On path equivalence of nondeterministic finite automata*

Wen-Guey Tzeng*

Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050, ROC

Received 1 May 1995; revised 18 September 1995
Communicated by K. Ikeda

Abstract

Two nondeterministic finite automata (NFAs) are said to be path equivalent if each string is accepted by the two automata via the same number of computation paths. In this paper we show the following. (1) The path equivalence problem for NFAs without \( \lambda \)-cycles is solvable not only in polynomial sequential time but also in \( O(\log^*(n)) \) parallel time using a polynomial number of processors, where \( n \) is the total number of states in two NFAs. (2) The path equivalence problem for NFAs with \( \lambda \)-cycles is PSPACE-complete, hence decidable.

Keywords: Algorithms; Computational complexity; Path equivalence; Automaton

1. Introduction

Finite automata have been used to model problems in many areas such as switching circuits, concurrent processing, networks, machine learning, software engineering, etc. [6,7]. It is not only of practical interest but also of theoretical interest to determine whether two automata, which model two instances of a problem, are equivalent under different conditions. In this paper we study the path equivalence problem for nondeterministic finite automata (NFAs) with \( \lambda \)-transitions (\( \lambda \) is the null string).

A string \( x \) is accepted by an NFA \( A = (S, \Sigma, \delta, s_1, F) \) via the computation path

\[
q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_{n+1}
\]

if \( a_i \in \Sigma \cup \{\lambda\}, q_i \in S, q_{i+1} \in \delta(q_i, a_i) \) for \( 1 \leq i \leq n, q_1 = s_1, q_{n+1} \in F \) and \( x = a_1 a_2 \cdots a_n \). That is, an accepting computation path of \( x \) by \( A \) is a path in the transition diagram of \( A \) which leads from the start state to a final state and the concatenation of the passed symbols is \( x \). We note that an NFA with \( \lambda \)-transitions can accept a (finite) string via an infinite number of computation paths. An NFA is said to have \( \lambda \)-cycles if there is a cycle in its transition diagram which passes edges labeled by \( \lambda \)s only. Two NFAs are said to be path equivalent if for each string \( x \) the numbers of distinct accepting computation paths of \( x \) by the two automata are either the same or both infinite. It has been known that the path equivalence problem for NFAs without \( \lambda \)-transitions is polynomial time solvable and the same problem for NFAs with \( \lambda \)-transitions is PSPACE-hard [4,10]. However whether the latter problem is decidable was unknown.

In this paper we show that the intractability of the path equivalence problem for NFAs arises from \( \lambda \)-cycles. In other words we show the following: (1) The
path equivalence problem for NFAs without \( \lambda \)-cycles is solvable not only in polynomial sequential time but also in \( O(\log^2(n)) \) parallel time using a polynomial number of processors, i.e., in \( NC^2 \), where \( n \) is the total number of states in two NFAs. Since containing as a special case the (language) equivalence problem for deterministic finite automata, which is known to be \( NL \)-complete [5], the problem has a lower bound of \( NL \)-completeness. As it is known that \( NC^1 \subseteq NL \subseteq NC^2 \) [2], our solution is almost optimal in the sense of parallel computation time. More elaborately, we actually show that the problem is in the complexity class \( DET \) of computing determinants of integer matrices, while \( NL \subseteq DET \subseteq NC^2 \) [2]. The used parallel computation model is CREW PRAMs. (2) The path equivalence problem for NFAs with \( \lambda \)-cycles is \( PSPACE \)-complete, which solves an open problem in [4,10].

### 2. Path equivalence of NFAs without \( \lambda \)-cycles

An NFA \( A \) is a 5-tuple \( (S, \Sigma, \delta, s_1, F) \), where \( S = \{s_1, s_2, \ldots, s_n\} \) is a finite set of states, \( \Sigma \) is an input alphabet, \( \delta \) is a transition function from \( S \times (\Sigma \cup \{\lambda\}) \) to the power set of \( S \), \( s_1 \in S \) is the start state and \( F \subseteq S \) is a set of final states. A computation path \( \theta \) of \( A \) is a finite nonempty sequence

\[
((q_1, a_1), (q_2, a_2), \ldots, (q_n, a_n), q_{n+1}),
\]

where \( (q_i, a_i) \in S \times (\Sigma \cup \{\lambda\}) \) for \( 1 \leq i \leq n \), \( q_{n+1} \in S \) and \( q_{i+1} \in \delta(q_i, a_i) \) for \( 1 \leq i \leq n \). If \( q_1 = s_1 \) and \( q_{n+1} \in F \) then the sequence is said to be an accepting computation path of \( x \) by \( A \), where \( x = a_1a_2 \cdots a_n \). A \( \lambda \)-path of \( A \) is a computation path \( ((q_1, a_1), (q_2, a_2), \ldots, (q_n, a_n), q_{n+1}) \) for \( A \) such that all \( a_i \)s are \( \lambda \). If \( q_1 = q_{n+1} \) then the \( \lambda \)-path is a \( \lambda \)-cycle. If \( A \) is without \( \lambda \)-cycles then every (finite) string is accepted by \( A \) via a finite number of computation paths. Two NFAs \( A \) and \( B \) are said to be path equivalent if for each string \( x \) the number of distinct accepting computation paths of \( x \) by \( A \) is equal to that of \( x \) by \( B \) or both are infinite.

In this section, hereafter, we assume that all NFAs are without \( \lambda \)-cycles. Therefore, every string is accepted via a finite number of computation paths. Without loss of generality, we also assume that \( \Sigma = \{0, 1\} \) and \( \delta(s_1, \lambda) = \phi \), that is, there are no \( \lambda \)-edges coming out of the start state \( s_1 \). For a matrix \( M \), its \((i, j)\)-entry is \( M[i, j] \). Let \( I_n \) be the \((n \times n)\)-dimensional identity matrix. Let \( M_A(a), a \in \Sigma \cup \{\lambda\} \), be the \((n \times n)\)-dimensional matrix such that \( M_A(a)[i, j] \) is 1 if \( s_j \in \delta(s_i, a) \) and is 0 otherwise. Let

\[
M_A^*(\lambda) = I_n + \sum_{i=1}^{n-1} M_A^i(\lambda).
\]

We can see that \( M_A^*(\lambda)[i, j] \) is 1 if \( i = j \) and is the number of distinct \( \lambda \)-paths from \( s_i \) to \( s_j \) if \( i \neq j \). Let, for each \( \sigma \in \Sigma \), \( M_A^*(\sigma) = M_A(\sigma)M_A^*(\lambda) \). Then, \( M_A^*(\sigma)[i, j] \) is the number of computation paths of the form \( ((s_i, \sigma), (q_{i+1}, \lambda), \ldots, (q_{i+k}, \lambda), s_j) \) for some \( 0 \leq k \leq n - 2 \). Let \( \rho_A \) and \( \eta_A \) be the \( n \)-dimensional row vectors such that

\[
\rho_A[i] = \begin{cases} 1 & \text{if } i = 1, \\ 0 & \text{otherwise}, \end{cases} \quad \eta_A[i] = \begin{cases} 1 & \text{if } s_i \in F, \\ 0 & \text{otherwise}. \end{cases}
\]

Then, the number of accepting computation paths of string \( x = \sigma_1\sigma_2 \cdots \sigma_m \), \( m \geq 1 \) and \( \sigma_i \in \Sigma \) for \( 1 \leq i \leq m \), by \( A \) is

\[
\#A(x) = \rho_A M_A^*(\sigma_1)M_A^*(\sigma_2) \cdots M_A^*(\sigma_m)\eta_A^T.
\]

The total number of accepting computation paths of strings of length \( n \) by \( A \) is defined as

\[
\#A(n) = \sum_{|x|=n} \#A(x).
\]

Let \( A \times B \) be the Cartesian product of \( A \) and \( B \) as the standard definition. It is easy to see that

\[
\#A \times B(x) = \#A(x) \cdot \#B(x).
\]

Our parallel algorithm is based on the following lemma.

**Lemma 1.** For two NFAs \( A \) of \( n_1 \) states and \( B \) of \( n_2 \) states without \( \lambda \)-cycles and without \( \lambda \)-transitions coming out of the start states, \( A \) and \( B \) are path equivalent if and only if for all \( n \), \( 0 \leq n \leq n_1 + n_2 - 1 \),

\[
\#A \times A(n) + \#B \times B(n) = 2\#A \times B(n).
\]

**Proof.** We first prove that the path equivalence problem for NFAs with \( \lambda \)-transitions but without \( \lambda \)-cycles is polynomial time solvable. We reduce the problem to the equivalence problem of probabilistic automata.
as follows. We refer detailed definitions of probabilistic automata to the book [8]. A probabilistic automaton \( P \) is an automaton with probabilistic transitions among states. With its transition matrix \( M_P(\sigma) \), the entry \( M_P(\sigma)[i, j] \) is the probability that state \( s_i \) goes to state \( s_j \) when \( \sigma \) is input. Note that each row of \( M_P(\sigma) \) is stochastic, i.e., its entries sum to 1.

For an NFA \( C \) of \( n \) states, we construct a probabilistic automaton \( P_C \) as follows. Let \( N \) be the maximum entry in matrices \( M_F(u), CT \in \Sigma \). The probability that state \( s_i \) goes to state \( s_j \), when \( \sigma \) is input, is assigned as \( M_P(\sigma)[i, j]/(nN) \). To make \( M_P(\sigma) \) stochastic we add a new dead state to make up. Automaton \( P_C \) has an initial distribution concentrating on state \( s_i \). We can see that for every string \( x \) of length \( k \), \( P_C \) accepts \( x \) with probability \( \#c(x)/(n^kN^k) \). So, for two given NFAs \( A \) and \( B \) without \( \lambda \)-cycles probabilistic automata \( P_A \) and \( P_B \) are equivalent if and only if NFAs \( A \) and \( B \) are path equivalent. Note that in construction of \( P_A \) and \( P_B \) we should choose \( n = \max\{n_1, n_2\} \) and \( N \) be the maximum entry in matrices \( M_A(\sigma) \) and \( M_B(\sigma) \) for \( \sigma \in \Sigma \). Furthermore, by the results in [10] if \( A \) and \( B \) are not path equivalent then there is a string of length at most \( n_1 + n_2 - 1 \) that is accepted by \( A \) and \( B \) via different numbers of computation paths. Therefore, \( A \) and \( B \) are path equivalent.

Algorithm for path equivalence of NFAs without \( \lambda \)-cycles.

**Input:** NFAs \( A \) of \( n_1 \) states and \( B \) of \( n_2 \) states without \( \lambda \)-cycles;

1. Construct NFAs \( A \times A \), \( A \times B \), and \( B \times B \);
2. For \( 0 \leq n \leq n_1 + n_2 - 1 \),
   compute \( \#A \times A(n), \#B \times B(n) \) and \( \#A \times B(n) \);
3. If \( \exists n_0, 0 \leq n_0 \leq n_1 + n_2 - 1, \#A \times A(n_0) + \#B \times B(n_0) \neq 2\#A \times B(n_0) \)
   then \( A \) and \( B \) are not path equivalent
   else \( A \) and \( B \) are path equivalent.

**Complexity.** The Cartesian product of two NFAs can be computed in \( O(\log(n)) \) space since a Turing machine can scan the state transition functions of two NFAs and writes down the state transition of their product into the output tape using at most \( O(\log n) \) space in the working tape. Hence it is in \( O(\log^2(n)) \) parallel time. The matrix \( M_A(\lambda) = I_\lambda + \sum_{k=1}^{\lambda} M_A(\lambda)^k \) can be computed in \( O(\log^2(n)) \) parallel time using a polynomial number of processors [1]. The remaining work is to show that for any NFA \( A \) of \( n \) states \( \#A(m) \) can be computed in \( O(\log(n) \cdot \log(m)) \) parallel time. We observe that

\[
\#A(m) = \rho_A(M_A(0) + M_A(1))^{\eta_A}.
\]

Again this is the problem of computing powers of integer matrices.

**Theorem 2.** There is an algorithm running in \( O(\log^2(n_1 + n_2)) \) parallel time using a polynomial number of processors, that takes as input two NFAs, of \( n_1 \) and \( n_2 \) states respectively, without \( \lambda \)-cycles and determines whether they are path equivalent.

Actually, by the above discussion, the parallel complexity of the path equivalence problem for NFAs without \( \lambda \)-cycles is in \( \text{DET} \), which is the complexity class of computing determinants of integer matrices and lies in between NL and NC2 [2].

3. Path equivalence of NFAs with \( \lambda \)-cycles

In this section we show that the most general case for path equivalence of NFAs is PSPACE-complete.

For an NFA \( A = (S, \Sigma, \delta, s_1, F) \), let \( S_\lambda \) be the set of states in \( S \) by which some \( \lambda \)-cycle of \( A \) passes. This \( S_\lambda \)
can be found in polynomial time since it is equivalent to find vertices in cycles of an undirected graph, which can be found in polynomial time. Let $L(A)_\lambda$ be the set of strings that are accepted by $A$ via an infinite number of computation paths. It can be seen that $x \in L(A)_\lambda$ if and only if one of its accepting computation paths passes a state in $S_\lambda$. We construct NFA $A_\lambda$ from $A$ such that $L(A_\lambda) = L(A)_\lambda$ as follows,

$$A_\lambda = (S \times S_\lambda \times \{0,1\} \cup \{s'_1\},$$

$$\Sigma, \delta', s'_1, F \times S_\lambda \times \{1\}),$$

where

$$\delta'(s'_1, \lambda) = \{[s_1,q,0] | q \in S_\lambda\},$$

$$\delta'([p,q,b], a) = \{[r,q,b] | r \in \delta(p,a)\}$$

for $p \in S$, $q \in S_\lambda$, $b \in \{0,1\}$ and $a \in \Sigma \cup \{\lambda\}$, and $\delta'([q,q,0], \lambda)$ contains $[q,q,1]$. We also construct $A_f$ from $A$ as follows. If $s_1 \notin S_\lambda$ then $A_f = (S - S_\lambda, \Sigma, \delta', s_1, F - S_\lambda)$, where $\delta'$ is the restriction of $\delta$ on the state set $S - S_\lambda$. Otherwise $A_f$ is empty. Without loss of generality, we assume that $A_f$ has no $\lambda$-transitions coming out of the start state. It can be seen that $x$ is accepted by $A$ via a finite number of computation paths only if it is accepted by $A_f$ via the same number of computation paths. However the inverse is not true; that is, $L(A_f) \supseteq L(A) - L(A_\lambda)$ since a string accepted by $A$ via an infinite number of computation paths could also be accepted by $A$ through a computation path that does not pass any state in $S_\lambda$, which makes it in $L(A_f)$.

**Theorem 3.** NFAs $A$ and $B$ are not path equivalent if and only if $L(A_\lambda) \neq L(B_\lambda)$ or there is a string $x \notin L(A_\lambda) \cup L(B_\lambda)$ that is accepted by $A_f$ and $B_f$ via different numbers of computation paths. Therefore, the complexity of determining whether two NFAs are path-equivalent is PSPACE-complete.

**Proof.** If $L(A_\lambda) \neq L(B_\lambda)$ then $A$ and $B$ are certainly not path equivalent since some string is accepted by $A$ via an infinite number of computation paths and by $B$ via a finite number of computation paths (which could be zero), or vice versa. If $L(A_\lambda) = L(B_\lambda)$ then some string is accepted by $A$ and $B$ via different finite numbers of computation paths and thus accepted by $A_f$ and $B_f$ via different finite numbers of computation paths. This string is not in $L(A_\lambda) \cup L(B_\lambda)$.

To test whether $L(A_\lambda) = L(B_\lambda)$ can be done in PSPACE [3]. This is the difficult part that makes the problem intractable. If they are equal, it can guess a string $x$ of length less than the sum of the numbers of states of $A_f$ and $B_f$ such that it is not in $L(A_\lambda)$ (also not in $L(B_\lambda)$) and is accepted by $A_f$ and $B_f$ via different numbers of computation paths. To compute the numbers of its accepting computation paths by $A_f$ or $B_f$ can be done in polynomial time.

It was known that the problem is PSPACE-hard [10]. Therefore, the complexity of solving the problem is PSPACE-complete.

**References**


