Journal of Computer and System Sciences 78 (2012) 1795–1813

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

The inclusion problem for regular expressions [☆]

Dag Hovland [*]

Department of Informatics, University of Bergen, Norway

ARTICLE INFO

Article history:
Received 2 October 2010
Received in revised form 10 January 2011
Accepted 17 November 2011
Available online 21 December 2011

Keywords:
Formal languages
Regular expressions
Inclusion
1-unambiguity

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.

1. Introduction

The inclusion problem for regular expressions was shown PSPACE-complete in Meyer and 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 and 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 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.

[*] Present address: Department of Informatics, University of Oslo, Postboks 1080 Blindern, N-0316 Oslo, Norway.
E-mail address: hovlanddag@gmail.com.

0022-0000/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2011.12.003

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 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 $(a+(b+c)^* \times c(b+c)\cdots(b+c))b$, 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: $\|r_1 + r_2\| = \|r_1\| \cup \|r_2\|$, $\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\|$, $\|r^*\| = \bigcup_{0 \leqslant i} \|r\|^i$ 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).* (See [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.

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 and McNaughton [8].

Lemma 2.5 (first)*. (See [7,8].) For any regular expression r, $\mathsf{first}(r) = \{l \in \Sigma \mid \exists w\colon lw \in \|r\|\}$ 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.)*