Scanning Regular Languages by Dual Finite Automata*

Pei-Chi Wu Feng-Jian Wang Kai-Ru Young

Institute of Computer Science and Information Engineering
National Chiao Tung University
1001 Ta Hsueh Road, Hsinchu, Taiwan, R.O.C.
pcwu@csunix.csie.nctu.edu.tw fjwang@twnctu01.bitnet

ABSTRACT

A regular language is generally accepted by a single finite automaton. An approach of dual finite automata is presented here. An input string is scanned by two deterministic finite automata (DFA's): reading from the string's head and tail respectively. One of them accepts the regular language itself; the other accepts the language's reversal. Whether a string is accepted depends on the states of both automata, when their reading heads meet. Dual finite automata can be applied in compiler generation and parallel computing.

Keywords: finite automata, regular languages, compiler, parallel computing.

1. INTRODUCTION

Regular languages (sets) are generally used in compiler design, text editor, and pattern-scanning language [1]. A regular language can be accepted by a single finite automaton scanning with single reading head. An approach of dual finite automata is presented here. An input string is scanned by two deterministic finite automata (DFA's): reading from the string's head and tail respectively. One of them called *obverse* DFA accepts the regular language; the other called *reverse* DFA accepts the language's reversal. The reverse DFA is constructed by reversing obverse DFA: the initial state and final states are swapped, with transition function reversed. A reverse DFA's state is represented by a set of obverse DFA's states: a set of possible states which may come back to the start state of obverse DFA. An input string $\omega_1\omega_2$ is processed as follows: ω_1 is processed left to right by obverse DFA, and ω_2 right to left by reverse DFA. A string is accepted when both reading heads meet (i.e. between ω_1 and ω_2) and states both are *joinable*, the state of obverse one is included in that of reverse one.

The reverse DFA constructed after removing inaccessible states has minimum number of states. It is shown that the space complexity of this approach is $O(|\mathcal{L}|+|\mathcal{L}^R|)$, linear to the size of the regular language and its reversal.

Scanning string by dual finite automata can improve efficiency if both automata are implemented as units of parallel computing. Another application is in our research of compiler. Dual finite automata are used to recognize the sender and receivers of message path during bottom-up parsing [5-7].

---

* This work is supported by National Science Council, Taiwan R.O.C., under contract No. NSC 80-0408-E009-32.

2. SCANNING BY DUAL DETERMINISTIC FINITE AUTOMATA

Section 2.1 constructs the reverse DFA from the obverse DFA. Section 2.2 uses them in scanning a string. Section 2.3 discusses the duality between them.

2.1 Constructing Reverse DFA

Let $\mathcal{M}$ and $\mathcal{M}'$ respectively be the obverse DFA and reverse DFA for a regular language. Definition 2.1 constructs the reverse DFA $\mathcal{M}'$ from $\mathcal{M}$. Theorem 2.3 proves the correctness of construction.

Definition 2.1: Let obverse DFA $\mathcal{M}=(Q, \Sigma, \delta, q_0, F)$ with no inaccessible states. Let reverse DFA $\mathcal{M}'=(2^Q, \Sigma, \delta', q_0', F')$, where $\delta'(q', a)=\{ q \mid \delta(q, a)\in q', q\in Q\}$, $q_0'=F$, $F'=\{ q' \mid q'\supset\{q_0\}, q'\in 2^Q\}$.

q' here is a state of $\mathcal{M}'$, and q is a state of $\mathcal{M}$. Let q' be denoted as $[q_1', ..., q_j']$. $q\in q'$ if $q\in \{q_1', ..., q_j'\}$.

Lemma 2.2: $\delta'(q', \omega)=\{ q \mid \delta(q, \omega^R)\in q', q\in Q\}$, $q'\in 2^Q$, $\omega\in\Sigma^*$.
Proof:

Basis. $|\omega|=0$, $\delta'(q', \varepsilon)=\{ q \mid \delta(q, \varepsilon^R)\in q'\}=q'$ hold.

Assume $|\omega|=n$ hold. $\delta'(q', \omega)=\{ q \mid \delta(q, \omega^R)\in q'\}$.

Induction. Let $\alpha=a\omega$, $a\in\Sigma$, $|\alpha|=n+1$.

$\delta'(q', \alpha)=\delta'(q', a\omega)=\delta'(\delta'(q', a), \omega)$.

Let $q''=\delta'(q', a)=\{ q \mid \delta(q, a)\in q'\}$.

$\delta'(q', a\omega)=\delta'(q'', \omega)=\{ q \mid \delta(q, \omega^R)\in q''\}$

$=\{ q \mid \delta(\delta(q, \omega^R), a)\in q'\}=\{ q \mid \delta(q, (a\omega)^R)\in q'\}$.

$\delta'(q', \alpha)=\{ q \mid \delta(q, \alpha^R)\in q'\}$.

Q.E.D.

Lemma 2.3: $\mathcal{M}'$ accepts $L(\mathcal{M})^R$.
Proof:

$\omega\in L(\mathcal{M})$.

$\Leftrightarrow$ $\delta(q_0, \omega)\in F=q_0'$.

$\Leftrightarrow$ $q_0\in \delta'(q_0', \omega^R)=\{ q \mid \delta(q, \omega)\in q_0', q\in Q\}$.

$\Leftrightarrow$ $\delta'(q_0', \omega^R)\in F'$.

$\Leftrightarrow$ $\omega^R\in L(\mathcal{M}')$.

Q.E.D.

2.2 Scanning by Obverse and Reverse DFA's

A string here is scanned by the obverse DFA starting from the string's head and by reverse DFA from the tail. The states of both are *joinable* if the states of obverse one is included in that of reverse one. Theorem 2.4 shows that a string is accepted if the state of obverse one and that of reverse one are joinable.

Theorem 2.4: Let $\omega_1, \omega_2 \in \Sigma^*$. $\omega_1\omega_2 \in L(\mathcal{M})$ iff $\delta(q_0, \omega_1) \in \delta'(q_0', \omega_2^R)$.
Proof:

$\delta(q_0, \omega_1) \in \delta'(q_0', \omega_2^R)$.
$\Leftrightarrow$ $\delta(q_0, \omega_1) \in \{ q \mid \delta(q, \omega_2) \in q_0', q \in Q \}$.
$\Leftrightarrow$ $\delta(\delta(q_0, \omega_1), \omega_2) \in q_0' = F$.
$\Leftrightarrow$ $\delta(q_0, \omega_1\omega_2) \in F$.
$\Leftrightarrow$ $\omega_1\omega_2 \in L(\mathcal{M})$.

Q.E.D.

2.3 Dual Finite Automata

Theorem 2.4 seems inapplicable: 1) the reverse DFA from Definition 2.1 has exponential size; 2) merging states by applying minimization process (e.g. [1, Algorithm 3.6]) may lose the information necessary for state matching between two automata. An intuitive improvement is to construct the minimum DFA's for $\mathcal{L}$ and $\mathcal{L}^R$ first. It then applies Definition 2.1 to construct the reverse of the smaller one. However, the space complexity is $exp(\ min\ (|\mathcal{L}|, |\mathcal{L}^R|)\ )$, still inefficient.

The above treatments on $\mathcal{L}$ and $\mathcal{L}^R$ are not elegant. In practice, most of the states in reverse DFA are observed inaccessible. The DFA resulted from removing inaccessible states (as in Definition 2.5) seems to be a minimum. This property is proved in Theorem 2.6. The space complexity is then $\boldsymbol{O}(|\mathcal{L}|+|\mathcal{L}^R|)$, linear to the size of the regular language and its reversal.

Definition 2.5: Let a function *Reverse* : DFA $\rightarrow$ DFA. *Reverse*($\mathcal{M}$), or $\mathcal{M}^R$, is the DFA resulting after removing inaccessible states from $\mathcal{M}'$, where $\mathcal{M}'$ is as defined in Definition 2.1.

Theorem 2.6: $\mathcal{M}^R$ is a minimum DFA for $L(\mathcal{M})^R$.
Proof (by contradiction):

Assume that there exist distinct states p' and q' in $\mathcal{M}^R$, s.t. for any string $\alpha \in \Sigma^*$, $\delta'(p', \alpha)$ entering F' iff $\delta'(q', \alpha)$ entering F'. The following hold:

1) $p' \neq q'$:
$\exists q_x \in Q$, $q_x \in (p'-q') \cup (q'-p')$.
$q_x \notin (p' \cap q')$, and $q_x \notin (\neg p' \cap \neg q')$.

2) For any string $\alpha \in \Sigma^*$, $\delta'(p', \alpha)$ entering F' iff $\delta'(q', \alpha)$ entering F':
$\delta'(p', \alpha) = \{ q \mid \delta(q, \alpha^R) \in p', q \in Q \}$. $\delta'(q', \alpha) = \{ q \mid \delta(q, \alpha^R) \in q', q \in Q \}$.
case (a) $q_0 \in \delta'(p', \alpha)$. $q_0 \in \delta'(q', \alpha)$.
$\delta(q_0, \alpha^R) \in p'$. $\delta(q_0, \alpha^R) \in q'$.
case (b) $q_0 \notin \delta'(p', \alpha)$. $q_0 \notin \delta'(q', \alpha)$.
$\delta(q_0, \alpha^R) \notin p'$. $\delta(q_0, \alpha^R) \notin q'$.

From 1) and 2), for any string $\alpha \in \Sigma^*$, $\delta(q_0, \alpha^R) \neq q_x$.
q_x is inaccessible in $\mathcal{M}$. $\rightarrow\leftarrow$.

Q.E.D.

Theorem 2.7: Let $\mathcal{M}$ be a minimum DFA for $L(\mathcal{M})$. $\mathcal{M}$ and $(\mathcal{M}^R)^R$ are isomorphic.

Proof:

Both $\mathcal{M}$ and *Reverse*(*Reverse*($\mathcal{M}$)) are minimum DFA's for $L(\mathcal{M})$, so they are isomorphic.

Q.E.D.

Reverse function is invertible up to an isomorphism (a renaming of the states), and its inverse function is itself. It is interesting that a DFA can be minimized by applying *Reverse* function twice: first constructing its reverse DFA and then transforming back. *Reverse* function can be applied any times on a minimum DFA $\mathcal{M}$: $\mathcal{M}$, $\mathcal{M}^R$, $(\mathcal{M}^R)^R$, $((\mathcal{M}^R)^R)^R$, *etc.* Theorem 2.7 shows that both $\mathcal{M}$, $(\mathcal{M}^R)^R$, ..., and $\mathcal{M}^R$, $((\mathcal{M}^R)^R)^R$, ..., are isomorphic. We then call $\mathcal{M}$ and $\mathcal{M}^R$ as *dual finite automata* (under *Reverse* operator).

3. AN EXAMPLE

The obverse and reverse DFA's ($\mathcal{M}$ and $\mathcal{M}'$) for regular expression "**01*+10***" are shown in Figures 3.1-3.2.

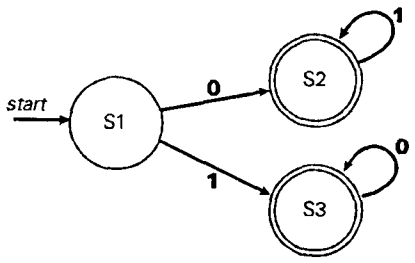

Figure 3.1: Obverse DFA for **01*+10***.

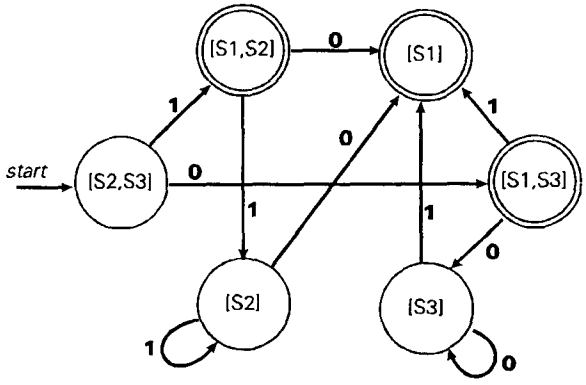

Figure 3.2: Reverse DFA for **01*+10***.

Figure 3.3 shows the transition functions of $\mathcal{M}$ and $\mathcal{M}'$. A start state has an asterisk * on its left; a final state is bold-faced; X means dead state. $\mathcal{M}''$ is reverse of $\mathcal{M}'$ by renumbering states of $\mathcal{M}'$ ([S2,S3]=1, ..., [S2]=6) and constructing reverse of $\mathcal{M}'$. The transition function of $\mathcal{M}''$ is δ'', equivalent to δ as shown. The reverse DFA constructed from $\mathcal{M}'$ is isomorphic to $\mathcal{M}$.

δ	**0**	**1**
*S1	**S2**	**S3**
S2	X	S2
S3	**S3**	X

δ'	**0**	**1**
*[S2,S3]	**[S1,S3]**	**[S1,S2]**
[S1,S3]	[S3]	**[S1]**
[S1,S2]	**[S1]**	[S2]
[S3]	[S3]	**[S1]**
[S1]	X	X
[S2]	**[S1]**	[S2]

δ''	**0**	**1**
*[235]	**[136]**	**[124]**
[136]	X	**[136]**
[124]	**[124]**	X

Figure 3.3: Transition functions for $\mathcal{M}$, $\mathcal{M}'$, and $\mathcal{M}''$.

A string "**0111**", for example, is scanned by $\mathcal{M}$ and $\mathcal{M}'$. There are five cases as shown in Figure 3.4. The states of obverse and reverse DFA's are joinable in all five cases; "**0111**" is accepted. There are also five cases for an input string "**1010**". As in Figure 3.4, one of the automata goes to dead state in cases 1, 2, 4, and 5. The reading head of both automata meet only when $\mathcal{M}$

scans "**10**" and $\mathcal{M}'$ scans "**10**" (right to left); where $\mathcal{M}$ enters `S3`, and $\mathcal{M}'$ enters `[S1]`. `S3` is not in `[S1]`; "**1010**" is rejected.

	"0111"	$\mathcal{M}$	$\mathcal{M}'$	"1010"	$\mathcal{M}$	$\mathcal{M}'$
1	**ε, 0111**	S1	[S1]	**ε, 1010**	S1	X
2	**0, 111**	S2	[S2]	**1, 010**	S3	X
3	**01, 11**	S2	[S2]	**10, 10**	S3	[S1]
4	**011, 1**	S2	[S1,S2]	**101, 0**	X	[S1,S3]
5	**0111, ε**	S2	[S2,S3]	**1010, ε**	X	[S2,S3]

Figure 3.4: Scanning the input strings "**0111**" and "**1010**".

4. APPLICATIONS AND FUTURE WORK

Scanning regular languages by dual finite automata can improve the efficiency in scanning strings if each automaton is implemented by a computing unit. A daily use system command directory listing, e.g. "ls dfa??.c" in UNIX, can be executed more faster: each file name in current directory can be scanned with double speed. A scanner of dual reading head has no difficulties in cooperating with a traditional parser. The tokens scanned by the reverse DFA can be stacked for the later usages of parser. It is interesting whether a dual reading head parser can be constructed in cooperating with a scanner. Other parallel techniques about lexical analysis and parsing can be found in [2, 4].

This work is partially conducted during our study on an object-oriented compiler specification method [5, 6] and its message passing mechanism on parse trees [7]. Regular expressions are used in modeling (specifying) propagations paths of messages in parse trees. Each path is divided into two parts processed by dual finite automata. The sender and receiver(s) of a path is recognized at their youngest common ancestor, when both the states of obverse and reverse automata are joinable. The recognitions of propagation paths are done during bottom-up parsing. We are also interested in extending the modeling language of propagation paths from regular languages to other category, e.g. context-free languages.

References

1. Aho, A. V., Sethi, R., and Ullman, J. D., *Compilers - Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. Donegan, M. K. and Katzke, S. W., "Lexical Analysis and Parsing Techniques for a Vector Machine," *ACM SIGPLAN Notices* , Vol. 10, No. 3, March 1975, pp. 138-145.
3. Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
4. Sarkar, D. and Deo, N., "Estimating the Speedup in Parallel Parsing," *IEEE Trans. Software Engineering*, Vol. 16, No. 7, July 1990, pp. 677-683.
5. Wu, P.-C. and Wang, F.-J., "An Object-Oriented Specification for Compiler," to appear in *ACM SIGPLAN Notices*.
6. Wu, P.-C. and Wang, F.-J., "Applying Classification and Inheritance to Compiling," to appear in *ACM OOPS Messenger*.
7. Wu, P.-C. and Wang, F.-J., "Message Passings on a Parse Tree," submitted to *IEEE Computer Society 1992 Int'l Conf. on Computer Languages*.