Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 306 (2003) 373–390

Theoretical
Computer Science

www.elsevier.com/locate/tcs

Reducing NFAs by invariant equivalences☆

Lucian Ilie*,[1], Sheng Yu[2]

Department of Computer Science, University of Western Ontario, London, ON, Canada N6A 5B7

Received 22 April 2002; accepted 1 May 2003
Communicated by G. Rozenberg

Abstract

We give new general methods for constructing small non-deterministic finite automata (NFA) from arbitrary ones. Given an NFA, we compute the largest right-invariant equivalence on the set of states and then merge the equivalent states to obtain a smaller automaton. When applying this method to position automata, we get a way to convert regular expressions into NFAs which are always smaller than or equal to the position, partial derivative, and follow automata; it can be arbitrarily smaller. The construction can be dually made for left-invariant equivalences and then the two can be combined for even better results.

Keywords: Non-deterministic finite automata; Regular expressions; Automata minimization; Invariant equivalences; Derivatives of regular expressions

1. Introduction

The importance of regular expressions for applications is well known. They describe lexical tokens for syntactic specifications and textual patterns in text manipulation systems. Regular expressions have become the basis of standard utilities such as scanner generators (lex), editors (emacs, vi), or programming languages (perl, awk), see [1,9]. The implementation of the regular expressions is done using finite automata. As the

☆ An extended abstract of this paper has been presented at The 27th International Symposium on Mathematical Foundations of Computer Science (MFCS'02) (Warsaw, 2002); see [16].

* Corresponding author.
E-mail address: ilie@csd.uwo.ca (L. Ilie).

[1] Research partially supported by NSERC Grant R3143A01.
[2] Research partially supported by NSERC Grant OGP0041630.

0304-3975/03/$ - see front matter
doi:10.1016/S0304-3975(03)00311-6

deterministic finite automata (DFA) obtained from regular expressions can be exponentially larger in size and minimal non-deterministic ones (NFA) are hard to compute (the problem is PSPACE-complete, see [21]), other methods need to be used to find small automata. The idea is to find automata which are not minimal but "reasonably" small and which can be constructed efficiently.

In the case of NFAs with ε-transitions (εNFA) the classical solution is due to Thompson [20]. It constructs εNFAs which have linear size. There are also improvements of this, e.g., the one of Sippu and Soisalon-Soininen [19] which builds smaller εNFAs than Thompson's. Still, all these automata build εNFAs of linear size and so the differences between those are not very big.

For NFAs (without ε-transitions), the situation is much less clear. A well-known method for constructing NFAs is the position automaton, due to Glushkov [10] and McNaughton–Yamada [18]; the size of their NFA is between linear and quadratic and can be computed by the algorithm of Brüggemann–Klein [4] in quadratic time.

There are several other constructions, of which we mention the most important ones. Antimirov [2] constructed an NFA based on partial derivatives. Champarnaud and Ziadi [6,7] improved very much Antimirov's $\mathcal{O}(n^5)$ algorithm for the construction of such NFA; their algorithm runs in quadratic time. They proved also that the partial derivative automaton is a quotient of the position automaton and so it is always smaller. Hromkovic et al. [14] gave the construction with the best worst case so far—the common follow sets automaton; this NFA has size at most $\mathcal{O}(n(\log n)^2)$ and, by the algorithm of Hagenah and Muscholl [11], it can be computed in time $\mathcal{O}(n(\log n)^2)$. As discussed later, this construction performs poorly in practice.

It is unexpected that this very important problem, of computing small NFAs from regular expressions, did not receive more attention. Also, the more general problem of reducing the size of arbitrary NFAs was even less investigated. We address these problems and give new methods to construct small finite automata from regular expressions as well as new methods to reduce the size of arbitrary NFAs.

Our basic idea resembles the minimization algorithm for DFAs where indistinguishable states are first computed and then merged; see, e.g., [13]. The idea cannot be applied directly to NFAs and we give a discussion on what is the best way to do it. It is unexpected that this old idea was not used for reducing NFAs. This is probably because in the case of DFAs a very nice mathematical object is obtained—the (unique) minimal DFA—while for NFAs this is hopeless. However, an interesting mathematical object can also be obtained in the case of NFAs. We show how to compute the largest right-invariant equivalence on states and then merge states according to this equivalence.

The above idea can be easily employed to construct small NFAs [16] from regular expressions. Simply take any NFA obtained from a regular expression and reduce it according to the algorithm. The automata we use as a start are either the position automaton or quotients of it. We compare our NFAs to the position and partial derivative automata, as well as with the new follow automata introduced by the authors in [15].

Follow automata are a new type of automata obtained from regular expressions and seem to have several remarkable properties. The idea is to build first εNFAs which are always smaller than the ones of Thompson [20] or Sippu and Soisalon-Soininen [19].

Their size is at most $\frac{3}{2}|\alpha| + \frac{5}{2}$ which is very close to the optimal because of the lower bound $\frac{4}{3}|\alpha| + \frac{5}{2}$. Then, elimination of ε-transitions from these εNFAs is used. Although this method of constructing NFAs has, apparently, nothing to do with positions, it turns out, unexpectedly, that the NFA it produces is a quotient of the position automaton with respect to the equivalence given by the following relation, therefore giving the name of follow automaton. The follow automaton is always smaller than or equal to the position automaton and incomparable with the partial derivative or common follow sets automaton.

The NFA we obtain from a regular expression by state merging is always smaller than the partial derivative automaton and follow automaton since both are quotients of the position automaton with respect to right-invariant equivalences. Moreover, it can be arbitrarily smaller! Even if the size in the worst case is quadratic, on most examples it is smaller than all known constructions and seems to be a good candidate for the best method to reduce the size of NFAs. We bring this idea further by duality: NFAs can be also reduced using left-invariant equivalences. Moreover, the two can be combined for even better results.

2. Basic notions

We recall here the basic definitions we need throughout the paper. For further details we refer to [13] or [21].

Let A be an alphabet and A^* the set of all words over A; ε denotes the empty word and the length of a word w is denoted $|w|$. A *language* over A is a subset of A^*. A *regular expression* over A is $\emptyset$, ε, or $a \in A$, or is obtained from these applying the following rules finitely many times: for two regular expressions α and β, the *union*, $\alpha + \beta$, the *catenation*, $\alpha \cdot \beta$, and the *star*, α^*, are regular expressions. For a regular expression α, the language denoted by it is $L(\alpha)$, and $\varepsilon(\alpha)$ is ε if $\varepsilon \in L(\alpha)$ and $\emptyset$ otherwise. Also, $|\alpha|$ and $|\alpha|_A$ denote the size of α and the number of occurrences of letters from A in α, respectively (parentheses are not counted in $|\alpha|$).

A *finite automaton* is a quintuple $M = (Q, A, \delta, q_0, F)$, where Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ is the transition mapping; we shall denote, for $p \in Q, a \in A \cup \{\varepsilon\}$, $\delta(p,a) = \{q \in Q \mid (p,a,q) \in \delta\}$. The automaton M is called *deterministic* (DFA) if $\delta : Q \times A \to Q$ is a (partial) function, *non-deterministic* (NFA) if $\delta \subseteq Q \times A \times Q$, and *non-deterministic with ε-transitions* (εNFA) if there are no restrictions on δ. The *language* recognized by M is denoted $L(M)$. The *size* of a finite automaton M is $|M| = |Q| + |\delta|$, that is, we count both states and transitions.

Let $\equiv$ be an equivalence relation over Q. For $q \in Q$, $[q]_\equiv$ denotes the equivalence class of q w.r.t. $\equiv$ and, for $S \subseteq Q$, $S/_\equiv$ denotes the quotient set $S/_\equiv = \{[q]_\equiv \mid q \in S\}$. We say that $\equiv$ is *right invariant* w.r.t. M if and only if two conditions (i) and (ii) hold true:

(i) $\equiv \subseteq (Q - F)^2 \cup F^2$ (final and non-final states are not $\equiv$-equivalent),

(ii) for any $p, q \in Q$, $a \in A$, if $p \equiv q$, then $\delta(p,a)/_\equiv = \delta(q,a)/_\equiv$ (equivalent states lead to equivalent states by any letter).

If $\equiv$ is right invariant, the *quotient automaton* $M/_{\equiv}$ is constructed by $M/_{\equiv} = (Q/_{\equiv}, A, \delta_{\equiv}, [q_0]_{\equiv}, F/_{\equiv})$ where $\delta_{\equiv} = \{([p]_{\equiv}, a, [q]_{\equiv}) \mid (p, a, q) \in \delta\}$. Notice that $Q/_{\equiv} = (Q - F)/_{\equiv} \cup F/_{\equiv}$, so we do not merge final with non-final states. Also, we have that $L(M/_{\equiv}) = L(M)$.

3. Position automata

We recall in this section the well-known construction of the position automaton, discovered independently by Glushkov [10] and McNaughton and Yamada [18].

Let α be a regular expression. The basic idea of the position automaton is to assume that all occurrences of letters in α are different. For this, all letters are made different by marking each letter with a unique index, called its *position* in α. The set of positions of α is $\mathsf{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_A\}$. We shall denote also $\mathsf{pos}_0(\alpha) = \mathsf{pos}(\alpha) \cup \{0\}$. The expression obtained from α by marking each letter with its position is denoted $\bar{\alpha} \in \bar{A}^*$, where $\bar{A} = \{a_i \mid a \in A, 1 \leqslant i \leqslant |\alpha|_A\}$. For instance, if $\alpha = a(baa + b^*)$, then $\bar{\alpha} = a_1(b_2a_3a_4 + b_5^*)$. Notice that $\mathsf{pos}(\alpha) = \mathsf{pos}(\bar{\alpha})$.

Three mappings first, last, and follow are then defined as follows (see [10]). For any regular expression α and any $i \in \mathsf{pos}(\alpha)$, we have

$$\begin{aligned} \mathsf{first}(\alpha) &= \{i \mid a_i w \in L(\bar{\alpha})\}, \\ \mathsf{last}(\alpha) &= \{i \mid w a_i \in L(\bar{\alpha})\}, \\ \mathsf{follow}(\alpha, i) &= \{j \mid u a_i a_j v \in L(\bar{\alpha})\}. \end{aligned} \tag{1}$$

For reasons that are made clear later, we extend $\mathsf{follow}(\alpha, 0) = \mathsf{first}(\alpha)$. Also, let $\mathsf{last}_0(\alpha)$ stand for $\mathsf{last}(\alpha)$ if $\varepsilon(\alpha) = \emptyset$ and $\mathsf{last}(\alpha) \cup \{0\}$ otherwise.

The *position automaton* for α is

$$\mathbf{A}_{\mathrm{pos}}(\alpha) = (\mathsf{pos}_0(\alpha), A, \delta_{\mathrm{pos}}, 0, \mathsf{last}_0(\alpha)),$$

where

$$\delta_{\mathrm{pos}} = \{(i, a, j) \mid j \in \mathsf{follow}(\alpha, i), a = \overline{a_j}\}.$$

Glushkov [10] and McNaughton and Yamada [18] proved the following result which states that the position automaton works as intended.

Theorem 1. *For any regular expression α, we have $L(\mathbf{A}_{\mathrm{pos}}(\alpha)) = L(\alpha)$.*

We give next an example of a position automaton. The same example will be used also later to show other constructions. It has been carefully contrived to show the differences between various constructions.

Example 2. Consider the regular expression $\tau = (a + b)(a^* + ba^* + b^*)^*$. The marked version of τ is $\bar{\tau} = (a_1 + b_2)(a_3^* + b_4a_5^* + b_6^*)^*$. The values of the mappings first, last, and follow for τ and the corresponding position automaton $\mathbf{A}_{\mathrm{pos}}(\tau)$ are given in Fig. 1.

$\mathsf{first}(\tau) = \{1, 2\}$
$\mathsf{last}(\tau) = \{1, 2, 3, 4, 5, 6\}$

i	$\mathsf{follow}(\tau, i)$
0	$\{1, 2\}$
1	$\{3, 4, 6\}$
2	$\{3, 4, 6\}$
3	$\{3, 4, 6\}$
4	$\{3, 4, 5, 6\}$
5	$\{3, 4, 5, 6\}$
6	$\{3, 4, 6\}$

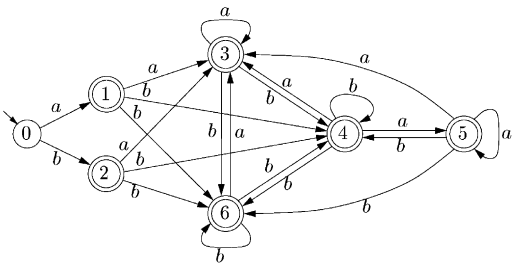

Fig. 1. $\mathbf{A}_{\mathrm{pos}}(\tau)$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

Te position automaton can be computed easily in cubic time using the inductive definitions of first, last, and follow, but Brüggemann-Klein [4] showed how to compute it in quadratic time.

4. Partial derivative automata

Another construction we recall is the partial derivative automaton, introduced by Antimirov [2]. Recall first the notion of partial derivative introduced by Antimirov. For a regular expression α and a letter $a \in A$, the set $\partial_a(\alpha)$ of partial derivatives of α w.r.t. a is defined inductively as follows:

$$\begin{aligned}
\partial_a(\varepsilon) &= \partial_a(\emptyset) = \emptyset, \\
\partial_a(b) &= \begin{cases} \{\varepsilon\} & \text{if } a = b, \\ \emptyset & \text{otherwise,} \end{cases} \\
\partial_a(\alpha + \beta) &= \partial_a(\alpha) \cup \partial_a(\beta), \\
\partial_a(\alpha\beta) &= \begin{cases} \partial_a(\alpha)\beta & \text{if } \varepsilon(\alpha) = \emptyset, \\ \partial_a(\alpha)\beta \cup \partial_a(\beta) & \text{if } \varepsilon(\alpha) = \varepsilon, \end{cases} \\
\partial_a(\alpha^*) &= \partial_a(\alpha)\alpha^*.
\end{aligned}$$

The definition of partial derivatives is extended to words by $\partial_\varepsilon(\alpha) = \{\alpha\}$, $\partial_{wa}(\alpha) = \partial_a(\partial_w(\alpha))$, for any $w \in A^*$, $a \in A$.

The set of all partial derivatives of α is denoted

$$\mathrm{PD}(\alpha) = \{\partial_w(\alpha) \mid w \in A^*\}.$$

Antimirov [2] constructed the *partial derivative automaton*

$$\mathbf{A}_{\mathrm{pd}}(\alpha) = (\mathrm{PD}(\alpha), A, \delta_{\mathrm{pd}}, \alpha, F_{\mathrm{pd}}),$$

$\partial_a(\tau) = \{\tau_1\}$ $\quad \tau_1 = (a^* + ba^* + b^*)^*$
$\partial_b(\tau) = \{\tau_1\}$
$\partial_a(\tau_1) = \{\tau_2\}$ $\quad \tau_2 = a^*\tau_1$
$\partial_b(\tau_1) = \{\tau_2, \tau_3\}$ $\tau_3 = b^*\tau_1$
$\partial_a(\tau_2) = \{\tau_2\}$
$\partial_b(\tau_2) = \{\tau_2, \tau_3\}$
$\partial_a(\tau_3) = \{\tau_2\}$
$\partial_b(\tau_3) = \{\tau_2, \tau_3\}$

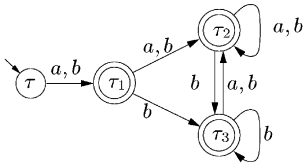

Fig. 2. $\mathbf{A}_{\mathrm{pd}}(\tau)$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

where

$$\delta_{\mathrm{pd}}(q,a) = \partial_a(q) \quad \text{for any } q \in \mathrm{PD}(\alpha),\ a \in A,$$
$$F_{\mathrm{pd}} = \{q \in \mathrm{PD}(\alpha) \mid \varepsilon(q) = \varepsilon\}.$$

He proved the following results.

Theorem 3. *For any regular expression* α, *we have*
(i) $|\mathrm{PD}(\alpha)| \leqslant |\alpha|_A + 1$,
(ii) $L(\mathbf{A}_{\mathrm{pd}}(\alpha)) = L(\alpha)$.

Example 4. Consider the regular expression τ from Example 2. The partial derivatives of τ are computed in Fig. 2 where also its partial derivative automaton $\mathbf{A}_{\mathrm{pd}}(\tau)$ is shown.

Antimirov's algorithm to compute the partial derivative automaton runs in quintic time. Champarnaud and Ziadi [6,7] proved that the partial derivative automaton is a quotient of the position automaton and gave also an algorithm to compute the partial derivative automaton in quadratic time.

The right-invariant equivalence they use is defined as follows. For a regular expression α and a letter $a_i \in \bar{A}$, the continuation of a_i in α, denoted $c_i(\bar{\alpha})$, is, according to Berry and Sethi [3], the unique expression (modulo associativity, commutativity and idempotence of union) $(wa_i)^{-1}(\bar{\alpha}) \neq \emptyset$, where $w \in \bar{A}^*$. Here the $^{-1}$ operator stands for the total derivative of Brzozowski [5].[3] The equivalence $\equiv_c \subset (\mathsf{pos}(\alpha))^2$ is defined by $i \equiv_c j$ iff $\overline{c_i(\bar{\alpha})} = \overline{c_j(\bar{\alpha})}$; here the inner bars stand for marking while the outer ones stand for unmarking (removing indices). Champarnaud and Ziadi proved that

Theorem 5. (i) $\equiv_c$ *is a right-invariant equivalence w.r.t.* $\mathbf{A}_{\mathrm{pos}}(\alpha)$ *and*
(ii) $\mathbf{A}_{\mathrm{pd}}(\alpha) \simeq \mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_c}$.

Next is an example of an application of Theorem 5.

[3] One should be aware of the difference between Antimirov's partial derivatives and Brzozowski's total derivatives, since they are very similar. Essentially, the former gives a set of expressions while the latter constructs a single expression. Therefore, the former leads to NFAs while the latter builds DFAs.

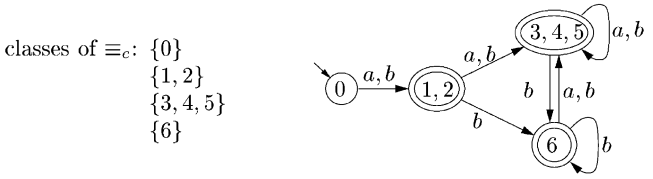

Fig. 3. $\mathbf{A}_{\mathrm{pd}}(\tau) \simeq \mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_c}$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

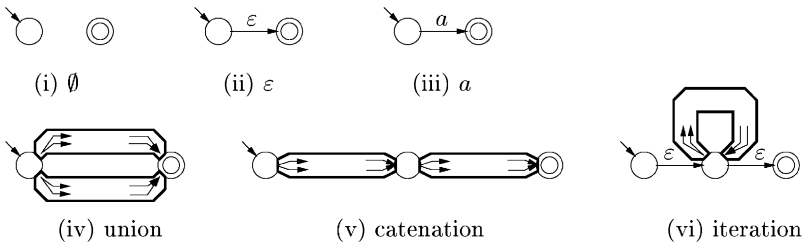

Fig. 4. The construction of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$.

Example 6. For the same regular expression τ from Example 2, we show in Fig. 3 the equivalence classes of $\equiv_c$ and the automaton $\mathbf{A}_{\mathrm{pd}}(\tau)$ as a quotient of $\mathbf{A}_{\mathrm{pos}}(\tau)$; compare to Figs. 1 and 2.

5. Follow automata

A new type of automata obtained from regular expressions was introduced by the authors in [15]; see also [17]. We recall it in this section. The idea is to construct first an εNFA (Algorithm 7) from a given regular expression and then an NFA by ε-elimination in the εNFA (Algorithm 9).

Algorithm 7. Given a regular expression α, the algorithm constructs an εNFA for α inductively, following the structure of α, and is shown in Fig. 4. The steps should be clear from the figure but we bring further improvements at each step which, rather informally, are described as follows (precise details are given in [17]):

(a) if $p \xrightarrow{\varepsilon} q$ and the outdegree of p (or indegree of q) is one, then p and q can be merged,

(b) any cycle of ε-transitions can be collapsed (all states merged and transitions removed),

(c) multiple transitions (same source, target, and label) are removed.

The obtained non-deterministic finite automaton with ε-transitions is called the *follow* εNFA (the reason for this name will be clear later) and denoted $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\alpha) = (Q_f^{\varepsilon}, A, \delta_f^{\varepsilon}, 0_f, q_f)$.

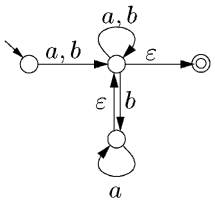

Fig. 5. $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\tau)$ for $\tau=(a+b)(a^*+ba^*+b^*)^*$.

Example 8. An example of the construction in Algorithm 7 for our running example $\tau=(a+b)(a^*+ba^*+b^*)^*$ is given in Fig. 5.

The basic idea of ε-elimination is given in Algorithm 9. For the precise way the ε-elimination is done, we refer to [17].

Algorithm 9. For any path labelled ε between two states $p,q\in Q_f^{\varepsilon}$ in $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$, $p\overset{\varepsilon}{\rightsquigarrow}q$, and any transition $q\xrightarrow{a}r$, add a transition $p\xrightarrow{a}r$ if there is no such transition already. Also, if q is a final state of $\mathbf{A}_{\mathrm{f}}^{\varepsilon}$, then make p a final state. Then remove all ε-transitions and unreachable states.

The obtained NFA is called the *follow* NFA, denoted $\mathbf{A}_{\mathrm{f}}(\alpha)$. As shown by the authors in [15,17], $\mathbf{A}_{\mathrm{f}}(\alpha)$ is a quotient of the position automaton. The equivalence involved is $\equiv_f\ \subseteq \mathsf{pos}_0(\alpha)^2$, defined by

$$i\equiv_f j \quad \text{iff} \quad \begin{array}{l}\text{(i) both } i,j \text{ or none belong to } \mathsf{last}(\alpha) \text{ and}\\ \text{(ii) } \mathsf{follow}(\alpha,i)=\mathsf{follow}(\alpha,j).\end{array}$$

Notice that we restrict the equivalence so that we do not make equivalence between a final and a non-final state in $\mathbf{A}_{\mathrm{pos}}(\alpha)$. Also, the name of the follow mapping gives the name of these automata.

The authors proved in [15,17] that

Theorem 10. (i) $\equiv_f$ *is a right-invariant equivalence w.r.t.* $\mathbf{A}_{\mathrm{pos}}(\alpha)$ *and*
(ii) $\mathbf{A}_{\mathrm{f}}(\alpha)\simeq\mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_f}$.

We notice that the restriction we imposed on $\equiv_f$ so that final and non-final states in $\mathsf{pos}_0(\alpha)$ cannot be $\equiv_f$-equivalent is essential, as shown by the expression $\alpha=(a^*b)^*$. Here $\mathsf{follow}(\alpha,i)=\{1,2\}$, for any $0\leqslant i\leqslant 2$. However, merging all three states of $\mathbf{A}_{\mathrm{pos}}(\alpha)$ is an error as the resulting automaton would accept the language $(a+b)^*$.

Example 11. We give an example of an application of Theorem 10. For the same regular expression $\tau=(a+b)(a^*+ba^*+b^*)^*$ from Example 2, we give in Fig. 6 the equivalence classes of $\equiv_f$ and the automaton $\mathbf{A}_{\mathrm{f}}(\tau)$.

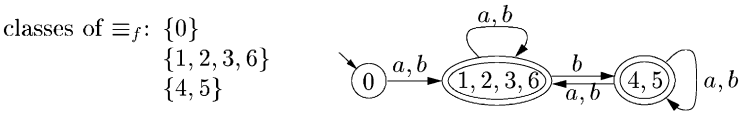

Fig. 6. $\mathbf{A}_{\mathrm{f}}^{\varepsilon}(\tau)$ and $\mathbf{A}_{\mathrm{f}}(\tau)=\mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_f}$ for $\tau=(a+b)(a^*+ba^*+b^*)^*$.

6. Basic idea for reducing arbitrary NFAs

We consider now the very general problem of constructing small NFAs when starting from arbitrary NFAs. Indeed, the results here can (and will) be applied to the problem of constructing small NFAs from regular expressions, simply by constructing first any NFA from a given regular expression and then reduce its size.

The basic idea comes from the algorithm for DFA minimization, where indistinguishable states are merged together, thus reducing the size of the automaton. Consider an automaton $M=(Q,A,\delta,q_0,F)$ and, for $q\in Q$, denote by M_q the automaton obtained from M by replacing the initial state q_0 with q. We say that two states p and q are *distinguishable* if there is a word w which can lead the automaton from p to a final state but cannot lead from q to a final state. Put otherwise, p and q are *indistinguishable* iff $L(M_p)=L(M_q)$. This is the place where the notion of right invariant equivalence comes into picture. If $\equiv$ is an equivalence on Q which is right invariant w.r.t. M, then $\equiv$-equivalence implies indistinguishability.

Lemma 12. *If $\equiv$ is a right-invariant equivalence, then $p\equiv q$ implies that p and q are indistinguishable.*

Proof. Take $w=a_1a_2\ldots a_n\in L(M_p)$. Then there is in M a path $p\xrightarrow{a_1}p_1\xrightarrow{a_2}p_2\xrightarrow{a_3}\cdots\xrightarrow{a_n}p_n$ such that $p_n\in F$. Since $p\equiv q$ and $\equiv$ is right-invariant, it follows inductively that there is in M also a path $q\xrightarrow{a_1}q_1\xrightarrow{a_2}q_2\xrightarrow{a_3}\cdots\xrightarrow{a_n}q_n$ such that $p_i\equiv q_i$, for all $1\leqslant i\leqslant n$. Now p_n is final implies that q_n must be final as well and hence $w\in L(M_q)$. The converse is proved analogously. □

This gives, in particular, a proof of the fact that the quotient automaton recognizes the same language as the initial one, which we already mentioned at the end of Section 2.

Corollary 13. *If $\equiv$ is right invariant, then $L(M/_{\equiv})=L(M)$.*

In the deterministic case, we start by distinguishing between final and non-final states. Then, two states that have a transition by the same letter to some distinguishable states become distinguishable. With non-determinism, we have sets of states and it is no longer clear how we should decide that two states are distinguishable. In fact, this seems very difficult to decide. What we shall do is find a superset, such that we are sure we do not merge state which we should not. There can always be states which could be merged but detecting those is too expensive.

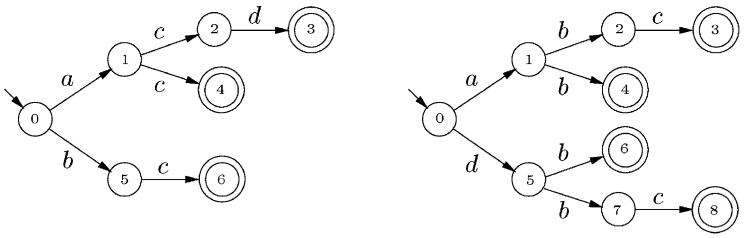

Fig. 7. (i) and (ii) are not good conditions.

Therefore, our goal is to compute an equivalence relation (denoted subsequently by $\equiv_R$) such that any two states $\equiv_R$-equivalent are indistinguishable but the converse need not be true. To avoid confusions, we shall call such states *equivalent*. So, equivalent states will be indistinguishable but non-equivalent states will not necessarily be distinguishable. As in the deterministic case, we compute the complement of this equivalence (denoted by $\not\equiv_R$); that is, we compute which states are not equivalent.

The starting point is that final states are not equivalent to non-final states; this is the only reasonable way to start with. Then, we discuss how to use current information about non-equivalent states in order to compute further non-equivalent pairs. Assume p and q are two states about which we do not know whether they are equivalent or not and a is a letter. (We shall complete the automaton such that both $\delta(p,a)$ and $\delta(q,a)$ are non-empty.) Two candidates for the condition implying non-equivalence of p and q come first to mind:

(i) any state in $\delta(p,a)$ is not equivalent to any state in $\delta(q,a)$,

(ii) there is a state in $\delta(p,a)$ and one in $\delta(q,a)$ which are not equivalent.

We see next why neither of (i) and (ii) is good. The condition in (i) is too strong as it turns out that we mark as non-equivalent too few pairs and then merge states which must not be merged. For example, in the first NFA in Fig. 7, states 1 and 5 will be merged and the resulting automaton accepts a strictly bigger language. On the other hand, condition (ii) is too weak and we obtain a too small reduction of the automaton. For instance, in the second NFA in Fig. 7, states 1 and 5 are not merged although they clearly should.

The right condition is in between (i) and (ii):

(iii) there is a state in $\delta(p,a)$ which is not equivalent to any state in $\delta(q,a)$.

We shall see in the next section that (iii) is precisely what we need to merge as many states as one can hope. We conclude this section with an example in Fig. 8 showing that finding indistinguishable states is hopeless.

We have in Fig. 8 that 3 and 8 are non-equivalent (as being final and non-final, resp.) and also 4 and 13 are non-equivalent. Therefore, no matter how we find further non-equivalent states, 2 and 7 will become non-equivalent, since $\delta(2,d)=\{3\}$, $\delta(7,d)=\{8\}$. Similarly, 2 and 11 will be non-equivalent. From this, it follows that 1 and 6 will be non-equivalent. But, clearly, 1 and 6 are indistinguishable.

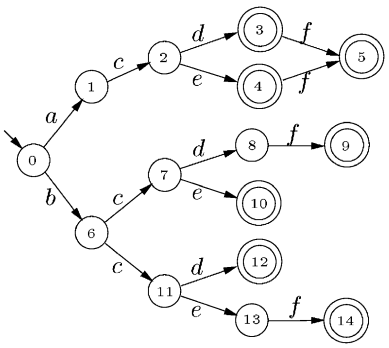

Fig. 8. (iii) is the best we can hope.

7. The largest right-invariant equivalence

Based on condition (iii) in the previous section, we give in this section an algorithm to compute the equivalent states. In fact, the algorithm below computes the complement of the equivalence relation, that is, the non-equivalent states.

Notice that we consider automata with arbitrarily many initial states. This more general setting will be useful for the dual approach in Section 10.

Algorithm 14. Given an NFA $M=(Q,A,\delta,I,F)$, the algorithm computes a relation $\equiv_R$ on states; $p\equiv_R q$ means p and q are equivalent and therefore can (and will) be merged.

1. complete the automaton M:
2. add a new (non-initial and non-final) state, denoted $\emptyset$, to Q;
3. the new set of states is $Q^{\emptyset}=Q\cup\{\emptyset\}$;
4. add all missing transitions—the new transition function is
5. $\delta^{\emptyset}=\delta\cup\{(p,a,\emptyset)\mid\delta(p,a)=\emptyset\}\cup\{(\emptyset,a,\emptyset)\mid a\in A\}$;
6. $\not\equiv_R\leftarrow\emptyset$;
7. **for** any $(p,q)\in(Q-F)\times F$ **do**
8. $\not\equiv_R\leftarrow\not\equiv_R\cup\{(p,q),(q,p)\}$;
9. **for** any $p\in Q$ **do**
10. $\not\equiv_R\leftarrow\not\equiv_R\cup\{(\emptyset,p),(p,\emptyset)\}$
11. **while** $\exists p,q\in Q^{\emptyset},\exists a\in A,\exists r\in\delta^{\emptyset}(p,a)\forall s\in\delta^{\emptyset}(q,a), r\not\equiv_R s$ **do**
12. choose one such pair (p,q);
13. $\not\equiv_R\leftarrow\not\equiv_R\cup\{(p,q),(q,p)\}$;
14. **return** $\equiv_R\ =\ \big((Q^{\emptyset}\times Q^{\emptyset})-\not\equiv_R\big)-\{(\emptyset,\emptyset)\}$.

Several comments are in order. As mentioned above, we compute the complement of the relation $\equiv_R$, that is, $\not\equiv_R$. Steps 7 and 8 impose the condition that final and non-final states will not be made equivalent. Condition (iii) appears in step 11.

Notice that steps 9 and 10 are not really needed. It is not difficult to see that the pairs which have exactly one component $\emptyset$ would be added anyway to $\not\equiv_R$. This is true based on the assumption that M has no useless states. Thus, for a state $q\in Q$,

there is a word w which leads from q to a final state but which leads from $\emptyset$ to $\emptyset$. Therefore, steps 9 and 10 are there just to make more clear that the state $\emptyset$ will finally be non-equivalent to any other state, as it should.

Notice also that, due to steps 9, 10, and 14, no pair in $\equiv_R$ contains the state $\emptyset$, so $\equiv_R$ is over Q.

We prove next some properties of the relation $\equiv_R$. In the proofs below, we shall need a precise view on the way pairs of states are added to $\not\equiv_R$. We therefore consider a topological order on $\not\equiv_R$, say

$$\not\equiv_R = \{(p_1,q_1),(p_2,q_2),\ldots,(p_t,q_t)\},$$

where $i<j$ means that (p_i,q_i) has been added (at steps 8, 10, or 13) to $\not\equiv_R$ before (p_j,q_j). The idea is that (p_i,q_i) was added without using the information that $(p_j,q_j)\in\not\equiv_R$ while (p_j,q_j) might have used the information that (p_i,q_i) was already in $\not\equiv_R$. We make also the distinction between the pairs added at steps 8 or 10 and the others. Put $1\leqslant r<s\leqslant t$ such that the pairs from 1 to r have been added at step 8, the ones from $r+1$ to s have been added at step 10, while the remaining ones have been added at step 13.

Lemma 15. *$\equiv_R$ is an equivalence relation over Q.*

Proof. $\equiv_R$ is reflexive because $\not\equiv_R$ is irreflexive and symmetric because $\not\equiv_R$ is symmetric. About transitivity, we prove by induction on the index of $(p,q)\in\not\equiv_R$ that if $p\not\equiv_R q$, then, for any t, either $t\not\equiv_R p$ or $t\not\equiv_R q$. This clearly implies transitivity. Put $(p,q)=(p_i,q_i)$, for some $1\leqslant i\leqslant t$. The assertion is clear if $1\leqslant i\leqslant s$. Assume that $i\geqslant s+1$ and the property is true for the pairs $1,2,\ldots,i-1$. By the definition of $\not\equiv_R$ and of the topological order, there is $a\in A$ and $r\in\delta^{\emptyset}(p,a)$ such that, for any $s\in\delta^{\emptyset}(q,a)$, the pair (r,s) belongs to $\not\equiv_R$ and has the index at most $i-1$. Assume $t\equiv_R p$ and $t\equiv_R q$. From $t\equiv_R p$ we get that there is $u\in\delta^{\emptyset}(t,a)$ such that $u\equiv_R r$. Now, $t\equiv_R q$ gives that there is $v\in\delta^{\emptyset}(q,a)$ such that $v\equiv_R u$. But then $(r,v)\in\not\equiv_R$ has index strictly smaller than i and u is $\equiv_R$-equivalent with both r and s, contradicting the inductive hypothesis. □

Lemma 16. *$\equiv_R$ is right invariant w.r.t. M.*

Proof. Assume p,q such that $p\equiv_R q$ but $\delta^{\emptyset}(p,a)/_{\equiv_R}\neq\delta^{\emptyset}(q,a)/_{\equiv_R}$. Take $[r]_{\equiv_R}\in\delta^{\emptyset}(p,a)/_{\equiv_R}-\delta^{\emptyset}(q,a)/_{\equiv_R}$ (or symmetrically). Then, for any $s\in\delta^{\emptyset}(q,a)$, we have $r\not\equiv_R s$, and hence (p,q) must have been added to $\not\equiv_R$ at step 13, a contradiction. □

The statement of Lemma 16 follows easily because condition 11 in Algorithm 14 was actually chosen to build a right-invariant equivalence. Moreover, it builds the largest such equivalence, as seen in Theorem 17.

Theorem 17. *$\equiv_R$ is the largest equivalence over Q which is right invariant w.r.t. M.*

Proof. Consider an arbitrary equivalence $\equiv$ over Q which is right invariant w.r.t. M. Assume $\equiv\not\subseteq\equiv_R$ and take $(p,q)\in\equiv-\equiv_R$. Thus, $p\not\equiv_R q$ and assume (p,q) has the

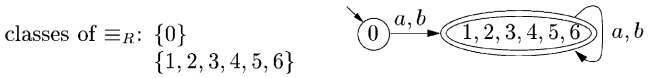

Fig. 9. $\mathbf{A}_{\mathrm{R}}(\tau) = \mathbf{A}_{\mathrm{pos}}(\tau)/_{\equiv_R}$ for $\tau = (a+b)(a^* + ba^* + b^*)^*$.

lowest possible index in $\not\equiv_R$; let it be i. Because any class of $\equiv$ does not contain both final and nonfinal states, we have $i \geqslant r+1$. Thus, there are $a \in A$ and $r \in \delta^{\emptyset}(p,a)$ such that, for any $s \in \delta^{\emptyset}(q,a)$, $r \not\equiv_R s$ and also any such (r,s) has lower index than (p,q). Therefore, for all such (r,s), we must have $r \not\equiv s$ which contradicts the right invariance of $\equiv$. □

The automaton $M/_{\equiv_R}$ is the one we look for. It follows from Lemma 16 and Corollary 13 that it recognizes the same language as M does. Also, by Theorem 17, no other right-invariant equivalence can give a smaller automaton.

Corollary 18. *Among all quotient automata $M/_{\equiv}$, where $\equiv$ is an equivalence on Q which is right invariant w.r.t. M, the automaton $M/_{\equiv_R}$ is the smallest.*

8. Small NFAs from regular expressions

We apply now the construction in the previous section to NFAs obtained from regular expressions. Consider a regular expression α. We have three equivalences on $\mathsf{pos}_0(\alpha)$ which are right invariant w.r.t. $\mathbf{A}_{\mathrm{pos}}(\alpha)$: $\equiv_c$ from Section 4, $\equiv_f$ from Section 5, and $\equiv_R$ from Section 7. Denote $\mathbf{A}_{\mathrm{R}}(\alpha) = \mathbf{A}_{\mathrm{pos}}(\alpha)/_{\equiv_R}$. The following result is a corollary of Theorems 17, 5, and 10.

Theorem 19. *For any regular expression α, $\mathbf{A}_{\mathrm{R}}(\alpha)$ is an NFA which accepts the language $L(\alpha)$ and is always smaller than or equal to either of $\mathbf{A}_{\mathrm{f}}(\alpha)$ and $\mathbf{A}_{\mathrm{pd}}(\alpha)$.*

As seen in Example 22, $\mathbf{A}_{\mathrm{R}}(\alpha)$ can be arbitrarily smaller! Notice also that we need not compute $\mathbf{A}_{\mathrm{pos}}(\alpha)$ first; $\mathbf{A}_{\mathrm{R}}(\alpha)$ can be obtained from either of $\mathbf{A}_{\mathrm{f}}(\alpha)$ or $\mathbf{A}_{\mathrm{pd}}(\alpha)$.

Example 20. We show here the automaton $\mathbf{A}_{\mathrm{R}}(\tau)$ for the same expression $\tau = (a+b)(a^* + ba^* + b^*)^*$ from Example 2. We see that the automaton $\mathbf{A}_{\mathrm{pos}}(\tau)$ is complete, see Fig. 2, thus the state $\emptyset$ will be completely separated from all the others and will not influence the relation $\not\equiv_R$ in any way. On the other hand, the only pairs not containing $\emptyset$ in $\not\equiv_R$ are of the form $(0,i)$ or $(i,0)$, for $1 \leqslant i \leqslant 6$. Since the state 0 has no incoming transitions, these pairs cannot help bring other pairs inside $\not\equiv_R$. Therefore, $\not\equiv_R$ will be able only to distinguish between final and non-final states (added at step 8 in Algorithm 14). The classes of $\equiv_R$ are shown in Fig. 9 which contains also the corresponding automaton. We can see it is strictly smaller than either of $\mathbf{A}_{\mathrm{pd}}(\tau)$ and $\mathbf{A}_{\mathrm{f}}(\tau)$; compare to Figs. 3 and 6.

9. Comparing A_R to other constructions

We give in this section several examples to compare the automaton $\mathbf{A}_R$ with the other constructions mentioned above. It is shown that it can be arbitrarily smaller.

Example 21. Consider the expression $\alpha_n = a_1^*(a_1 + a_2)^* \cdots (a_1 + a_2 + \cdots + a_n)^*$. We have $|\alpha_n| = n^2 + 2n - 1$ and

- $|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = \Theta(|\alpha_n|^2)$,
- $|\mathbf{A}_{\mathrm{f}}(\alpha_n)| = |\mathbf{A}_{\mathrm{pd}}(\alpha_n)| = \Theta(|\alpha_n|)$, and
- $|\mathbf{A}_{\mathrm{R}}(\alpha_n)| = \Theta(|\alpha_n|)$.

Although the order is the same, $\mathbf{A}_{\mathrm{R}}(\alpha_n)$ is much smaller than either of $\mathbf{A}_{\mathrm{f}}(\alpha_n)$ and $\mathbf{A}_{\mathrm{pd}}(\alpha_n)$. It has in fact only one state and the size comes from the fact that we have an unbounded alphabet. In the next example we work with a fixed alphabet and get a big gap.

Example 22. For $\alpha_n = (a + b + \varepsilon)(a + b + \varepsilon) \cdots (a + b + \varepsilon)(a + b)^*$, where $(a + b + \varepsilon)$ is repeated n times, we have $|\alpha_n| = 6n + 4$ and

- $|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = \Theta(|\alpha_n|^2)$,
- $|\mathbf{A}_{\mathrm{f}}(\alpha_n)| = |\mathbf{A}_{\mathrm{pd}}(\alpha_n)| = \Theta(|\alpha_n|^2)$, and
- $|\mathbf{A}_{\mathrm{R}}(\alpha_n)| = 3$.

So, $\mathbf{A}_{\mathrm{R}}(\alpha_n)$ has a fixed size (independent of n), and hence can be arbitrarily smaller than the others.

10. Dual reduction to the left

When reducing NFAs according to $\equiv_R$, we considered only the distinguishability of the states to the right, that is, for two states p and q, we considered the words that led from p or q to final states. It is interesting to remark that the same thing can be done symmetrically to the left, considering words that lead from the initial state(s) to p or q. We shall not redo the whole thing but simply use what we have so far. Consider an NFA, $M = (Q, A, \delta, I, F)$, and construct the *reversed* automaton $M^R = (Q, A, \delta^R, F, I)$, where, for any $p, q \in Q$ and $a \in A$, $(p, a, q) \in \delta^R$ iff $(q, a, p) \in \delta$. So, we simply exchange the sets of initial and final states and reverse the direction of all transitions (keeping the labels unchanged). Now, we can compute as before the equivalence $\equiv_R$ corresponding to M^R; let us denote it by $\equiv_L$. The notion of left-invariant equivalence is then defined similar to the right-invariant equivalence. A relation $\equiv$ is *left invariant* w.r.t. M iff

(i) $\equiv \subseteq (Q - \{q_0\})^2 \cup \{q_0\}^2$ (the initial state, final in M^R, is not $\equiv$-equivalent to any other state),

(ii) for any $p, q \in Q$, $a \in A$, if $p \equiv q$, then $\{r \in Q \mid p \in \delta(r, a)\}/_{\equiv} = \{r \in Q \mid q \in \delta(r, a)\}/_{\equiv}$ (or, equivalently, $\delta^R(p, a)/_{\equiv} = \delta^R(q, a)/_{\equiv}$; for equivalent states p and q and a given letter a, we can move to p and q, resp., from the same equivalence classes).

We can reduce an automaton using a left-invariant equivalence in the same way as we did for a right-invariant one. This because

$$L(M/_{\equiv_L}) = L((M/_{\equiv_L})^R)^R = L(M^R/_{\equiv_L})^R = (L(M)^R)^R = L(M).$$

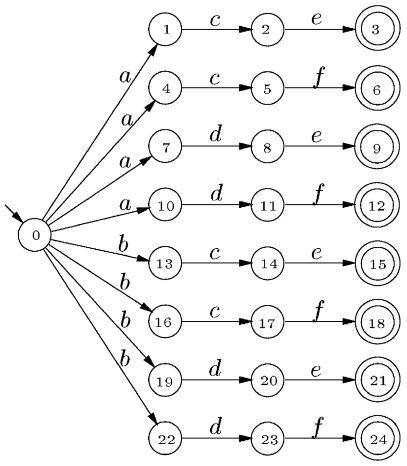

Fig. 10. $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$.

The following results can be proved as above.

Theorem 23. *$\equiv_L$ is the largest equivalence over Q which is left invariant w.r.t. M.*

Corollary 24. *Among all quotient automata $M/_{\equiv}$, where $\equiv$ is an equivalence on Q which is left invariant w.r.t. M, the automaton $M/_{\equiv_L}$ is the smallest.*

11. Reducing NFAs in both directions

We have seen so far that both $\equiv_R$ and $\equiv_L$ are powerful tools in reducing the size of automata. A natural problem occurs: Is it possible to combine the two for even more powerful reduction? The answer is positive, as we shall see in this section. However, how the two should be combined for best results is not at all obvious. We show that there are examples where a unique way to combine the two does not exist.

Consider first the regular expression $\alpha_n = \bigoplus_{1 \leqslant i_j \leqslant 2} a_{1i_1} a_{2i_2} \cdots a_{ni_n}$ (this is $\bigodot_{i=1}^{n} (a_{i1} + a_{i2})$ after opening the parentheses—using distributivity of catenation over union). We have

- $|\alpha_n| = 2n2^n - 1$,
- $|\mathbf{A}_{\mathrm{pos}}(\alpha_n)| = 2n2^n + 1 = \Theta(|\alpha_n|)$,
- $|\mathbf{A}_{\mathrm{R}}(\alpha_n)| = 3 \cdot 2^n - 2 = \Theta(|\alpha_n|/\log|\alpha_n|)$,
- $|\mathbf{A}_{\mathrm{L}}(\alpha_n)| = 4 \cdot 2^n - 3 = \Theta(|\alpha_n|/\log|\alpha_n|)$.

Interestingly, the two equivalences can be used to obtain an automaton of size $3n + 1 = \Theta(\log|\alpha_n|)$, that is, exponentially smaller. The two are used in sequence and the order, incidentally, does not matter.

We see here in detail the case $n = 3$. For clarity, we rename the letters in α_3 and put $\alpha_3 = ace + acf + ade + adf + bce + bcf + bde + bdf$. The position automaton $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$ is shown in Fig. 10; the mappings first, last, and follow are clear from the figure.

classes of $\equiv_R$:
$\{0\}$
$\{1,13\}$
$\{7,19\}$
$\{4,16\}$
$\{10,22\}$
$\{2,8,14,20\}$
$\{5,11,17,23\}$
$\{3,6,9,12,15,18,21,24\}$

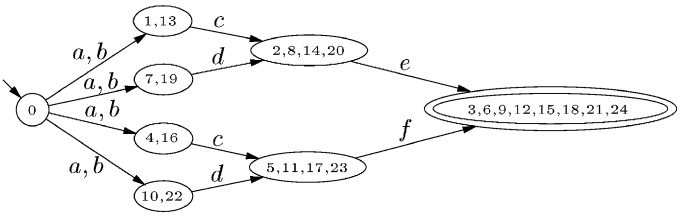

Fig. 11. $\mathbf{A}_{\mathrm{R}}(\alpha_3)$.

classes of $\equiv_L$:
$\{0\}$
$\{1,4,7,10\}$
$\{13,16,19,22\}$
$\{2,5\}$
$\{8,11\}$
$\{14,17\}$
$\{20,23\}$
$\{3\}$
$\{6\}$
$\{9\}$
$\{12\}$
$\{15\}$
$\{18\}$
$\{21\}$
$\{24\}$

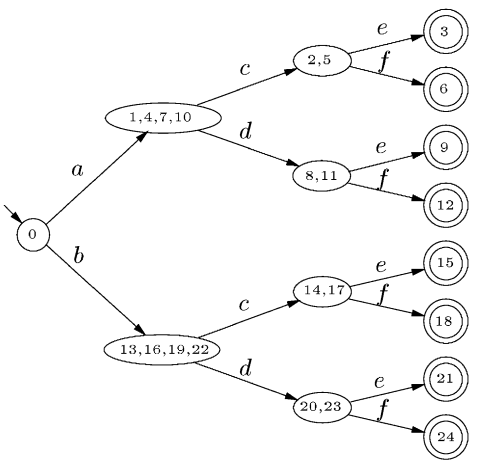

Fig. 12. $\mathbf{A}_{\mathrm{L}}(\alpha_3)$.

Next, we show in Fig. 11 the equivalence classes of $\equiv_R$ and the automaton $\mathbf{A}_{\mathrm{R}}(\alpha_3)$. First, there is no way to make inequivalent any two final states since they have all transitions leading to the state $\emptyset$ (not shown). Next, all states leading by a transition labeled the same (e or f) to a final state cannot be made inequivalent and this will give the two four-element classes of $\equiv_R$. The four two-element classes are obtained similarly.

The equivalence classes of $\equiv_L$ and the corresponding automaton are shown in Fig. 12. It is obtained following a reasoning similar to the one for $\equiv_R$.

Finally, the two equivalences $\equiv_R$ and $\equiv_L$ can be used to construct a much smaller automaton to accept the same language. This is shown in Fig. 13. This automaton is obtained as follows. We use first $\equiv_R$ in $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$ and then $\equiv_L$ in $\mathbf{A}_{\mathrm{R}}(\alpha_3) = \mathbf{A}_{\mathrm{pos}}(\alpha_3)/_{\equiv_R}$ (here $\equiv_L$ is the one corresponding to $\mathbf{A}_{\mathrm{R}}(\alpha_3)$ and not the one of $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$). The same is obtained as $(\mathbf{A}_{\mathrm{pos}}(\alpha_3)/_{\equiv_L})/_{\equiv_R}$, where each equivalence refers here to the automaton the quotient of which it is producing. Notice that the automaton in Fig. 13 could not be obtained by using the equivalences $\equiv_R$ and $\equiv_L$ of $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$ (at least not immediately)

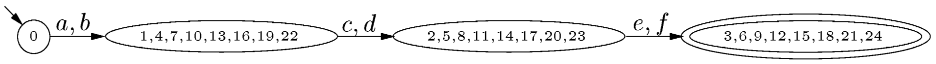

Fig. 13. $\mathbf{A}_{\mathrm{pos}}(\alpha_3)$ reduced both ways.

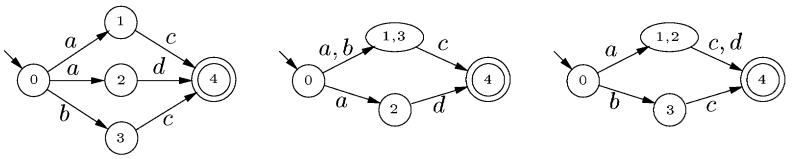

Fig. 14. An NFA and its corresponding quotients modulo $\equiv_R$ and $\equiv_L$.

since, for instance, the states 1 and 16 are not equivalent w.r.t. either one but still together in the same class in Fig. 13.

We conclude this section by an example when there is no unique way to reduce optimally an automaton using $\equiv_R$ and $\equiv_L$. Consider the first automaton in Fig. 14. The only non-trivial pair $\equiv_R$-equivalent is $(1,3)$ and the only non-trivial $\equiv_L$-equivalent pair is $(1,2)$. Since 1, 2, and 3 cannot be merged together (the language accepted would become $ac + ad + bc + bd$), $\equiv_R$ and $\equiv_L$ can be used only independently; each will reduce the size by the same amount; the corresponding quotient automata w.r.t. $\equiv_R$ and $\equiv_L$ are shown as the second and third automata, resp., in Fig. 14.

12. Conclusions and further research

We gave a general method to reduce the size of arbitrary NFAs, which is based on construction of invariant equivalences on the set of states such that equivalent states can be merged together. When applying the construction to regular expressions, we obtain NFAs which can be arbitrarily smaller (!) than position, partial derivative, or follow automata. Although the best worst case is still the one given by Hromkovic et al. [14], our construction seems to perform better on most examples. The worst case seems to be irrelevant (see also the discussion in [17]). We have not compared our automata with those of Chang and Paige [8] since we do not work with compressed automata.

Notice also that comparing two constructions of automata is very difficult. The case of quotients is a happy one since we know that the quotient always gives a better construction, though not necessarily strictly better.

Several important problems remain to be investigated further. First, even though the Algorithm 14 runs in low polynomial time, further work should be done to improve its running time. Again, the running time in the worst case seems to be far from the expected one.

Second, we have seen that using $\equiv_R$ and $\equiv_L$ together can result in much smaller NFAs. Further research should investigate how the two can be combined in an optimal way, which, according to the example in the previous section, is in general not unique.

Third, if combining optimally $\equiv_R$ and $\equiv_L$ turns out to be hard, then a trade-off will be needed such that they help reducing the size but in a way which can be performed efficiently.

References

[1] A. Aho, R. Sethi, J. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, Reading, MA, 1988.
[2] V. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, Theoret. Comput. Sci. 155 (1996) 291–319.
[3] G. Berry, R. Sethi, From regular expressions to deterministic automata, Theoret. Comput. Sci. 48 (1986) 117–126.
[4] A. Brüggemann-Klein, Regular expressions into finite automata, Theoret. Comput. Sci. 120 (1993) 197–213.
[5] J. Brzozowski, Derivatives of regular expressions, J. ACM 11 (1964) 481–494.
[6] J.-M. Champarnaud, D. Ziadi, New finite automaton constructions based on canonical derivatives, in: S. Yu, A. Paun (Eds.), Proc. CIAA 2000, Lecture Notes in Computer Science, Vol. 2088, Springer, Berlin, 2001, pp. 94–104.
[7] J.-M. Champarnaud, D. Ziadi, Computing the equation automaton of a regular expression in $\mathcal{O}(s^2)$ space and time, in: A. Amir, G. Landau (Eds.), Proc. 12th CPM, Lecture Notes in Computer Science, Vol. 2089, Springer, Berlin, 2001, pp. 157–168.
[8] C.-H. Chang, R. Paige, From regular expressions to DFA's using compressed NFA's, Theoret. Comput. Sci. 178 (1997) 1–36.
[9] J. Friedl, Mastering Regular Expressions, O'Reilly, Sebastopol, CA, 1998.
[10] V.M. Glushkov, The abstract theory of automata, Russian Math. Surveys 16 (1961) 1–53.
[11] C. Hagenah, A. Muscholl, Computing ε-free NFA from regular expressions in $O(n \log 2(n))$ time, Theor. Inform. Appl. 34 (4) (2000) 257–277.
[12] J. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, Proc. Internat. Symp. Theory of Machines and Computations, Technion, Haifa, Academic Press, New York, 1971, pp. 189–196.
[13] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[14] J. Hromkovic, S. Seibert, T. Wilke, Translating regular expressions into small ε-free non-deterministic finite automata, J. Comput. System Sci. 62 (4) (2001) 565–588.
[15] L. Ilie, S. Yu, Constructing NFAs by optimal use of positions in regular expressions, in: A. Apostolico, M. Takeda (Eds.), Proc. 13th CPM, Fukuoka, 2002, Lecture Notes in Computer Science, Springer, Berlin, 2002, pp. 279–288.
[16] L. Ilie, S. Yu, Algorithms for computing small NFAs, in: K. Diks, W. Rytter (Eds.), Proc. 27th MFCS, Warszawa, 2002, Lecture Notes in Computer Science, Vol. 2420, Springer, Berlin, 2002, pp. 328–340.
[17] L. Ilie, S. Yu, Follow automata, Inform. Comput., to appear.
[18] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IEEE Trans. Electron. Comput. 9 (1) (1960) 39–47.
[19] S. Sippu, E. Soisalon-Soininen, Parsing Theory: I Languages and Parsing, in: EATCS Monographs on Theoretical Computer Science, Vol. 15, Springer, New York, 1988.
[20] K. Thompson, Regular expression search algorithm, Comm. ACM 11 (6) (1968) 419–422.
[21] S. Yu, Regular Languages in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. I, Springer, Berlin, 1997, pp. 41–110.