form $r_5 r_6$ where $r_5 \in \mathfrak{N}_\Sigma$, and the induction hypothesis is that $\|r_2\| \subseteq \{w \mid lw \in \|r_6\|\}$. But since $\|r_6\| \subseteq \|r_5 r_6\|$ follows from Definition 2.2 and $r_5 \in \mathfrak{N}_\Sigma$, we also get that $\{w \mid lw \in \|r_6\|\} \subseteq \{w \mid lw \in \|r_5 r_6\|\}$, so the lemma holds.

□

Lemma 5.8. *For any node $r_1 \sqsubseteq r_2$ in an execution graph, and for any $w \in \|r_1\|$, $w \neq \epsilon$, there is a letter-path from this node to an instance of* (Letter) *such that w is in the language of the left-hand expression in the conclusion of this instance of* (Letter)*.*

Proof. For all rules, except (Letter), the union of the languages of the left-hand expressions in the premise(s) equals the language of the left-hand expression in the conclusion. We can therefore construct the letter-path by repeatedly choosing the next node corresponding to a premise where the left-hand expression matches w. This process will terminate in an instance of (Letter) by the following arguments. Instances of (Axm) will not occur as $w \notin \|\epsilon\|$, and Lemma 5.5 assures that all loops contain at least one instance of (Letter). □

Lemma 5.9. *For any $r_1 \sqsubseteq r_2$ in an execution graph of the algorithm, $\|r_1\| \subseteq \|r_2\|$.*

Proof. The lemma can be reformulated, stating that for all $w \in \Sigma^*$, and all $r_1 \sqsubseteq r_2$ in the execution graph, $w \in \|r_1\|$ implies $w \in \|r_2\|$. We prove this simultaneously for all nodes in the execution graph, by induction on the length of w. The base case is that $w = \epsilon$. In this case $r_1 \in \mathfrak{N}_\Sigma$, and the algorithm guarantees that also $r_2 \in \mathfrak{N}_\Sigma$. The induction case is that $w = lw'$ for some $l \in \Sigma$ and $w' \in \Sigma^*$. Assume some $r_1 \sqsubseteq r_2$ in the execution graph, where $lw' \in \|r_1\|$. We must prove that $lw' \in \|r_2\|$. From Lemma 5.8 there is a letter-path starting with the instance with conclusion $r_1 \sqsubseteq r_2$, and ending in an instance of (Letter) with conclusion $lr_3 \sqsubseteq lr_4$ such that $w' \in \|r_3\|$. From using the induction hypothesis on w' and the premise $r_3 \sqsubseteq r_4$ of this instance of (Letter) we then get that $w' \in \|r_4\|$, and therefore $w = l \cdot w' \in \|lr_4\|$. Lemma 5.7 now states that $\|r_4\| \subseteq \{v \mid lv \in \|r_2\|\}$, so we get that $w \in \|r_2\|$. □

Soundness is now an immediate corollary of the previous lemma.

Theorem 5.10 (Soundness)**.** *Let r_1, r_2 be regular expressions. If the algorithm is run with r_1 and r_2 as input, and returns "Yes", then $\|r_1\| \subseteq \|r_2\|$.*

Proof. Since the input is r_1 and r_2 we know that $r_1 \sqsubseteq r_2$ occurs in the corresponding execution graph. From Lemma 5.9 we then get that $\|r_1\| \subseteq \|r_2\|$. □

Since the rules are invertible, and, as seen above, the algorithm always terminates, we get completeness almost for free.

Theorem 5.11 (Completeness)**.** *If $\|r_1\| \subseteq \|r_2\|$, the algorithm will either accept $r_1 \sqsubseteq r_2$, or it will report that the 1-ambiguity of r_2 is a problem.*

Proof. Since the rules are invertible, and the algorithm always terminates, all that remains is to show that for all regular expressions r_1 and r_2, where their languages are in an inclusion relation, there is at least one rule instance with conclusion $r_1 \sqsubseteq r_2$. But this follows directly from Lemma 4.1. □

6. Related Work and Conclusion

This paper is an extension of work in [11], and has previously appeared in the thesis [12]. Martens, Neven & Schwentick study in [13] the complexity of the inclusion problem for several sub-classes of the regular expressions. Colazzo, Ghelli & Sartiani, describe in [14] and [15] asymmetric polynomial-time algorithms for inclusion of a sub-class of regular expressions called collision-free. The collision-free regular expressions have at most one occurrence of each symbol from Σ, and the Kleene star can only be applied to disjunctions of letters. The latter class is strictly included in the class of 1-unambiguous regular expressions. The main focus of Colazzo, Ghelli and Sartiani is on the extensions of regular expressions used in XML Schemas. These extensions are not covered by the algorithm presented here. Hosoya et al. [16] study the inclusion problem for XML Schemas. They also use a syntax-directed inference

system, but the algorithm is not polynomial-time. Salomaa [17] presents two axiom systems for equality of regular expressions, but does not treat the run-time. The inference system used by our algorithm has some inspiration from the concept of derivatives of regular expressions, first defined by Brzozowski [18]. The first use of derivatives for the inclusion problem is by Brzozowski in [19]. Antimirov reinvents and details this approach in [20], as a term rewriting system for inequalities of regular expressions. Chen & Chen [21] adopt Antimirov's algorithm to the inclusion problem for 1-unambiguous regular expressions. They do not treat the left-hand and right-hand together in the way the rules of the algorithm in this paper do. The analysis of their algorithm depends on both the left-hand and the right-hand regular expressions being 1-unambiguous.

6.1. Conclusion

We have described a polynomial-time algorithm for language inclusion of regular expressions. The algorithm is based on a syntax-directed inference system, and is guaranteed to give the correct answer if the right-hand expression is 1-unambiguous. If the right-hand expression is 1-ambiguous the algorithm either reports an error or gives the correct answer. In certain cases, irrelevant parts of the right-hand expression are automatically discarded. This is the main advantage over the classical algorithms for inclusion. An implementation of the algorithm is available on the author's website.

7. Acknowledgments

This paper was written while the author was a PhD student at Department of Informatics, University of Bergen, Norway. The paper has benefited greatly from the input of my adviser, Marc Bezem. I also wish to thank Federico Mancini, Andrew Polonsky, and the anonymous reviewers.

7.1. Role of the Funding Source

This research has been funded by "The Research Council of Norway" through the project "Secure Heterogeneous Information Presentation" in the VERDIKT program.

References

[1] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proceedings of FOCS, IEEE, 1972, pp. 125–129.
[2] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
[3] A. Nerode, Linear automaton transformations, Proceedings of the American Mathematical Society 9 (1958) 541–544.
[4] ISO 8879. Information processing — text and office systems — standard generalized markup language (SGML), 1986.
[5] A. Brüggemann-Klein, Regular expressions into finite automata, Theoretical Computer Science 120 (1993) 197–213.
[6] A. Brüggemann-Klein, D. Wood, One-unambiguous regular languages, Information and Computation 140 (1998) 229–253.
[7] V. M. Glushkov, The abstract theory of automata, Uspekhi Mat. Nauk 16 (1961) 3–62. Translated in [22].
[8] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IRE Transactions on Electronic Computers 9 (1960) 39–47.
[9] M. Bezem, J. W. Klop, R. de Vrijer (Eds.), Term Rewriting Systems, Cambridge University Press, 2003.
[10] R. Book, S. Even, S. Greibach, G. Ott, Ambiguity in graphs and expressions, IEEE Transactions on Computers c-20 (1971) 149–153.
[11] D. Hovland, The inclusion problem for regular expressions, in: A. H. Dediu, H. Fernau, C. Martín-Vide (Eds.), LATA, volume 6031 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 309–320.
[12] D. Hovland, Feasible Algorithms for Semantics — Employing Automata and Inference Systems. `http://hdl.handle.net/1956/4325`, Ph.D. thesis, Universitetet i Bergen, 2010.
[13] W. Martens, F. Neven, T. Schwentick, Complexity of decision problems for simple regular expressions, in: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), MFCS, volume 3153 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 889–900.
[14] D. Colazzo, G. Ghelli, C. Sartiani, Efficient asymmetric inclusion between regular expression types, in: R. Fagin (Ed.), ICDT, volume 361 of *ACM International Conference Proceeding Series*, ACM, 2009, pp. 174–182.
[15] G. Ghelli, D. Colazzo, C. Sartiani, Efficient inclusion for a class of xml types with interleaving and counting, in: M. Arenas, M. I. Schwartzbach (Eds.), DBPL, volume 4797 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 231–245.
[16] H. Hosoya, J. Vouillon, B. C. Pierce, Regular expression types for XML, ACM Trans. Program. Lang. Syst. 27 (2005) 46–90.
[17] A. Salomaa, Two complete axiom systems for the algebra of regular events, J. ACM 13 (1966) 158–169.
[18] J. A. Brzozowski, Derivatives of regular expressions, J. ACM 11 (1964) 481–494.
[19] J. A. Brzozowski, Roots of star events, J. ACM 14 (1967) 466–477.
[20] V. M. Antimirov, Rewriting regular inequalities (extended abstract), in: H. Reichel (Ed.), FCT, volume 965 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 116–125.
[21] H. Chen, L. Chen, Inclusion test algorithms for one-unambiguous regular expressions, in: J. S. Fitzgerald, A. E. Haxthausen, H. Yenigün (Eds.), ICTAC, volume 5160 of *LNCS*, Springer, 2008, pp. 96–110.
[22] V. M. Glushkov, The abstract theory of automata, Russian Mathematical Surveys 16 (1961) 1–53.

Appendix A. Proofs

For completeness, proofs of the Lemmas from Section 2 are given here.

Proof of Lemma 2.4. By induction on the regular expression r. The base cases $r = \epsilon$ and $r \in \Sigma$, and the induction case where r is of the form r_1^* are immediate from Definitions 2.2 and 2.3.

For the induction case where $r = r_1 + r_2$, we first treat the direction from left to right, that is, we assume $\epsilon \in \|r_1 + r_2\|$, and will prove that $r \in \mathfrak{N}_\Sigma$. From Definition 2.2 this implies that $\epsilon \in \|r_1\|$ or $\epsilon \in \|r_2\|$. Using the induction hypothesis we get that $r_1 \in \mathfrak{N}_\Sigma$ or $r_2 \in \mathfrak{N}_\Sigma$. From Definition 2.3 we then get that $r \in \mathfrak{N}_\Sigma$, as needed. For the other direction, assume $r_1 + r_2 \in \mathfrak{N}_\Sigma$. From Definition 2.3 we then get that $r_1 \in \mathfrak{N}_\Sigma$ or $r_2 \in \mathfrak{N}_\Sigma$. But then the induction hypothesis implies that $\epsilon \in \|r_1\|$ or $\epsilon \in \|r_2\|$. By using Definition 2.2 we then get that $\epsilon \in \|r\|$, as needed.

The induction case where $r = r_1 \cdot r_2$ can be shown by replacing "+" with "·" and "or" with "and" in the previous paragraph. □

Proof of Lemma 2.5. We first prove $\mathsf{first}(r) = \{l \in \Sigma \mid \exists w : lw \in \|r\|\}$ by induction on r. The base cases are immediate from Definition 2.2 and Table 1. For the induction cases, we must also use the induction hypothesis for the subexpression(s) joined by the root operator. To prove that $\mathsf{first}(r)$ can be calculated in time $O(|r| \cdot |\Sigma|)$ we also use an induction on r, applying Table 1 and the fact that the first-set is not larger than the alphabet, and therefore each of the union operations can be done in time $O(|\Sigma|)$.

□

Proof of Lemma 2.9.

1. We first show that for any $r_1 \in R_\Sigma - \{\epsilon\}$ and any $r_2 \in R_\Sigma$, $\mathsf{header}(r_1, r_2)$ is in header form by induction on r_1. If r_1 is not a concatenation, then, since, $r_1 \neq \epsilon$, we get directly from the definitions that $\mathsf{header}(r_1, r_2) = r_1 \cdot r_2$ is in header form. Otherwise, if r_1 is of the form $r_1' \cdot r_2'$, we get that the result is a new call to header where the first argument is r_1'. Since we have assumed that ϵ prefixes are removed, $r_1' \neq \epsilon$. Thus we can apply the induction hypothesis to r_1' and get that the result is in header form.

   Now, by definition $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$. If $r = \epsilon$, $\mathsf{hdf}(r) = \epsilon$, which is in header form. Otherwise, we can use the result above to get that $\mathsf{header}(r, \epsilon)$ is in header form.

2. Since $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$ we only need to show that for any r_1, r_2 $\|\mathsf{header}(r_1, r_2)\| = \|r_1 r_2\|$. The latter follows almost directly from associativity of concatenation and neutrality of concatenation with ϵ.

3. If $r = \epsilon$, $\mathsf{hdf}(r) = \epsilon$ so we are done. Otherwise, since $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$, it is sufficient to prove by induction on $r \in R_\Sigma - \{\epsilon\}$ that for any $r_1', \ldots, r_m' \in R_\Sigma - \{\epsilon\}$ there are $r_1, \ldots, r_n \in R_\Sigma - \{\epsilon\}$ such that

$$\mathsf{header}(r, r_1' \cdots r_m' \cdot \epsilon) = r_1 \cdots r_n \cdot \epsilon$$

   If r is not a concatenation, then we get

$$\mathsf{header}(r, r_1' \cdots r_m' \cdot \epsilon) = r \cdot r_1' \cdots r_m' \cdot \epsilon$$

   Otherwise, if $r = r_1'' \cdot r_2''$ is a concatenation, we get a new call to header, where the first argument is r_1'' and the second argument is of the form required by the induction hypothesis. (Recall that ϵ prefixes have been removed). Therefore we can apply the induction hypothesis to get the result.

4. From the previous item, there are $r_1, \ldots, r_n \in R_\Sigma - \{\epsilon\}$ such that $\mathsf{hdf}(r) = r_1 \cdots r_n \cdot \epsilon$. If $n = 0$, the lemma holds since $\mathsf{hdf}(\epsilon) = \mathsf{header}(\epsilon, \epsilon) = \epsilon$. Otherwise, we get from the definitions of hdf and header

$$\begin{aligned}
\mathsf{hdf}(\mathsf{hdf}(r)) = \\
\mathsf{hdf}(r_1 \cdots r_n \cdot \epsilon) = \\
\mathsf{header}(r_1 \cdots r_n \cdot \epsilon, \epsilon) = \\
\mathsf{header}(r_1, r_2 \cdots r_n \cdot \epsilon) = \\
r_1 \cdots r_n \cdot \epsilon = \\
\mathsf{hdf}(r)
\end{aligned}$$

□

Proof of Lemma 2.12. 1. By induction on r. The base cases $r = \epsilon$ and $r \in \Sigma$ are immediate from Definitions 2.2 and 2.11 and the definition of $r[]$.

For the inductive case where $r = r_1 + r_2$, by Definition 2.11, $\mu(r_1 + r_2) = 1 \downarrow \mu(r_1) + 2 \downarrow \mu(r_2)$. Applying Definition 2.2 to the latter we get $\|\mu(r_1 + r_2)\| = \|1 \downarrow \mu(r_1)\| \cup \|2 \downarrow \mu(r_2)\|$. Hence, by definition of concatenating a position with a regular expression, $\|\mu(r_1 + r_2)\| = 1 \downarrow \|\mu(r_1)\| \cup 2 \downarrow \|\mu(r_2)\|$. By applying distributivity of $r[]$ over concatenation we get $r[\|\mu(r_1 + r_2)\|] = r[1 \downarrow \|\mu(r_1)\|] \cup r[2 \downarrow \|\mu(r_2)\|]$. Note now that for any $i \in \{1, 2\}$ and any $q \in \mathsf{pos}(r_i)$, we have $r[i \downarrow q] = r[\langle i \rangle][q] = r_i[q]$. Applying this we get $r[\|\mu(r_1 + r_2)\|] = r_1[\|\mu(r_1)\|] \cup r_2[\|\mu(r_2)\|]$. By applying the induction hypothesis we get $r[\|\mu(r_1 + r_2)\|] = \|r_1\| \cup \|r_2\|$. Hence, by Definition 2.2, $r[\|\mu(r_1 + r_2)\|] = \|r_1 + r_2\|$.

The inductive case where $r = r_1 \cdot r_2$, can be shown by replacing "+" and "$\cup$" with "$\cdot$" in the previous paragraph.

For the inductive case where $r = r_1^*$, by Definition 2.11, $\mu(r_1^*) = (1 \downarrow \mu(r_1))^*$. Applying Definition 2.2 to the latter we get $\|\mu(r_1^*)\| = \bigcup_{0 \le i} \|1 \downarrow \mu(r_1)\|^i$. Hence, by definition of concatenating a position with a regular expression, $\|\mu(r_1^*)\| = \bigcup_{0 \le i} (1 \downarrow \|\mu(r_1)\|)^i$. By applying distributivity of $r[]$ over union and concatenation we get $r[\|\mu(r_1^*)\|] = \bigcup_{0 \le i} (r[1 \downarrow \|\mu(r_1)\|])^i$. Note now that for any $q \in \mathsf{pos}(r_1)$, we have $r[1 \downarrow q] = r[\langle 1 \rangle][q] = r_1[q]$. Applying this we get $r[\|\mu(r_1^*)\|] = \bigcup_{0 \le i} (r_1[\|\mu(r_1)\|])^i$. By applying the induction hypothesis we get $r[\|\mu(r_1^*)\|] = \bigcup_{0 \le i} \|r_1\|^i$. Hence, by Definition 2.2, $r[\|\mu(r_1^*)\|] = \|r_1^*\|$.

2. By induction on r. The base case $r = \epsilon$ holds vacuously. The base case $r \in \Sigma$ holds immediately from Definition 2.11. For the inductive cases where $r = r_1 \cdot r_2$ or $r = r_1 + r_2$, we assume some $p \in \mathsf{sym}(\mu(r))$ and proceed to show that $\mu(r)[p] = p$. By Definition 2.11, $\mathsf{sym}(\mu(r)) = 1 \downarrow \mathsf{sym}(\mu(r_1)) \cup 2 \downarrow \mathsf{sym}(\mu(r_2))$. Hence, there is $i \in \{1, 2\}$ and $p' \in \mathsf{sym}(\mu(r_i))$ such that $p = i \downarrow p'$. By the induction hypothesis for r_i, $\mu(r_i)[p'] = p'$, hence $(i \downarrow \mu(r_i))[p'] = p$. Since $\mu(r)[p] = (1 \downarrow \mu(r_1) \cdot 2 \downarrow \mu(r_2))[i \downarrow p'] = (i \downarrow \mu(r_i))[p']$ we get $\mu(r)[p] = p$. The inductive case where $r = r_1^*$ is similar to the previous case, but easier.

3. By induction on r. The base cases, and the cases where $p = \langle\rangle$, hold directly from Definition 2.11.

For the inductive cases where $r = r_1 \cdot r_2$ or $r = r_1 + r_2$, and $p \neq \langle\rangle$, there is $i \in \{1, 2\}$ and $p' \in \mathsf{pos}(r_i)$ such that $p = i \downarrow p'$. We have $r[p] = r_i[p']$ and that $p \in \mathsf{sym}(\mu(r))$ iff $p' \in \mathsf{sym}(\mu(r_i))$. Hence, the lemma holds by applying the induction hypothesis for r_i.

For the inductive case where $r = r_1^*$ and $p \neq \langle\rangle$, we can use the same argument as in the previous case, except that i is set to 1.

□

Proof of Lemma 2.15. First we prove by induction on r_1, where $r_1 \neq \epsilon$, that if $r_1 \cdot r_2$ is 1-unambiguous, then $\mathsf{header}(r_1, r_2)$ is 1-unambiguous. The cases where $r_1 \neq \epsilon$ is not a concatenation hold immediately, as $\mathsf{header}(r_1, r_2) = r_1 \cdot r_2$. For the remaining cases, there are $n \ge 1$ and $r_1', \ldots, r_n' \in R_\Sigma - \{\epsilon\}$ such that either $n \ge 2$, $r_1 = r_1' \cdots r_n'$, and r_n' is not a concatenation, or $r_1 = r_1' \cdots r_n' \cdot \epsilon$. We first show that $r_1' \cdots r_n' \cdot r_2$ is 1-unambiguous. Let u, p, q, v, w as in Definition 2.14 such that $u \cdot p \cdot v, u \cdot q \cdot w \in \|\mu(r_1' \cdots r_n' \cdot r_2)\|$ and $(r_1' \cdots r_n' \cdot r_2)[p] = (r_1' \cdots r_n' \cdot r_2)[q] \in \mathsf{sym}(\mu(r_1' \cdots r_n' \cdot r_2))$. Note now that the u, p, q, v, w can easily be modified to get u', p', q', v', w' such that $p = q \Leftrightarrow p' = q'$, $u' \cdot p' \cdot v', u' \cdot q' \cdot w' \in \|\mu(r_1 \cdot r_2)\|$, and $(r_1 \cdot r_2)[p'] = (r_1 \cdot r_2)[q']$. But since $r_1 \cdot r_2$ 1-unambiguous by assumption, we get from Definition 2.14 that $p' = q'$. Therefore $p = q$. Thus $r_1' \cdots r_n' \cdot r_2$ is 1-unambiguous. By the induction hypothesis on r_1' this implies that $\mathsf{header}(r_1', r_2' \cdots r_n' \cdot r_2)$ is 1-unambiguous. Hence, $\mathsf{header}(r_1, r_2) = \mathsf{header}(r_1', r_2' \cdots r_n' \cdot r_2)$ is 1-unambiguous.

Secondly, we prove that if $r \neq \epsilon$ and r is 1-unambiguous, then also $r \cdot \epsilon$ is 1-unambiguous. Take any u, p, q, v, w as in Definition 2.14 for $r \cdot \epsilon$ such that $u \cdot p \cdot v, u \cdot q \cdot w \in \|\mu(r \cdot \epsilon)\|$ and $(r \cdot \epsilon)[p] = (r \cdot \epsilon)[q]$. It is easy to see that there are u', p', q', v', w' such that $u = 1 \downarrow u'$, $p = 1 \downarrow p'$, $q = 1 \downarrow q'$, $v = 1 \downarrow v'$, and $w = 1 \downarrow w'$. This implies that $u' \cdot p' \cdot v', u' \cdot q' \cdot w' \in \|\mu(r)\|$ and $r[p'] = r[q']$. We can use Definition 2.14 for r to get $p' = q'$. Therefore $p = q$ and $r \cdot \epsilon$ is 1-unambiguous.

Finally, if $r = \epsilon$, $\mathsf{hdf}(r) = \epsilon$ is 1-unambiguous. Otherwise, if $r \neq \epsilon$, we have by the previous paragraph that $r \cdot \epsilon$ is 1-unambiguous. By the paragraph above, this implies that $\mathsf{header}(r, \epsilon)$ is 1-unambiguous. Since $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$ we get that $\mathsf{hdf}(r)$ is 1-unambiguous. □

Proof of Lemma 2.16. We first prove by induction on r that if r is in star normal form, and r is ambiguous, then there are $u, u' \in \|\mu(r)\|$ such that $u \neq u'$ but $r[u] = r[u']$. The base cases hold vacuously.

For the induction case where $r = r_1 + r_2$, there must be a word w which is either generated in two ways by r_1 or by r_2, or which is generated by both r_1 and r_2. In the former case, it suffices to use the induction hypothesis for r_1 or r_2. In the latter case, we get $u \in \|\mu(r_1)\|$ and $u' \in \|\mu(r_2)\|$ such that $r_1[u] = w = r_2[u']$. Hence, $r[1 \downarrow u] = w = r[2 \downarrow u']$ and $1 \downarrow u \neq 2 \downarrow u'$.

For the induction case where $r = r_1 \cdot r_2$, let w be a witness that r is ambiguous. There must be $w_1 \in \|r_1\|$ and $w_2 \in \|r_2\|$ such that $w = w_1 \cdot w_2$. For $i \in \{1, 2\}$, if w_i can be generated in two ways by r_i, we get the result by the induction hypothesis for r_i. Otherwise, we get $w'_1 \in \|r_1\|$ and $w'_2 \in \|r_2\|$ such that $w_1 \neq w'_1$, $w_2 \neq w'_2$ and $w = w'_1 \cdot w'_2$. Furthermore, there are $u_1, u'_1 \in \|\mu(r_1)\|$ and $u_2, u'_2 \in \|\mu(r_2)\|$ such that $w_1 = r_1[u_1]$, $w'_1 = r_1[u'_1]$, $w_2 = r_2[u_2]$, and $w'_2 = r_2[u'_2]$. Hence, $r[1 \downarrow u_1 \cdot 2 \downarrow u_2] = r[1 \downarrow u'_1 \cdot 2 \downarrow u'_2]$ and $1 \downarrow u_1 \cdot 2 \downarrow u_2 \neq 1 \downarrow u'_1 \cdot 2 \downarrow u'_2$.

For the induction case where $r = r_1^*$, let w be a witness that r_1^* is ambiguous. Note that ϵ can only be generated in one way, since r is in star normal form and $r_1 \notin \mathfrak{N}_\Sigma$. Hence, $w \neq \epsilon$, and there must be $w_1, \ldots, w_n \in \|r_1\|$ such that $w = w_1 \cdots w_n$. If one of the w_i's is generated in two ways by r_1 we get the result from the induction hypothesis for r_1. Otherwise, there must be $w'_1, \ldots, w'_m \in \|r_1\|$ different from $w_1, \ldots, w_n$ such that $w = w'_1 \cdots w'_m$. Then there is i such that $w_i \neq w'_i$, but for $0 < j < i$, $w_j = w'_j$. Hence, there is $l \in \Sigma$ and $w' \in \Sigma^*$ such that either $w_i = w'_i l w'$ or $w_i l w' = w'_i$. The cases are symmetric, so we treat only $w_i = w'_i l w'$. Then there are $u_1, \ldots, u_n, u'_1, \ldots, u'_m \in \|\mu(r_1)\|$ such that $\forall j \in \{1, \ldots, n\} : w_j = r_1[u_j]$ and $\forall j \in \{1, \ldots, m\} : w'_j = r_1[u'_j]$. Hence, $r[1 \downarrow (u_1 \cdots u_n)] = w = r[1 \downarrow (u'_1 \cdots u'_m)]$. There are also $p, p' \in \mathsf{sym}(\mu(r_1))$ and $u', u'', u''' \in \mathsf{sym}(\mu(r_1))^*$ such that $u_i = u' p u''$, $u'_{i+1} = p' u'''$, $l = r_1[p] = r_1[p']$, $w' = r_1[u'']$, and $w'_i = r_1[u']$. Since $p' u''' \in \|\mu(r_1)\|$, $p' \in \mathsf{first}(\mu(r_1))$. If $u'_i \neq u'$ we get immediately $u'_1 \cdots u'_m \neq u_1 \cdots u_n$ and we are done. Otherwise, since $u'_i, u'_i p u'' \in \|\mu(r_1)\|$, we get $p \in \mathsf{followLast}(\mu(r_1))$. Since r is in star normal form $p \neq p'$, hence $u'_1 \cdots u'_m \neq u_1 \cdots u_n$.

We now proceed to show that the class of 1-unambiguous regular expressions is included in the class of unambiguous regular expressions. We prove the contra-positive statement. We assume that r is ambiguous and proceed to show that r is 1-ambiguous. If r is not in star normal form, we get that r is 1-ambiguous from Definitions 2.14 and 2.13. Otherwise, we get from the arguments above $u, u' \in \|\mu(r)\|$ such that $u \neq u'$ but $r[u] = r[u']$. Let u_1 be the longest common prefix of u and u'. Then there are p, q, v, w such that $u_1 pv, u_1 qw \in \|\mu(r)\|$, $p \neq q$ and $r[p] = r[q]$. By Definition 2.14 this means r is 1-ambiguous. □