The Inclusion Problem for Regular Expressions

Dag Hovland

Institutt for Informatikk, Universitetet i Bergen, Norway,
dag.hovland@uib.no

Abstract. This paper presents a new polynomial-time algorithm for the inclusion problem for certain pairs 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 unnecessary parts of the right-hand expression. In these cases 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, and if the 1-ambiguity of the right-hand expression becomes a problem, the algorithm will report this.

1 Introduction

The inclusion problem for regular expressions was shown PSPACE-complete by Meyer & Stockmeyer [10]. The input to the problem is two expressions, which we will call the *left-hand* expression and the *right-hand* expression, where 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 constructs an NFA recognizing the intersection of the language of the left-hand expression with the complement of the language of the right-hand expression, and finally 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 [11,7]. All the other steps, seen separately, are polynomial-time.

1-unambiguous regular expressions were introduced by Brüggemann-Klein & Wood [3,2]. They show a polynomial-time construction of DFAs from 1-unambiguous regular expressions. The algorithm above 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 algorithm above, the left-hand expression can be an arbitrary regular expression. An implementation of the algorithm is available from the website of the 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 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 algorithm described above 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 Sect. 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 1 (Regular Expressions). *The* regular expressions *over the language Σ are denoted R_Σ and defined in the following inductive manner:*

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

$r, r_1, r_2, \ldots$ are used as variables for regular expressions. The sign for concatenation, $\cdot$, will often be omitted. The regular expressions denoting the empty language are not included, as they are irrelevant to the results in this paper.

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 $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$. ϵ denotes the *empty word* of zero length, such that for all $w \in \Sigma^*$, $\epsilon \cdot w = w \cdot \epsilon = w$. Therefore we also assume $r\epsilon = \epsilon r = r$ for regular expressions r. 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 $L^0 = \{\epsilon\}$. $\mathsf{sym}(r)$ denotes the set of letters from Σ occurring in r.

Definition 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 $\epsilon \cdot \epsilon$, $\epsilon + \epsilon$ or ϵ^* can be removed in linear time, working bottom up. We therefore can safely assume there are no subexpressions of these forms. We use r^i as a short-hand for r concatenated with itself i times.

The First-set of a regular expression is the set of letters that can occur first in a word in the language, while the followLast-set is the set of letters which can follow a word in the language. An easy, linear time, algorithm for calculating the First-set has been given by many others, e.g., Glushkov [6] and Yamada & McNaughton [9].

Definition 3 (First and followLast). *[2,6,9]*

$$\mathsf{first}(r) = \{l \in \Sigma \mid \exists w : lw \in \|r\|\}$$

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

Definition 4 (Nullable Expressions). *[6,9] The* nullable *regular expressions are denoted $\mathfrak{N}$ and are defined inductively as follows:*

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

It can be proved by induction on the regular expressions, that $\mathfrak{N}$ are exactly the regular expressions that have ϵ in the language.

Definition 5 (Marked Expressions). *[6,9] If $r \in R_\Sigma$ is a regular expression, $\mu(r)$ is the marked expression, that is, the expression where every instance of any symbol from Σ is subscripted with an integer, starting with 1 at the left and increasing.*

For example, $\mu((a + b)^* a) = (a_1 + b_2)^* a_3$. The mapping $\sharp$ removes subscripts on letters, such that $\sharp(\mu(r)) = r$.

Definition 6 (Star Normal Form). *[3,2]: A regular expression is in* star normal form *iff for all subexpressions r^*: $r \notin \mathfrak{N}$ and $\mathsf{first}(\mu(r)) \cap \mathsf{followLast}(\mu(r)) = \varnothing$.*

Brüggemann-Klein & Wood described also in [3,2] a linear time algorithm mapping a regular expression to an equivalent expression in star normal form. We can therefore safely assume that all regular expressions are in star normal form.

Definition 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 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:

$$\mathsf{header}(\epsilon, r) = r$$
$$\mathsf{header}(r_1, r_2) = \begin{cases} \text{if } r_1 \text{ is of the form } r_3 \cdot r_4 : & \mathsf{header}(r_3, r_4 \cdot r_2) \\ \text{else:} & r_1 \cdot r_2 \end{cases}$$

For any regular expression r, $\mathsf{hdf}(r) = \mathsf{header}(r, \epsilon)$ is in header-form and recognizes the same language as r. hdf also preserves star normal form, as starred subexpressions are not altered.

2.1 1-Unambiguous Regular Expressions

Intuitively, a regular expression is 1-unambiguous if there is only one way a word in its language can be matched when working from left to right with only one letter of look-ahead.

Definition 8. *[3,2] A regular expression r is 1-unambiguous if for any two $upv, uqw \in \|\mu(r)\|$, where $p, q \in \mathsf{sym}(\mu(r))$ and $u, v, w \in \mathsf{sym}(\mu(r))^*$ such that $\sharp(p) = \sharp(q)$, we have $p = q$.*

Examples of 1-unambiguous regular expressions are $(a^* + b)^*$, $a(a + b)^*$ and $b^*a(b^*a)^*$, while $(\epsilon + a)a$ and $(a + b)^*a$ are not 1-unambiguous. An expression which is not 1-unambiguous is called 1-ambiguous. A language is called 1-unambiguous if there is a 1-unambiguous regular expression denoting it. Otherwise, the language is called 1-ambiguous.

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 [1]. From [1]: "A regular expression is called unambiguous if every tape in the event can be generated from the expression in one way only". It follows almost directly from the definitions that the class of 1-unambiguous regular expressions is included in the class of unambiguous regular expressions. The inclusion is strict, as for example the expression $(a+b)^*a$ is both unambiguous and 1-ambiguous. See also [3,2] for comparisons of unambiguity and 1-unambiguity.

Brüggemann-Klein and Wood [3] showed that there exist 1-ambiguous regular languages, e.g., $\|(a+b)^*(ac+bd)\|$. They also showed that a regular expression is 1-unambiguous if and only if all of its subexpressions also are 1-unambiguous. We will use this property below. Note at this point that hdf preserves 1-unambiguity.

Taking $u = \epsilon$ in Definition 8 it follows that if $l_n, l_m \in \mathsf{first}(\mu(r))$ and r 1-unambiguous, then $n = m$. This fact is employed by the algorithm below.

3 Rules for Inclusion

The algorithm is based on an inference system described inductively in Table 1 for a binary relation $\sqsubseteq$ over 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 are in the relation $\sqsubseteq$ if and only if their languages are in the inclusion relation.

We will say that $r_1 \sqsubseteq r_2$ *holds*, if it is also true that $\|r_1\| \subseteq \|r_2\|$. Each rule consists of a horizontal line with a conclusion below it, and none, one, or two premises above the line. Some rules also have *side-conditions* in square brackets. We only allow rule instances where the side-conditions are satisfied. Note that (StarChoice1) and (LetterChoice) each have only one premise.

Figure 1 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.

Figures 2 and 3 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. 3. 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.

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

Fig. 1: Algorithm for inclusion of regular expressions

Table 1. The rules for the relation $\sqsubseteq$

(Axm)
$$\frac{}{\epsilon \sqsubseteq r}\ [r \in \mathfrak{N}]$$

(Letter)
$$\frac{r_1 \sqsubseteq r_2}{l \cdot r_1 \sqsubseteq l \cdot r_2}$$

(LetterStar)
$$\frac{l \cdot r_1 \sqsubseteq r_2 r_2^* r_3}{l \cdot r_1 \sqsubseteq r_2^* r_3}\ [l \in \mathsf{first}(r_2)]$$

(LetterChoice)
$$\frac{l \cdot r_1 \sqsubseteq r_i r_4}{l \cdot r_1 \sqsubseteq (r_2 + r_3) r_4} \begin{bmatrix} i \in \{2,3\} \\ l \in \mathsf{first}(r_i) \end{bmatrix}$$

(LeftChoice)
$$\frac{\begin{array}{c} r_1 r_3 \sqsubseteq r_4 \\ r_2 r_3 \sqsubseteq r_4 \end{array}}{(r_1 + r_2) r_3 \sqsubseteq r_4}$$

(LeftStar)
$$\frac{\begin{array}{c} r_1 r_1^* r_2 \sqsubseteq r_3 r_4 \\ r_2 \sqsubseteq r_3 r_4 \end{array}}{r_1^* r_2 \sqsubseteq r_3 r_4} \begin{bmatrix} \mathsf{first}(r_1) \cap \mathsf{first}(r_3) \neq \varnothing \\ r_4 \neq \epsilon \vee r_2 \neq \epsilon \\ \exists l, r_5 : r_3 = l \vee r_3 = r_5^* \end{bmatrix}$$

(StarStarE)
$$\frac{r_1 \sqsubseteq r_2^*}{r_1^* \sqsubseteq r_2^*}$$

(StarChoice1)
$$\frac{r_1^* r_2 \sqsubseteq r_i r_5}{r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5} \begin{bmatrix} i \in \{3,4\} \\ \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_i) \neq \varnothing \\ \mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_i r_5) \\ r_2 \notin \mathfrak{N} \vee r_i \in \mathfrak{N} \end{bmatrix}$$

(StarChoice2)
$$\frac{\begin{array}{c} r_1 r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5 \\ r_2 \sqsubseteq (r_3 + r_4) r_5 \end{array}}{r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5} \begin{bmatrix} \mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 + r_4) \neq \varnothing \\ (r_2 \in \mathfrak{N} \wedge r_3 \notin \mathfrak{N}) \vee \mathsf{first}(r_1^* r_2) \not\subseteq \mathsf{first}(r_3 r_5) \\ (r_2 \in \mathfrak{N} \wedge r_4 \notin \mathfrak{N}) \vee \mathsf{first}(r_1^* r_2) \not\subseteq \mathsf{first}(r_4 r_5) \end{bmatrix}$$

(ElimCat)
$$\frac{r_1 \sqsubseteq r_3}{r_1 \sqsubseteq r_2 r_3} \begin{bmatrix} \exists l, r_4, r_5 : r_1 = l \cdot r_4 \ \vee \ r_1 = r_4^* r_5 \\ r_2 \in \mathfrak{N} \\ \mathsf{first}(r_1) \subseteq \mathsf{first}(r_3) \end{bmatrix}$$

4 Properties of the Algorithm

To aid the understanding of the algorithm and the rules, Table 2 shows what rules might apply for each combination of header-forms of the left-hand and right-hand expressions. The only combinations that are never matched are when the right-hand expression is ϵ while the left-hand expression is not, and the combinations where the left-hand expression is ϵ while the right-hand is of the form $l \cdot r$. That the former are not in the inclusion relation follows from the fact that subexpressions of the forms $\epsilon \cdot \epsilon$, $\epsilon + \epsilon$ and ϵ^* are not allowed, while the latter combinations follow from that $\epsilon \notin \|l \cdot r\|$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{Store}}{\text{(Letter)}\ a^*b^* \sqsubseteq (a+b)^*}}{\text{(LetterChoice)}\ aa^*b^* \sqsubseteq a(a+b)^*}}{\text{(LetterStar)}\ aa^*b^* \sqsubseteq (a+b)(a+b)^*}}{\text{(LeftStar)}\ aa^*b^* \sqsubseteq (a+b)^*} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{(Axm)}}{\text{(Letter)}\ \epsilon \sqsubseteq (a+b)^*}}{\text{(LetterChoice)}\ b \sqsubseteq b(a+b)^*}}{\text{(LetterStar)}\ b \sqsubseteq (a+b)(a+b)^*}}{\text{(StarStarE)}\ b \sqsubseteq (a+b)^*}}{b^* \sqsubseteq (a+b)^*}}{a^*b^* \sqsubseteq (a+b)^*}$$

Fig. 2: Example usage of the inference rules to decide $a^*b^* \sqsubseteq (a+b)^*$

Table 2. The rules that might apply for any combination of header-forms of the left-hand and right-hand expressions

Right / Left	ϵ	$l \cdot r$	$(r_1 + r_2) \cdot r_3$	$r_1^* \cdot r_2$
ϵ	(Axm)		(Axm)	(Axm)
$l \cdot r$		(Letter)	(ElimCat) (LetterChoice)	(ElimCat) (LetterStar)
$(r_1 + r_2) \cdot r_3$		(LeftChoice)	(LeftChoice)	(LeftChoice)
$r_1^* \cdot r_2$		(LeftStar)	(ElimCat) (StarChoice1) (StarChoice2)	(ElimCat) (LeftStar) (StarStarE)

4.1 Preservation of 1-Unambiguity

We must make sure that the rules given in Table 1 preserve 1-unambiguity for the *right-hand* expressions. That is, if the right-hand expression in the conclusion is 1-unambiguous, then the right-hand expression in all the premises are 1-unambiguous. 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 [3]. The only remaining rule is (LetterStar), where 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 must use the fact that all expressions are in star normal form (see Definition 6), thus $r_1 \notin \mathfrak{N}$, and $\mathsf{first}(\mu(r_1)) \cap \mathsf{followLast}(\mu(r_1)) = \varnothing$. Take u, p, q, v and w as in Definition 8, and assume (for contradiction) that $\sharp(p) = \sharp(q)$ and $p \neq q$. Since $r_1^* r_2$ and r_1 are 1-unambiguous and $r_1 \notin \mathfrak{N}$, we can by symmetry assume that p is from r_1 while q is from $r_1^* r_2$. This is only possible if $u \in \|\mu(r_1)\|$, $p \in \mathsf{followLast}(\mu(r_1))$, and q corresponds to a member of $\mathsf{first}(\mu(r_1))$ or of $\mathsf{first}(\mu(r_2))$. But since $\mathsf{first}(\mu(r_1)) \cap \mathsf{followLast}(\mu(r_1)) = \varnothing$, this means that also $r_1^* r_2$ is 1-ambiguous, which is a contradiction

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

Lemma 1. *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 2 and using Definition 8. For each case, we show that the existence of several rule instances with the same conclusion implies either that the right hand expression is 1-ambiguous, or that the side-conditions do not hold.

- The only rules of which there can be several instances with the same conclusion are ($\mathsf{StarChoice1}$) and ($\mathsf{LetterChoice}$). For ($\mathsf{LetterChoice}$), 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 ($\mathsf{StarChoice1}$), 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 the first-set of $r_3 r_5$. The expression $(r_3 + r_4) r_5$ is therefore 1-ambiguous.
- If instances of both ($\mathsf{ElimCat}$) and either ($\mathsf{LetterStar}$) or ($\mathsf{LetterChoice}$) match the pair of expressions, then the right-hand expression is of the form $r_2 r_3$, where $r_2 \in \mathfrak{N}$ and there is an l such that both $l \in \mathsf{first}(r_2)$ and $l \in \mathsf{first}(r_3)$. Therefore $r_2 r_3$ is 1-ambiguous.
- If instances of both ($\mathsf{ElimCat}$) and ($\mathsf{LeftStar}$) match the pair of expressions, then the relation is of the form $r_1^* r_2 \sqsubseteq r_3 r_4$, where $r_3 \in \mathfrak{N}$ and both $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_4)$ and $\mathsf{first}(r_3) \cap \mathsf{first}(r_1) \neq \varnothing$. This can only hold if $r_3 r_4$ is 1-ambiguous.
- If instances of both ($\mathsf{ElimCat}$) and either ($\mathsf{StarChoice1}$) or ($\mathsf{StarChoice2}$) had the same conclusion, then this conclusion is of the form $r_1^* r_2 \sqsubseteq (r_3 + r_4) r_5$, where $r_3 + r_4 \in \mathfrak{N}$ and both $\mathsf{first}(r_1^* r_2) \subseteq \mathsf{first}(r_5)$ and $\mathsf{first}(r_1^* r_2) \cap \mathsf{first}(r_3 + r_4) \neq \varnothing$. Therefore the right-hand expression $(r_3 + r_4) r_5$ is 1-ambiguous.
- It is not possible that instances of ($\mathsf{ElimCat}$) and ($\mathsf{StarStarE}$) have the same conclusion, because that would mean that r_3 in ($\mathsf{ElimCat}$) would be ϵ, and that cannot satisfy the side-conditions of ($\mathsf{ElimCat}$).
- It is neither possible to instantiate ($\mathsf{LeftStar}$) and ($\mathsf{StarStarE}$) with the same expressions below the line, as this would not satisfy the side-conditions of ($\mathsf{LeftStar}$).
- Finally, it is not possible to instantiate ($\mathsf{StarChoice1}$) and ($\mathsf{StarChoice2}$) with the same expressions below the line. The two last lines in the side-conditions of both rules prevent this.

4.2 Invertibility of the Rules

It is now not hard to prove that each of the rules given in Table 1 are *invertible*, in the sense that, for each rule instance, assuming that (1) the side-conditions hold and (2) no other rule instance matches the conclusion, then the conclusion holds if and only if the conjunction of the premises hold.

Proof. – 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. The absence of any premises is to be treated as an empty conjunction, which is always true.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\mathsf{Axm})}{(\mathsf{Letter})\ \ \epsilon \sqsubseteq \epsilon}}{(\mathsf{Letter})\ \ b \sqsubseteq b}}{(\mathsf{LetterChoice})\ \ ab \sqsubseteq ab}}{ab \sqsubseteq (a+(b+c)^*c(b+c)\cdots(b+c))b}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{Store}}{(\mathsf{Letter})\ \ (ab)^*a \sqsubseteq a(ba)^*}}{(\mathsf{LetterStar})\ \ b(ab)^*a \sqsubseteq ba(ba)^*}}{(\mathsf{Letter})\ \ b(ab)^*a \sqsubseteq (ba)^*}}{(\mathsf{LeftStar})\ \ ab(ab)^*a \sqsubseteq a(ba)^*} \quad \dfrac{\dfrac{(\mathsf{Axm})}{(\mathsf{Letter})\ \ \epsilon \sqsubseteq (ba)^*}}{a \sqsubseteq a(ba)^*}}{(ab)^*a \sqsubseteq a(ba)^*}$$

Fig. 3: Example usages of the inference rules

- For (Letter) we are just adding (removing) a single letter prefix to (from) both languages, and this preserves the inclusion relation.
- For (LetterStar), the conclusion is of the form $lr_1 \sqsubseteq r_2^*r_3$. We note first that $\|r_2r_2^*r_3\| \subseteq \|r_2^*r_3\|$, and therefore the premise implies the conclusion. For the other direction, note that since $l \in \mathsf{first}(r_2)$ and (ElimCat) does not match the conclusion, the l in r_2 must be the position used to match the first l in a word, and the premise must therefore also hold.
- For (LetterChoice), that the premise implies the conclusion follows from Definition 2, while showing the other direction depends on the fact that no other instance of (LetterChoice) nor (ElimCat) match the conclusion. The latter implies then that $l \notin (\mathsf{first}(r_{5-i}) \cup \mathsf{first}(r_4))$, so we must have the premise.
- For (LeftChoice), the implications follow from Definition 2.
- (LeftStar) and (StarChoice2) hold by Definition 2, as $\|r_1^*r_2\| = \|r_1r_1^*r_2\| \cup \|r_2\|$.
- For (StarStarE), note $\|r_1\| \subseteq \|r_1^*\|$. So, obviously, if $r_1^* \sqsubseteq r_2^*$, then also $r_1 \sqsubseteq r_2^*$. The other direction holds by first seeing that $\|r_1\| \subseteq \|r_2^*\|$ implies $\|r_1^*\| \subseteq \|r_2^{**}\|$, and secondly that $\|r_2^{**}\| = \|r_2^*\|$. Both are standard results from language theory.
- For (StarChoice1), that $\|r_1^*r_2\| \subseteq \|r_ir_5\|$ implies $\|r_1^*r_2\| \subseteq \|(r_3+r_4)r_5\|$, when $i \in \{3,4\}$, follows from Definition 2. The other direction follows from the assumption that no other rule instance matches the conclusion, combined with the third side-condition, which together imply that $\|r_1^*r_2\| \cap \|r_{7-i}r_5\| = \varnothing$.
- For (ElimCat), the fact that the premise implies the conclusion, can be seen using Definition 2 and $r_2 \in \mathfrak{N}$. For the other direction, note that since no other rule instance matches the conclusion $r_1 \sqsubseteq r_2r_3$, and since $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_3)$, we must have $\mathsf{first}(r_1) \cap \mathsf{first}(r_2) = \varnothing$. Therefore $\|r_1\| \cap \|r_2r_3\| = \|r_1\| \cap \|r_3\|$, and we get that $\|r_1\| \subseteq \|r_2r_3\|$ implies $\|r_1\| \subseteq \|r_3\|$.

□

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

The algorithm always terminates in polynomial time. Termination is guaranteed by two properties. First, the use of the store S means that any pair of regular expressions is treated at most once. Secondly, all regular expressions occurring in conclusions are either ϵ or of the form $r_1 \cdot r_2$, where r_1 is a subexpression of the corresponding expression input to the algorithm, while r_2 is unique for each r_1. Both properties can be shown by induction on the steps in an execution of the algorithm.

Since a regular expression has only a quadratic number of subexpressions, then the number of possible different rule instances in a run of the algorithm is $O(n^4)$, where n is the sum of the length of the regular expressions input to the algorithm. Since the work at each rule instance is polynomial in the size of the input to the algorithm, we get a polynomial run-time for the whole algorithm.

4.4 Soundness and Completeness

The only obstacle to showing soundness of the algorithm, is to show that our usage of the store is safe. Most critical is the use of the store to eliminate loops. To get an intuition as to why this is safe, we refer the reader to the right hand example in Fig. 3. Note that the conclusion holds if and only if $\forall i, i \geq 0$: $\|ab\|^i\{a\} \subseteq \|a(ba)^*\|$ This can be proved by an induction on i. The right-hand branch in Fig. 3 corresponds to the base case $i = 0$. And we get the induction case by taking the left-hand branch and replacing the $*$ in the left-hand expressions by ab repeated $i - 1$ times. We will use a similar observation to show that the use of the store is safe.

We model an execution of the algorithm as a directed tree. The internal nodes in this tree are rule instances, and the leaves are pairs where the first element is a pair of regular expressions and the second element is either (Axm), *Store* or *Fail*. Each node has, for each premise, an edge going either to a node with that conclusion, or to a leaf containing the corresponding pair of regular expressions.

With a *loop* in an execution of the algorithm, we mean a directed path in its tree, the start being an internal node and the end a leaf containing *Store*, such that the conclusion in the rule instance in the first node, corresponds to the pair of regular expressions in the leaf. The intuition is that this path would have been repeated indefinitely, *looped*, if the store S had not prevented it.

Let the *size* of a regular expression be the sum of the number of letters and operators $*$ and $+$ occurring in the expression. We will say that a rule instance in a directed path is *left-increasing* or *right-increasing*, respectively, if the left-hand or right-hand expression in the conclusion has smaller size than the corresponding expression in the next node in the loop. *Left-decreasing* and *right-decreasing* instances are defined similarly.

Instances of (StarChoice2) and (LeftStar) are either left-increasing or left-decreasing, while an instance is right-increasing if and only if it is an instance of (LetterStar). Instances of all other rules, except (Axm) and (ElimCat) are always left-decreasing, right-decreasing, or both. An instance which is neither left-increasing nor left-decreasing has the same expression on the left-hand side in the conclusion and the premise corresponding to the next node in the path. Except for certain instances of ($\mathsf{ElimCat}$), the same holds for the right-hand side.

Lemma 2. *In any loop, there is at least one right-increasing and one left-increasing instance.*

The proof is omitted for space considerations.

Remark at this point, that only the rules ($\mathsf{LeftStar}$) and ($\mathsf{StarChoice2}$) can have premises not containing starred sub-expressions which are in the left-hand expression of the conclusion. Thus, given a tree modeling an execution of the algorithm, in any directed path starting at a node where the left-hand expression has a subexpression r_1^* and going to a node where the left-hand expression does not contain such a subexpression, there is a left-decreasing instance where the conclusion has left-hand side $r_1^* r_2$ for some r_2.

Theorem 1 (Soundness). $(r_1 \sqsubseteq r_2) \Rightarrow \|r_1\| \subseteq \|r_2\|$

Proof. Assume a successful execution of the algorithm. Since the rules are invertible, and the base case (Axm) holds by definition of $\mathfrak{N}$, we only need to show that the usage of the store S was sound. The store is used in two different situations. The cases where the pair was added to the store in a different branch hold because the rules are used depth-first. The other cases correspond to the loops. From Lemma 2, every such leaf has a left-increasing parent node. If we can show that the conclusion $r_1^* r_2 \sqsubseteq r_3$ of these left-increasing nodes are true, we are done. We only need to show that for any $i > 0$, the branch rooted in the child (in the loop) of this node can be used to show that $\|r_1\|^i \|r_2\| \subseteq \|r_3\|$. This can be done by replacing $r_1 r_1^* r_2$ by $r_1^i r_2$ in the conclusion. The steps in the branch can be used in a similar way, except that the loop(s) will be *unrolled* at most $i-1$ times, and that at least $i-1$ left-increasing instances will be removed together with the subbranches corresponding to the premises with smaller left-hand expressions. At the ith minimal left-increasing instance we get that the conclusion is the same as the premise with the smaller left-hand expression, and can be treated by the corresponding branch. □

Theorem 2 (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 $r_1 \sqsubseteq r_2$. This is done by a case distinction on the header-forms of r_1 and r_2, using Tables 1 and 2 and Definitions 2 and 3, and noting that $\|r_1\| \subseteq \|r_2\|$ implies $\mathsf{first}(r_1) \subseteq \mathsf{first}(r_2)$. □

5 Related Work and Conclusion

Hosoya et al [8] study the inclusion problem for XML Schemas. They also use a syntax-directed inference system, but the algorithm is not polynomial-time.

Salomaa [12] presents an axiom systems for equality of regular expressions, but does not treat the run-time of doing inference in the system. The inference system used by our algorithm has some inspiration from the concept of derivatives of regular expressions, first defined by Brzozowski [4]. Chen & Chen [5] describe an algorithm for inclusion of 1-unambiguous regular expressions, which is based on derivatives, and which has some similarities with the algorithm presented in the present paper. They do not treat the left-hand and right-hand together in the way that the rules of the algorithm in this paper does. The analysis of their algorithm depends on both the left-hand and the right-hand regular expressions being 1-unambiguous.

We have described a polynomial-time algorithm for inclusion of regular expressions. The algorithm is based on a syntax-directed inference system, and is guaranteed to give an 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 answer. In addition, unnecessary parts of the right-hand expression are automatically discarded. This is an advantage over the classical algorithms for inclusion. An implementation of the algorithm is available on the author's website.

References

1. Book, R., Even, S., Greibach, S., Ott, G.: Ambiguity in graphs and expressions. IEEE Transactions on Computers c-20(2), 149–153 (1971)
2. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoretical Computer Science 120(2), 197–213 (1993)
3. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation 140(2), 229–253 (1998)
4. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
5. Chen, H., Chen, L.: Inclusion test algorithms for one-unambiguous regular expressions. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigün, H. (eds.) ICTAC. LNCS, vol. 5160, pp. 96–110. Springer (2008)
6. Glushkov, V.M.: The abstract theory of automata. Russian Mathematical Surveys 16(5), 1–53 (1961)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
8. Hosoya, H., Vouillon, J., Pierce, B.C.: Regular expression types for XML. ACM Trans. Program. Lang. Syst. 27(1), 46–90 (2005)
9. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers 9, 39–47 (1960)
10. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of FOCS. pp. 125–129. IEEE (1972)
11. Nerode, A.: Linear automaton transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
12. Salomaa, A.: Two complete axiom systems for the algebra of regular events. J. ACM 13(1), 158–169 (1966)