Left is Better than Right for Reducing Nondeterminism of NFAs

Sang-Ki Ko and Yo-Sub Han

Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Korea
{narame7,emmous}@cs.yonsei.ac.kr

Abstract. We study the NFA reductions by invariant equivalences. It is well-known that the NFA minimization problem is PSPACE-complete. Therefore, there have been approaches to reduce the size of NFAs in low polynomial time by computing invariant equivalence and merging the states within same equivalence class. Here we consider the nondeterminism reduction of NFAs by invariant equivalences. We, in particular, show that the left-invariant equivalence is more useful than the right-invariant equivalence for reducing NFA nondeterminism. We also present experimental evidence for showing that NFA reduction by left-invariant equivalence achieves the better reduction of nondeterminism than right-invariant equivalence.

Keywords: Nondeterministic finite automata, Regular expression, NFA reduction, Invariant equivalences.

1 Introduction

Regular expressions are widely used for many applications such as search engine, text editor, programming language, and so on. People often use regular expressions to describe a set of pattern strings for the pattern matching problem.

Once a regular expression is given, then we convert a regular expression into an equivalent nondeterministic finite-state automaton (NFA) by automata constructions such as Thompson construction [21] or the position construction[1] [6,17]. In some cases, the obtained NFA should be converted into a deterministic one by the subset construction. However, the size of the deterministic finite-state automaton (DFA) for the regular expression may be exponential. In addition to that, the problem of minimizing NFAs is PSPACE-complete [14], thus, intractable.

Since DFAs are usually much faster than NFAs, the most of applications prefer DFAs to NFAs. For example, consider the *membership problem* which is the simplest form of pattern matching problem based on FAs. Given an FA of size m and a string of length n, the problem requires $O(n)$ time if the FA is deterministic whereas it takes $O(m^2n)$ time [7,22] in the worst-case if the FA is nondeterministic.

[1] Also known as Glushkov construction.

M. Holzer and M. Kutrib (Eds.): CIAA 2014, LNCS 8587, pp. 238–251, 2014.

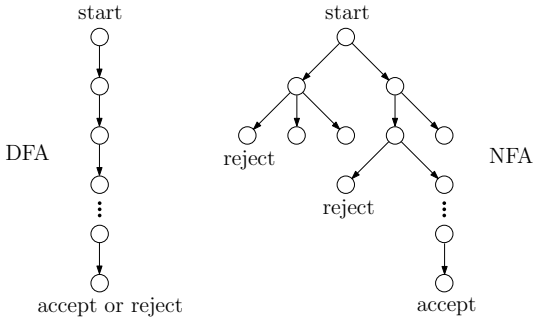

Fig. 1. Difference between deterministic and nondeterministic computations

The real problem is, it is impossible to have small DFAs as NFAs for the same regular languages. It is well known that exponential number of states may be required for an NFA to be represented by a DFA. As an alternative solution, there have been many approaches on NFA reduction techniques for the space-efficient implementations of the applications using regular expressions.

The idea of reducing the size of NFAs by equivalence relations was first proposed by Ilie and Yu [11]. Champarnaud and Coulon [5] modified the idea to use preorders over the set of states instead of equivalences for the better reduction. Later, Ilie et al. [9] showed that it is possible to reduce the size of an NFA with n states and m transitions in $O(m \log n)$ time by equivalences and $O(mn)$ time by preorders. Ilie et al. [10] also showed that the optimal use of equivalences can be computed in polynomial time and the optimal use of preorders is NP-hard.

Here we consider the problem of reducing the nondeterminism of NFAs by using invariant equivalences because the nondeterminism is also a very important factor for the efficient simulation of NFAs. We define the *computation graph* for estimating the nondeterminism of NFAs and investigate several properties. Then, we compare the right- and left-invariant equivalences by reducing NFAs by the equivalences and give experimental results with uniformly generated random regular expressions.

The paper is organized as follows. In Section 2, we shall give some definitions and notations. We introduce the well-known construction of the position automaton from a regular expression in Section 3. We present NFA reduction by invariant equivalences in Section 4 and consider the nondeterminism of NFAs in Section 5. The experimental results are given in Section 6. Section 7 concludes the paper.

2 Preliminaries

Here we briefly recall the basic definitions used throughout the paper. For complete background knowledge in automata theory, the reader may refer to textbooks [7,22].

Let Σ be a finite alphabet and Σ^* be the set of all strings over the alphabet Σ including the empty string λ. The size $|\Sigma|$ of Σ is the number of characters in Σ. For a string $w \in \Sigma^*$, we denote the length of w by $|w|$ and the ith character of w by w_i. A language over Σ is any subset of Σ^*. A *regular expression* over Σ is $\emptyset, \lambda$, or $a \in \Sigma$, or is obtained by applying the following rules finitely many times. For two regular expressions $\mathcal{R}_1$ and $\mathcal{R}_2$, the union $\mathcal{R}_1 + \mathcal{R}_2$, the concatenation $\mathcal{R}_1 \cdot \mathcal{R}_2$, and the star $\mathcal{R}_1^*$ are regular expressions. For a regular expression $\mathcal{R}$, the language represented by $\mathcal{R}$ is denoted by $\mathcal{L}(\mathcal{R})$. The size $|\mathcal{R}|$ of a regular expression $\mathcal{R}$ implies the number of symbols including the characters from Σ and syntactic symbols such as $+, \cdot$, and $*$. We denote the number of occurrences of characters from Σ in $\mathcal{R}$ by $|\mathcal{R}|_\Sigma$.

A *nondeterministic finite-state automaton* (NFA) $\mathcal{A}$ is specified by a 5-tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a multi-valued transition function, $s \in Q$ is the initial state and $F \subseteq Q$ is a set of final states.

For a transition $q \in \delta(p, a)$ in $\mathcal{A}$, we say that p has an *out-transition* and q has an *in-transition*. Furthermore, p is a *source state* of q and q is a *target state* of p. The transition function δ can be extended to a function $Q \times \Sigma^* \to 2^Q$ that reflects sequences of inputs. A string w over Σ is accepted by $\mathcal{A}$ if there is a labeled path from s to a state in F such that this path spells out the string w, namely, $\delta(s, w) \cap F \neq \emptyset$. The language $\mathcal{L}(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of all strings that are spelled out by paths from s to a final state in F. Formally we write

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(s, w) \cap F \neq \emptyset\}.$$

For a state $q \in Q$, we denote

$$\mathcal{L}_L(A, q) = \{w \in \Sigma^* \mid q \in \delta(s, w)\}, \quad \mathcal{L}_R(A, q) = \{w \in \Sigma^* \mid \delta(q, w) \cap F \neq \emptyset\};$$

when A is understood from the context, we simply write $\mathcal{L}_L(q), \mathcal{L}_R(q)$, respectively.

For a state $q \in Q$ and a string $w \in \Sigma^*$, the *q-computation tree* $T_{\mathcal{A},q,w}$ of $\mathcal{A}$ on w is a labeled tree where the nodes are labeled by elements of $Q \times (\Sigma \cup \{\lambda, \natural\})$, where $\natural \notin \Sigma$. Note that $T_{\mathcal{A},q,\lambda}$ is a single-node tree labeled by (q, λ). Assume that $w = au$, where $a \in \Sigma$, $u \in \Sigma^*$, and $\delta(q, a) = \emptyset$. Then, $T_{\mathcal{A},q,w}$ is again a single-node tree where the only node is labeled by $(q, \natural)$. If $\delta(q, a) = \{p_1, \ldots, p_m\}$, where $m \geq 1$, then $T_{\mathcal{A},q,w}$ is the tree with the root node labeled by (q, a) and the root node has m children where the subtree rooted at the ith child is $T_{\mathcal{A},p_i,u}$ for $i = 1, \ldots, m$. We call the tree $T_{\mathcal{A},s,w}$ the *computation tree* of $\mathcal{A}$ on w and simply denote $T_{\mathcal{A},w}$. If there is an accepting computation for w in the NFA $\mathcal{A}$, $T_{\mathcal{A},w}$ has a leaf labeled by (q, λ), where $q \in F$.

We also define the *computation graph* $G_{\mathcal{A},w}$ of $\mathcal{A}$ on w by merging equivalent subtrees of the computation tree as a single subtree. If $T_{\mathcal{A},w}$ has two computation trees $T_{\mathcal{A},q,v}$ as subtrees, where $w = uv$, $w, u, v \in \Sigma^*$, and $q \in Q$, then we merge the trees into one.

We denote the number of nodes and the number of edges of a computation tree $T_{\mathcal{A},w}$ by $|T_{\mathcal{A},w}|_N$ and $|T_{\mathcal{A},w}|_E$, respectively. We define the size $|T_{\mathcal{A},w}|$ of a

computation tree $T_{\mathcal{A},w}$ to be $|T_{\mathcal{A},w}|_N + |T_{\mathcal{A},w}|_E$. Note that the similar notations are defined analogously for the size of computation graph.

3 NFA Constructions from Regular Expressions

We first recall the well-known construction called the *position construction* for obtaining NFAs from regular expressions [6,17]. The automaton obtained from the construction is called the *position automaton* which is also called the *Glushkov automaton*.

Given a regular expression $\mathcal{R}$, we first mark each character of $\mathcal{R}$ with a unique index called the position. From the leftmost character of $\mathcal{R}$, we mark the index of each character with the number from 1 to $|\mathcal{R}|_\Sigma$. The set of indices is called the *positions* of $\mathcal{R}$ and denoted by $\mathsf{pos}(\mathcal{R}) = \{1, 2, \ldots, |\mathcal{R}|_\Sigma\}$. We also denote $\mathsf{pos}_0(\mathcal{R}) = \mathsf{pos}(\mathcal{R}) \cup \{0\}$. We denote the marked regular expression obtained from $\mathcal{R}$ by $\overline{\mathcal{R}}$. Note that $\mathcal{L}(\overline{\mathcal{R}}) \subseteq \overline{A}^*$, where $\overline{A} = \{a_i \mid a \in \Sigma, 1 \leq i \leq |\mathcal{R}|_\Sigma\}$. For instance, if $\mathcal{R} = abc + d(ef)^*$, then $\overline{\mathcal{R}} = a_1b_2c_3 + d_4(e_5f_6)^*$. For $a \in \Sigma$, $a = \overline{\overline{a}}$.

For a regular expression $\mathcal{R}$, we define first, last, and follow as follows:

$$\mathsf{first}(\mathcal{R}) = \{i \mid a_i w \in L(\overline{\mathcal{R}})\},$$

$$\mathsf{last}(\mathcal{R}) = \{i \mid w a_i \in L(\overline{\mathcal{R}})\},$$

$$\mathsf{follow}(\mathcal{R}, i) = \{j \mid u a_i a_j v \in L(\overline{\mathcal{R}})\}.$$

We extend $\mathsf{follow}(\mathcal{R}, 0) = \mathsf{first}(\mathcal{R})$ and define $\mathsf{last}_0(\mathcal{R})$ to be $\mathsf{last}(\mathcal{R})$ if $\lambda \in L(\mathcal{R})$ and $\mathsf{last}(\mathcal{R}) \cup \{0\}$ otherwise.

Then, the position automaton of $\mathcal{R}$ is defined as follows:

$$\mathcal{A}_{\mathsf{pos}}(\mathcal{R}) = (\mathsf{pos}_0(\mathcal{R}), \Sigma, \delta_{\mathsf{pos}}, 0, \mathsf{last}_0(\mathcal{R})),$$

where

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

Notice that the position automaton of $\mathcal{R}$ recognizes the same language with the regular expression $\mathcal{R}$, that is, $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{A}_{\mathsf{pos}}(\mathcal{R}))$.

The position automaton has two useful properties as follows.

Property 1. The position automaton for the regular expression $\mathcal{R}$ has always $|\mathcal{R}|_\Sigma + 1$ states.

Proof. Recall that every position automaton satisfies Property 2. The NFA $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})/_{\equiv_L}$ is obtained from $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})$ by merging states if they are in the same left-invariant equivalence class. In other words, any two states q and p are merged if $p \equiv_L q$, thus, $\mathcal{L}_L(p) = \mathcal{L}_L(q)$. This implies that p and q should have in-transitions consuming the same character because otherwise $\mathcal{L}_L(p) \neq \mathcal{L}_L(q)$. Therefore, the merged state by the left-invariant equivalence $\equiv_L$ also has in-transitions labeled by the same character. □

Property 1 guarantees that the position automaton always has smaller number of states than Thompson's automaton [21].

Property 2. All in-transitions for any state of the position automaton are labeled by the same character.

Caron and Ziadi [4] named the second property as the *homogeneous* property. We say that an FA is homogeneous if all in-transitions to a state have the same label. The homogeneous property helps to improve the regular expression search algorithms because we can represent the DFA using $O(2^{|\mathcal{R}|_\Sigma} + |\Sigma|)$ bit-masks of length $|\mathcal{R}_\Sigma|$ instead of $O(2^{|\mathcal{R}|_\Sigma} \cdot |\Sigma|)$ [18,19]. We can compute the position automaton in quadratic time in the size of regular expression using inductive definition of first, last, and follow [3].

Note that there have been proposed two more algorithms for obtaining smaller NFAs than position automata from regular expressions. Antimirov [2] introduced an NFA construction based on partial derivatives called the *partial derivate automaton*. Ilie and Yu [12] constructed an NFA called the *follow automaton* based on the follow relation. It is already proven that a partial derivative automaton and a follow automaton are quotients of the position automaton and always smaller than the position automaton.

4 NFA Reduction by Invariant Equivalences

There have been many results for reducing the size of NFAs by using invariant equivalences [9,11,13]. Here we briefly recall how the reduction works.

Basically, the idea of NFA reduction is from DFA minimization in the sense that we find indistinguishable states and merge them to reduce the size of DFAs. Let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be an NFA. For any two states p and q of $\mathcal{A}$, we say that p and q are *distinguishable* if there exists a string w such that $\delta(q, w) \cap F \neq \emptyset$ and $\delta(p, w) \cap F = \emptyset$. Naturally, this leads to the fact that p and q are *indistinguishable* if and only if $\mathcal{L}_R(p) = \mathcal{L}_R(q)$. If $\equiv$ is an equivalence on Q which is right-invariant with respect to $\mathcal{A}$, then $p \equiv q$ implies that p and q are indistinguishable.

The largest right-invariant equivalence relation $\equiv_R$ over Q should satisfy the following properties:

(i) $\equiv_R \cap (F \times (Q - F)) = \emptyset$,
(ii) for any $p, q \in Q, a \in \Sigma$, $p \equiv_R q$ if for all $q' \in \delta(q, a)$, there exists $p' \in \delta(p, a)$ such that $q' \equiv_R p'$.

After computing $\equiv_R$, we can reduce the NFA $\mathcal{A}$ by simply merging all states in the same equivalence class. Given an equivalence $\equiv$ and an NFA $\mathcal{A}$, we denote the NFA obtained after merging the equivalent states by $\mathcal{A}/_\equiv$. For any regular expression $\mathcal{R}$, $\mathcal{A}(\mathcal{R})/_{\equiv_R}$ is always smaller than the partial derivative automaton and the follow automaton since $\equiv_R$ is the largest one among all the right-invariant equivalence relations.

Note that the largest left-invariant equivalence relation $\equiv_L$ can be computed by reversing the given NFA and computing the largest right-invariant equivalence

of it. Although the partial derivative automaton [2] and the follow automaton [12] can be obtained by the right-invariant equivalence relations, the left-invariant equivalence has some nice properties.

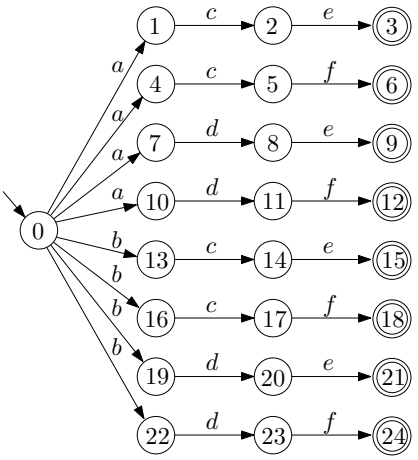

Fig. 2. A position automaton $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})$ where $\mathcal{R} = ace + acf + ade + adf + bce + bcf + bde + bdf$

See the position automaton $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})$ in Fig. 2 as an inspiring example. Note that this example is already used in the paper by Ilie and Yu [13]. Fig. 3 is an NFA reduced by $\equiv_R$ and Fig. 4 is an NFA reduced by $\equiv_L$.

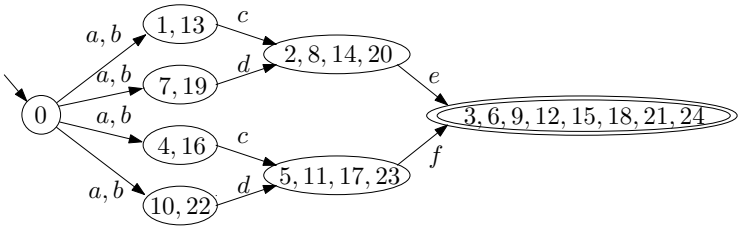

Fig. 3. An NFA $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})_{\equiv_R}$

While the number of states is smaller when reduced by $\equiv_R$ than $\equiv_L$, the NFA reduced by $\equiv_L$ is deterministic. Note that the NFA reduction by $\equiv_L$ does not always produce DFAs. However, this example implies that the left-invariant equivalence is useful for reducing nondeterminism of NFAs since the equivalence relation is computed in the same direction as the simulation of NFAs.

We also mention that the NFA reduction by the left-invariant equivalence preserves the homogeneous property which is very useful for the regular expression search algorithms.

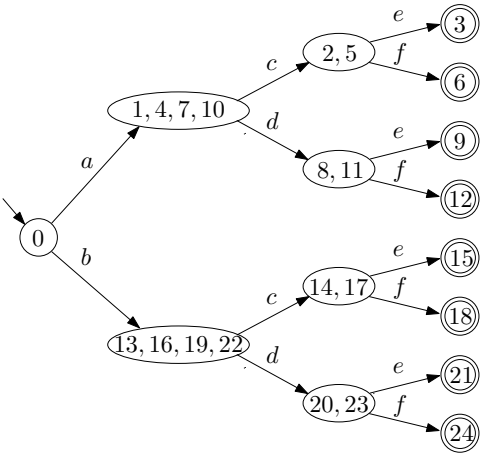

Fig. 4. An NFA $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})_{\equiv_L}$

Lemma 1. *Let $\mathcal{R}$ be a regular expression. Then, $\mathcal{A}_{\mathsf{pos}}(\mathcal{R})/_{\equiv_L}$ is homogeneous.*

5 Nondeterminism of NFAs

Many researchers have studied various measures of nondeterminism in NFAs [8,20]. Here we compare the nondeterminism of NFAs using the size of the computation graph. We give an example for the comparison of the computation tree and the computation graph.

Example 1. Let $\mathcal{A}$ be an NFA described in Fig. 5. Then, the computation tree

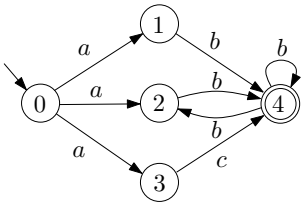

Fig. 5. An NFA $\mathcal{A}$

and the computation graph of $\mathcal{A}$ on the string *abbb* are depicted in Fig. 6.

Now let us discuss the reason why we compare the nondeterminism of NFAs using the computation graph instead of the computation tree. Let $\mathcal{A}$ be an NFA of size m and w be a string of length n. Hromkovič et al. [8] showed that the number of accepting computation can be exponential in the worst-case.

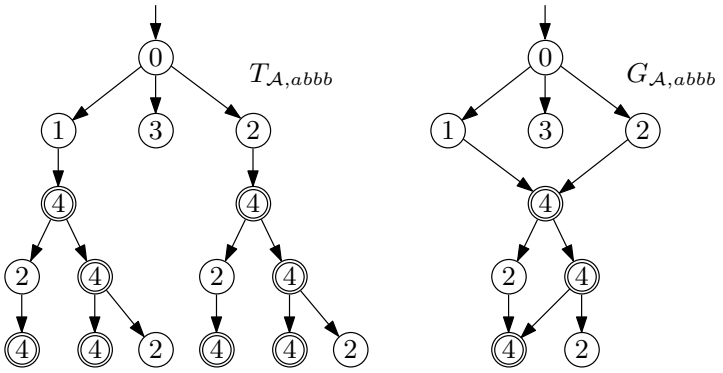

Fig. 6. The computation tree and the computation graph of the NFA $\mathcal{A}$ on the string $aaab$

This follows that the size of the computation tree can be also exponential since a computation tree has corresponding leaves for each accepting computation. This implies that the size of the computation tree can be quite different with the runtime complexity for simulating $\mathcal{A}$ on w, which is $O(m^2n)$ in the worst-case. On the other hand, the size of computation graph almost coincides with the algorithmic complexity of NFA simulation.

Lemma 2. *Given an NFA $\mathcal{A}$ with m states and a string $w \in \Sigma^*$ of length n, the following statements hold:*

(i) the number of nodes and edges in the computation graph of $\mathcal{A}$ on w are at most $mn+1$ and $m+m^2(n-1)$, respectively, and
(ii) the number of nodes and edges in the computation tree of $\mathcal{A}$ on w are at most $\frac{m^{n+1}-1}{m-1}$ and $\frac{m^{n+1}-1}{m-1}-1$, respectively.

Proof. We first prove (i). Since we assume that $\mathcal{A}$ has only one initial state, the simulation of $\mathcal{A}$ starts with one state. After then, $\mathcal{A}$ may simulate all states in the worst-case because of the nondeterminism. Therefore, the number of nodes in the computation graph of an NFA can be $mn+1$ in the worst-case.

Moreover, the graph can have m edges from the initial node since $\mathcal{A}$ can move to all states by reading a character in the worst-case. Then, since each state of $\mathcal{A}$ can have m options to move by reading a character, the number of edges in the computation graph of $\mathcal{A}$ can be $m+m^2(n-1)$ in the worst-case.

Now we prove (ii). The computation tree of $\mathcal{A}$ on w has one initial node and has m children by reading any character. Then, every node of the computation tree can have m children since there can be up to m transitions for each state and character. Thus, the total number of nodes can be

$$1+m+m^2+\cdots+m^{n-1}+m^n=\frac{m^{n+1}-1}{m-1}.$$

Note that the number of edges can be $\frac{m^{n+1}-1}{m-1}-1$ since a tree always has $t-1$ edges if there are t nodes. □

We have a simple lower bound NFA for the given upper bounds in Lemma 2, which is an m-state NFA $\mathcal{A} = (Q, \Sigma, \delta, s, F)$, where $Q = F$ and $Q = \delta(q, a)$ for all $q \in Q$ and $a \in \Sigma$. Fig. 7 depicts the lower bound when $m = 2$.

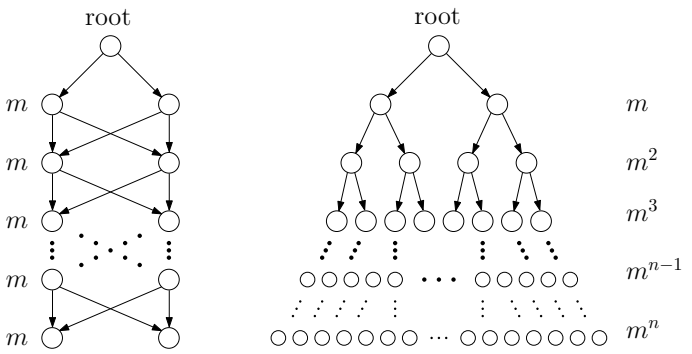

Fig. 7. A lower bound example when $m = 2$ for the upper bound in Lemma 2

We establish the following result as a corollary.

Corollary 1. *Given an NFA $\mathcal{A}$ with m states and a string $w \in \Sigma^*$ of length n, $|G_{\mathcal{A},w}|$ is in $O(m^2 n)$ and $|T_{\mathcal{A},w}|$ is in $O(m^n)$.*

However, the size of the computation graph and tree is linear in the length of the input string for DFAs in the worst-case.

Lemma 3. *Let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be a DFA and $w \in \Sigma^*$ be a string. Then, $|T_{\mathcal{A},w}|_N \leq |w| + 1$ and $|T_{\mathcal{A},w}|_E \leq |w|$.*

Proof. Since $\mathcal{A}$ is a DFA, we have only one transition from any state of $\mathcal{A}$ to proceed by reading an input character. Therefore, we always have a sequence of states visited by reading the string w instead of tree structure. If $w = \lambda$, then $|T_{\mathcal{A},w}| = 1$ since $T_{\mathcal{A},\lambda}$ consists of a single node labeled by (s, λ). Otherwise, we have an accepting computation

$$s \to q_1 \to q_2 \to q_3 \to \cdots \to q_{|w|-1} \to q_{|w|}$$

of length $|w| + 1$ in the worst-case, where $\delta(s, w_1) = q_1, \delta(q_i, w_{i+1}) = q_{i+1}$ and $q_i \in Q$ for $i = 1, \ldots, |w|$. Note that the sequence itself is the computation tree in DFAs. If the DFA is incomplete and the computation fails before completed, the length of the computation becomes shorter than $|w| + 1$. □

Corollary 2. *Let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be a DFA and $w \in \Sigma^*$ be a string. Then, $|G_{\mathcal{A},w}|_N \leq |w| + 1$ and $|G_{\mathcal{A},w}|_E \leq |w|$.*

Considering that the size of the computation graph represents the computational cost for simulating FAs well, it is evident that DFAs are much better than

NFAs regardless of the size of FAs because they make a deterministic choice at each step.

Here we consider the advantage of reducing NFAs by the left-invariant equivalence. The empirical studies on NFA reductions show that the right-invariant equivalence is more powerful in terms of better reduction on the number of states and transitions. However, it turns out that the left-invariant equivalence can reduce the nondeterminism of NFAs better than the right-invariant.

Lemma 4. *Let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be an NFA and $\mathcal{A}'$ be an NFA obtained from $\mathcal{A}$ by merging two equivalent states q and p in Q. Then, there exists a string $w \in \Sigma^*$ such that $|G_{\mathcal{A}',w}|_N < |G_{\mathcal{A},w}|_N$ if and only if $\mathcal{L}_L(p) \cap \mathcal{L}_L(q) \neq \emptyset$.*

Proof.
($\Longleftarrow$) We first prove the statement that if $\mathcal{L}_L(p) \cap \mathcal{L}_L(q) \neq \emptyset$, then there exists a string w such that $|G_{\mathcal{A}',w}|_N < |G_{\mathcal{A},w}|_N$. Since $\mathcal{L}_L(p) \cap \mathcal{L}_L(q) \neq \emptyset$, we say a string $w' \in \mathcal{L}_L(p) \cap \mathcal{L}_L(q)$. Consider the computation graphs $G_{\mathcal{A},w}$ and $G_{\mathcal{A}',w}$, where $w = w'u$ and $w', u \in \Sigma^*$. While the computation of $\mathcal{A}$ on w maintains two states p and q after reading w', the computation of $\mathcal{A}'$ only maintains the merged state. As a result, the number of nodes in the computation graph decrease.
($\Longrightarrow$) We prove that if there exists a string w such that $|G_{\mathcal{A}',w}|_N < |G_{\mathcal{A},w}|_N$, then $\mathcal{L}_L(p) \cap \mathcal{L}_L(q) \neq \emptyset$. The decrease on the number of nodes in the computation graph $G_{\mathcal{A}',w}$ implies that at least one state visited during the simulation of $\mathcal{A}$ on w is merged with the other state. This means that there exists a string w', where $w'u = w$ and $w', u \in \Sigma^*$, which makes $\mathcal{A}$ to visit the merged state. Since the merged states are p and q by assumption, $w' \in \mathcal{L}_L(p) \cap \mathcal{L}_L(q)$. □

From Lemma 4, the following result is immediate.

Theorem 1. *Let $\mathcal{A}$ be an NFA and $\equiv$ be a left-invariant equivalence. If there exist two distinct equivalent states in $\mathcal{A}$, then there exists a string $w \in \Sigma^*$ such that $|G_{\mathcal{A}/_\equiv,w}| < |G_{\mathcal{A},w}|$.*

We also observe that NFA reduction by the right-invariant does not guarantee the reduction of nondeterminism.

Corollary 3. *There exist an NFA $\mathcal{A}$ with two distinct equivalent states and a right-invariant equivalence $\equiv$ such that for any string $w \in \Sigma^*$, $|G_{\mathcal{A}/_\equiv,w}| < |G_{\mathcal{A},w}|$.*

6 Experimental Results

We present experimental results regarding the NFA reduction by invariant equivalences. Especially, we aim to analyze how the NFA reduction affects the nondeterminism of NFAs. For experiments, we have used uniformly generated random regular expressions by FAdo [1].

FAdo [1] is an ongoing project developed by Almeida et al. that provides a set of formal language manipulation tools. We have used 1,000 uniformly generated

regular expression by the FAdo system. Note that the random generation of regular expressions is based on Mairson's work [16] for generating words in a context-free language uniformly. The context-free language used for the random generation of regular expressions is presented by Lee and Shallit [15].

6.1 Size Reduction of NFAs

First we look at the size reduction of NFAs constructed from random regular expressions by invariant equivalences. See Table 1 for the result.

Table 1. The average states/transitions in position automata and reduced NFAs constructed from uniform random regular expressions

$\|\mathcal{R}\|$	$\|\Sigma\|$	Number of states/transitions				
		$\mathcal{A}_{\mathsf{pos}}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_L}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_R}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_{LR}}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_{RL}}$
20	2	13.2/21.0	11.6/16.9	9.5/14.0	9.2/13.5	9.0/13.3
	5	16.3/19.7	15.8/19.0	13.2/16.0	13.0/15.9	13.0/15.8
	10	17.3/19.5	17.1/19.2	14.8/16.9	14.7/16.8	14.7/16.7
50	2	29.8/58.3	24.4/41.4	20.2/34.7	19.0/32.4	18.6/31.8
	5	36.6/51.8	34.7/47.8	28.4/38.5	27.7/38.0	27.6/37.5
	10	40.7/50.1	39.6/48.1	34.1/41.2	33.7/40.8	33.6/40.6
100	2	57.9/122.9	45.5/83.8	38.2/70.2	35.5/65.8	34.7/63.1
	5	71.0/108.4	66.3/97.3	54.7/76.5	53.1/75.3	52.9/73.9
	10	80.0/103.0	77.8/98.9	67.1/81.9	66.1/81.1	65.9/80.4

When we compare the size reduction effects of the right- and left-invariant equivalences, it is obvious that the right-invariant equivalence is superior on average for reducing the size of NFAs from the result.

On the average of all the position automata used in the experiment, the number of states is reduced 8.3% by $\equiv_L$ and 22.7% by $\equiv_R$. The number of transitions is reduced 14.8% by $\equiv_L$ whereas 29.7% is reduced by $\equiv_R$.

We also compare two more reductions, where we reduce the NFAs in both directions. For simplicity, we write $\mathcal{A}_{\mathsf{pos}}/_{\equiv_{LR}}$ and $\mathcal{A}_{\mathsf{pos}}/_{\equiv_{RL}}$ instead of $(\mathcal{A}_{\mathsf{pos}}/_{\equiv_L})/_{\equiv_R}$ and $(\mathcal{A}_{\mathsf{pos}}/_{\equiv_R})/_{\equiv_L}$, respectively. On average, $\equiv_{RL}$ is slightly better in terms of the size reduction of NFAs than $\equiv_{LR}$ since $\equiv_{RL}$ reduces 25.0% of states and 31.6% of transitions whereas $\equiv_{LR}$ reduces 25.5% of states and 32.7% of transitions. However, the difference between the two-way reductions is very small compared to the difference between $\equiv_R$ and $\equiv_L$.

6.2 Reduction of Nondeterminism

For measuring the degree of the practical nondeterminism in NFAs, we use the following definition which can be the measurement of nondeterminism for simulating strings with the NFAs.

Let $\mathcal{A}$ be an NFA and w be a string. Then, we define the *redundancy of simulation* $\mathsf{RS}_{\mathcal{A},\mathsf{w}}$ of $\mathcal{A}$ on w as follows:

$$\mathsf{RS}_{\mathcal{A},\mathsf{w}} = \frac{|G_{\mathcal{A},w}|_E}{|w|}.$$

Recall that the number of edges in the computation graph $G_{\mathcal{A},w}$ almost coincides with the practical time complexity for simulating w on $\mathcal{A}$. We divide $|G_{\mathcal{A},w}|_E$ by $|w|$ to obtain the redundancy of simulation since $|w|$ is the optimal number of edges in the computation graph for simulating w on any FA $\mathcal{A}$ if $w \in \mathcal{L}(\mathcal{A})$. Therefore, $\mathsf{RS}_{\mathcal{A},\mathsf{w}} = 1$ if the given NFA is deterministic or simulates the string w deterministically. We conduct experiments with the randomly generated regular expressions used in the previous experiments.

For generating random strings of the regular expressions, we use a Java library called Xeger[2]. We use random strings from the regular expressions instead of uniformly generated random strings because if the computations fail easily, it is difficult to compare the nondeterminism of NFAs.

Once we choose 1,000 random regular expressions, we convert the regular expressions into position automata and reduce the automata by four different equivalences $\equiv_L$, $\equiv_R$, $\equiv_{LR}$, and $\equiv_{RL}$. Then, we generate 10,000 random strings by Xeger for each regular expression and simulate the five types of automata with the strings.

Table 2 summarizes the result of the experiment. Under the assumption that the redundancy ratio reflects the nondeterminism of NFAs for simulating strings, the best reduction is obtained by $\equiv_{LR}$ in all cases.

Table 2. The average states/transitions in position automata and reduced NFAs constructed from uniform random regular expressions

$\lvert\mathcal{R}\rvert$	$\lvert\Sigma\rvert$	$\mathsf{RS}_{\mathcal{A},\mathsf{w}}$				
		$\mathcal{A}_{\mathsf{pos}}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_L}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_R}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_{LR}}$	$\mathcal{A}_{\mathsf{pos}}/_{\equiv_{RL}}$
20	2	1.625	1.331	1.469	1.296	1.401
	5	1.103	1.046	1.096	1.046	1.076
	10	1.043	1.009	1.043	1.009	1.024
50	2	2.227	1.662	1.946	1.628	1.795
	5	1.132	1.043	1.122	1.042	1.086
	10	1.062	1.018	1.059	1.018	1.034
100	2	2.516	2.149	2.365	2.115	2.314
	5	1.172	1.066	1.164	1.065	1.112
	10	1.059	1.015	1.057	1.015	1.033

The interesting result is that $\equiv_L$ shows the better reduction than $\equiv_R$ as anticipated in Theorem 1. On average, the redundancy ratio is reduced 12.4%

[2] Xeger generates a random string from a regular expression. `https://code.google.com/p/xeger/`

by $\equiv_L$ and 4.8% by $\equiv_R$. This result clearly suggests that the reduction by the left-invariant equivalence is more useful than the right-invariant one to reduce the nondeterminism of NFAs.

One more thing to note is, $\equiv_L$ shows the better result than $\equiv_{RL}$. Recalling that $\equiv_{LR}$ shows better result than $\equiv_R$ in terms of the size reduction of NFAs, this result is noticeable. From the empirical result, $\equiv_{LR}$ can be the best option for reducing the size and the nondeterminism of NFAs at the same time.

7 Conclusions

We have studied the relationship between NFA reductions and nondeterminism of NFAs. The NFA reduction techniques based on the equivalence and preorder relations are well investigated in literature.

Here we have considered the NFA reduction by invariant equivalences. While the most of NFA constructions focus on the right-invariant equivalence for obtaining small NFAs from regular expressions, we have revealed that the reduction by left-invariant equivalence helps to reduce the nondeterminism of NFAs better than the right-invariant equivalence. We have presented empirical results with randomly generated regular expressions.

In future, we aim at comparing the NFA reduction techniques by equivalences and preorders with respect to the nondeterminism of reduced NFAs. Investigating how to optimally reduce the nondeterminism of NFAs is an open problem.

References

1. Almeida, A., Almeida, M., Alves, J., Moreira, N., Reis, R.: FAdo and gUItar: Tools for automata manipulation and visualization. In: Maneth, S. (ed.) CIAA 2009. LNCS, vol. 5642, pp. 65–74. Springer, Heidelberg (2009)
2. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science 155(2), 291–319 (1996)
3. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoretical Computer Science 120, 197–213 (1993)
4. Caron, P., Ziadi, D.: Characterization of Glushkov automata. Theoretical Computer Science 233(1-2), 75–90 (2000)
5. Champarnaud, J.-M., Coulon, F.: NFA reduction algorithms by means of regular inequalities. Theoretical Computer Science 327(3), 241–253 (2004)
6. Glushkov, V.M.: The Abstract Theory of Automata. Russian Mathematical Surveys 16, 1–53 (1961)
7. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (1979)
8. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Information and Computation 172(2), 202–217 (2002)
9. Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.) Theory Is Forever. LNCS, vol. 3113, pp. 112–124. Springer, Heidelberg (2004)

10. Ilie, L., Solis-Oba, R., Yu, S.: Reducing the size of nFAs by using equivalences and preorders. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 310–321. Springer, Heidelberg (2005)
11. Ilie, L., Yu, S.: Algorithms for computing small NFAs. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 328–340. Springer, Heidelberg (2002)
12. Ilie, L., Yu, S.: Follow automata. Information and Computation 186, 140–162 (2003)
13. Ilie, L., Yu, S.: Reducing NFAs by invariant equivalences. Theoretical Computer Science 306(1-3), 373–390 (2003)
14. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing 22(6), 1117–1141 (1993)
15. Lee, J., Shallit, J.: Enumerating regular expressions and their languages. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 2–22. Springer, Heidelberg (2005)
16. Mairson, H.G.: Generating words in a context-free language uniformly at random. Information Processing Letters 49(2), 95–99 (1994)
17. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers 9(1), 39–47 (1960)
18. Navarro, G., Raffinot, M.: Compact DFA representation for fast regular expression search. In: Proceedings of the 5th International Workshop on Algorithm Engineering, pp. 1–12 (2001)
19. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences. Cambridge University Press, New York (2002)
20. Palioudakis, A., Salomaa, K., Akl, S.G.: Comparisons between measures of nondeterminism on finite automata. In: Jurgensen, H., Reis, R. (eds.) DCFS 2013. LNCS, vol. 8031, pp. 217–228. Springer, Heidelberg (2013)
21. Thompson, K.: Regular expression search algorithm. Communications of the ACM 11(6), 419–422 (1968)
22. Wood, D.: Theory of Computation. Harper & Row (1987)