Journal of Software Engineering and Applications, **2013, 6, 33-37**
http://dx.doi.org/10.4236/jsea.2013.61005 Published Online January 2013 (http://www.scirp.org/journal/jsea)

The Equivalent Conversion between Regular Grammar and Finite Automata

Jielan Zhang[1], Zhongsheng Qian[2]

[1]Department of Information Technology, Yingtan Vocational and Technical College, Yingtan, China; [2]School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, China.
Email: changesme@163.com

Received November 21st, 2012; revised December 20th, 2012; accepted December 31st, 2012

ABSTRACT

The equivalence exists between regular grammar and finite automata in accepting languages. Some complicated conversion algorithms have also been in existence. The simplified forms of the algorithms and their proofs are given. And the construction algorithm 5 of the equivalent conversion from finite automata to left linear grammar is presented as well as its correctness proof. Additionally, a relevant example is expounded.

Keywords: Regular Grammar; Finite Automata; NFA; DFA

1. Introduction

A rapid development in formal languages has made a profound influence on computer science, especially played a greater role in the design of programming languages, compiling theory and computational complexity since formal language system was established by Chomsky in 1956. Chomsky's Conversion Generative Grammar was classified into phase grammar, context-sensitive grammar, context-free grammar and linear grammar (or regular grammar) that includes left linear grammar and right linear grammar. All these are just a simple introduction to grammar, and automata theory, which plays an important role in compiling theory and technology, has another far-reaching impact on computer science.

A regular grammar *G*, applied to formal representation and theoretical research on regular language, is the formal description of regular language, mainly describes symbolic letters and often identifies words in compiler. A finite automata *M* including NFA (Non-deterministic Finite Automata) and DFA (Deterministic Finite Automata), applied to the formal model representation and research on digital computer, image recognition, information coding and neural process etc., is the formal model of discrete and dynamic system that have finite memory, and is applied to word identification and the model representation and realization of generation process during the course of word analysis in compiler. As far as language representation is concerned, the equivalence exists between the language regular grammar G describes and that finite automata *M* identifies.

2. Some Equivalent Conversion Algorithms between Regular Grammar and Finite Automata

The definition of DFA where some notations in the remainder of this paper are shown is given first. The definition of NFA and regular grammar as well as the subset-based construction algorithm from NFA to DFA can be easily found in [1-4].

***Definition* 1.** A DFA M is an automatic recognition device that is a quintuple denoted by $(S,\sum,\delta,s_0,F)$, where each element in S indicates one state in present system; $\sum$ denotes the set of conditions under which the system may happen; δ is a single valued function from $S\times\sum$ to S with $\delta(s_1,a)=s_2$ indicating if the state of the current system is s_1 with an input a, there will be a transition from the current state to the successive one named s_2; s_0 is the very unique start state and F the set of final states.

With δ, one can easily identify whether the condition in $\sum$ can be accepted by DFA or not. Now, we extend the definition domain of δ to $S\times\sum^*$ meaning that for any $s\in S$, $a\in\sum$ and $w\in\sum^*$, $\delta(s,\varepsilon)=s$ and $\delta(s,aw)=\delta(\delta(s,a),w)$ hold. That is to say, if the condition is ε, the current state is unchanged; if the state is s and the condition aw, the system will first map $\delta(s, a)$ to s_1, then continue to map from s_1 until the last one. For some set ω where $\omega\in\sum^*$, if $\delta(s_0,\omega)=f$ where $f\in F$ holds, then we say that DFA can accept the condition set ω.

***Definition* 2.** If a regular grammar G describes the same language as that a finite automata M identifies, viz.,

$L(G)=L(M)$, then G is equivalent to M.

The following theorems are concerned about the equivalence between regular grammar and finite automata.

***Theorem* 1.** For each right linear grammar G_R (or left linear grammar G_L), there is one finite automata M where $L(M)=L(G_R)$ $\left(\text{or } L(M)=L(G_L)\right)$.

Here is the construction algorithm from regular grammar to finite automata, and the proof of correctness. It contains two cases, viz., one from right linear grammar and another from left linear grammar to finite automata.

***Construction Algorithm* 1.**

For a given right linear grammar $G_R=\langle V_T,V_N,S,\Psi\rangle$, there is a corresponding NFA

$$M=\left(V_N\cup\{f\},V_T,\delta,\{S\},\{f\}\right)$$

where f is a newly added final state with $f\notin V_N$ holding, the transition function δ is defined by the following rules.

1) For any $A\in V_N$ and $a\in V_T^*$, if $A\to a\in\Psi$ holds, then let $\delta(A,a)=f$ hold; or

2) For any $A,B\in V_N$ and $a\in V_T^*$, if $A\to aB\in\Psi$ holds, then let $\delta(A,a)=B$ hold.

Proof. For a right linear grammar G_R, in the leftmost derivation of S =>*ω ($\omega\in\sum$*), using $A\to aB$ once is equal to the case that the current state A meeting with a will be transited to the successive state B in M. In the last derivation, using $A\to a$ once is equal to the case that the current state A meeting with a will be transited to f, the final state in M. Here we let $\omega=\omega_1\cdots\omega_n$ where $\omega_i\in\sum$ $(i=1,\cdots,n)$, then S =>*ω if and only if

$$\delta(S,\ \omega)=\delta\left(\delta(S,\omega_1),\omega_2,\cdots,\omega_n\right)$$

$$=\delta\left(\delta\left(\delta(S,\omega_1),\omega_2\right),\omega_3,\cdots,\omega_n\right)$$

$$=\cdots=\delta\left(\delta\left(\cdots\delta\left(\delta(S,\omega_1),\omega_2\right),\cdots\right),\omega_i\right),\omega_{i+1},\cdots\omega_n\right)$$

$$=\cdots=\delta\left(\delta\left(\cdots\delta\left(\delta(S,\omega_1),\omega_2\right),\cdots\right),\omega_{n-1}\right),\omega_n\right)=f$$

holds.

For G_R, therefore, the enough and necessary conditions of S =>*ω are that there is one path from S, the start state to f, the final state in M. During the course of the transition, all the conditions met following one by one are just equal to ω, viz., $\omega\in L(G_R)$ if and only if $\omega\in L(M)$. Therefore, it is evident that $L(M)=L(G_R)$ holds.

***Construction Algorithm* 2.**

For a given left linear grammar $G_L=\langle V_T,V_N,S,\Psi\rangle$, there is a corresponding NFA

$$M=\left(V_N\cup\{q\},V_T,\delta,\{q\},\{S\}\right)$$

where q is a newly added start state with $q\notin V_N$ holding, the transition function δ is defined by the following rules.

1) For any $A\in V_N$ and $a\in V_T^*$, if $A\to a\in\Psi$ holds, then let $\delta(q,a)=A$ hold; or

2) for any $A,B\in V_N$ and $a\in V_T^*$, if $A\to Ba\in\Psi$ holds, then let $\delta(B,a)=A$ hold.

The proof of construction Algorithm 2 is similar to that of construction algorithm 1 and we obtain

$$L(M)=L(G_L)$$

***Theorem* 2.** For each finite automata M, there is one right linear grammar G_R or left linear grammar G_L where $L(G_R)=L(G_L)=L(M)$.

***Construction Algorithm* 3.**

For a given finite automata $M=(S,\sum,\delta,s_0,F)$, a corresponding right linear grammar $G_R=\langle\sum,S,s_0,\Psi\rangle$ can be constructed. We discuss this in two cases.

1) If $s_0\notin F$ holds, then Ψ is defined by the following rules.

For any $a\in\sum$ and $A,B\in S$, if $\delta(A,a)=B$ holds, then

a) if $B\notin F$ holds, let $A\to aB$ hold; or

b) if $B\in F$ holds, let $A\to a|aB$ hold. Or

2) if $s_0\in F$ holds, then $\varepsilon\in L(M)$ holds because of $\delta(s_0,\varepsilon)=s_0$. From step 1) we know that $L(G_R)=L(M)-\{\varepsilon\}$ holds. So, a new generation rule $s_1\to s_0|\varepsilon$ is added to G_R created from step 1) where s_1 is a newly added start symbol with the original symbol s_0 being no longer the start symbol any more and $s_1\notin S$ holding. Such a right linear grammar obtained is still named G_R, viz. $G_R=\left\langle\sum,S\cup\{s_1\},s_1,\Psi\cup\{s_1\to s_0|\varepsilon\}\right\rangle$.

3. The Improved Version for Construction Algorithm 3

Construction Algorithm 3 discussed above is complex in some sort. The following one named as Construction Algorithm 4, more easily understood, is its simplified version.

***Construction Algorithm* 4.**

For a given finite automata $M=(S,\sum,\delta,s_0,F)$, a corresponding right linear grammar $G_R=\langle\sum,S,s_0,\Psi\rangle$ can be constructed. For any $a\in\sum$ and $A,B\in S$,

1) If $\delta(A,a)=B$ holds, then let $A\to aB$ hold;

2) If $B\in F$ holds, then we add a generation rule $B\to\varepsilon$. Here B may be equal to s_0, and as long as B is a member of the set of final states, $B\to\varepsilon$ must be added.

Proof. For any $\omega\in\sum^*$ in G_R, if s_0 =>*ω holds, let $\omega=\omega_1\cdots\omega_n$ hold where $\omega_i\in\sum$ $(i=1,\cdots,n)$, we have s_0=> ω_1s_1 = > $\omega_1\omega_2s_2$ => ··· => ω_1 ··· ω_is_i => ··· => ω_1 ··· ω_n.

That's to say, s_0 = > *ω holds if and only if there is a path from s_0 meeting $\omega_1,\cdots,\omega_n$ one by one to final states in M. Therefore, $\omega\in L(G_R)$ if and only if $\omega\in L(M)$ holds, viz., $L(G_R)=L(M)$.

It is obvious that Construction Algorithm 4 is much simpler than Construction Algorithm 3.

4. The Proposed Construction Algorithm

The following Construction Algorithm 5 presented in this work as much as I know so far is an effective algorithm about the equivalent conversion from finite automata M to left linear grammar G_L according to construction algorithm 4; its proof of correctness is also given.

***Construction Algorithm* 5.**

Let a given finite automata be $M=(S,\Sigma,\delta,s_0,F)$, adding q, a new symbol, as the start symbol with $q\notin S$ holding. Let $G_L=\langle\Sigma,S\cup\{q\},q,\Psi\rangle$ hold where Ψ is defined by the following rules.

For any $a\in\Sigma$ and $A,B,f\in S$,

1) If $\delta(A,a)=B$ holds, then let $B\to Aa$ hold;

2) Add a generation rule $s_0\to\varepsilon$; and

3) For any $f\in F$, add a generation rule $q\to f$.

The rule 3) means that we add a new state q as the final state, and then link all the original final states which are no longer final ones to q through ε respectively in the state transition diagram of M.

In particular, we can let $G_L=\langle\Sigma,S,f,\Psi\rangle$ hold when F, the set of final states, contains only one final state f where Ψ is defined by the following rules.

For any $a\in\Sigma$ and $A,B\in S$,

1) If $\delta(A,a)=B$, let $B\to Aa$ hold;

2) Add a generation rule $s_0\to\varepsilon$.

Proof. For left linear grammar G_L, using $q\to f$ once is equivalent to the case one of the original states meeting ε will be transited to q in M in the very beginning of the rightmost derivation of $q=>^*\omega$ where $\omega\in\Sigma^*$; during the course of the derivation, using $B\to Aa$ once is equivalent to the case the state A meeting a will be transited to the successive state B in M; in the final step of the derivation, using $s_0\to\varepsilon$ once is equivalent to the case that the state s_0 meeting ε stops in s_0 in M. Therefore, the rightmost derivation of $q=>^*\omega$ is just the inverse chain of the path M transits from the very start state s_0 to the very final state f with all the conditions linked together in the path are just identical with ω.

Let $\omega=\omega_1\cdots\omega_n$ hold without thought where $\omega_i\in\Sigma\ (i=1,\cdots,n)$. If $q=>^*\omega$ holds, we have $q=>f=>s_{n-1}\omega_n=>s_{n-2}\omega_{n-1}\omega_n=>\cdots=>s_{i-1}\omega_i\cdots\omega_n=>\cdots=>s_0\omega_1\cdots\omega_n=>\omega_1\cdots\omega_n$, and there is a transition

$$\begin{aligned}&\delta(s_0,\omega)=\delta(\delta(s_0,\omega_1),\omega_2,\cdots,\omega_n)\\&=\delta(\delta(\delta(s_0,\omega_1),\omega_2),\omega_3,\cdots,\omega_n)\\&=\cdots=\delta(\delta(\delta(\cdots\delta(\delta(s_0,\omega_1),\omega_2),\cdots),\omega_i),\omega_{i+1},\cdots,\omega_n)\\&=\cdots=\delta(\delta(\delta(\cdots\delta(\delta(s_0,\omega_1),\omega_2),\cdots),\omega_{n-1}),\omega_n)=f\end{aligned}$$

of which each inverse step is corresponding to the one of the rightmost derivation above.

There, $\omega\in L(G_L)$ holds if and only if $\omega\in L(M)$ holds, viz. $L(G_L)=L(M)$ holds.

According to all of the above discussed and the equivalence between NFA and DFA, Theorem 2 is proved.

An example expatriated for Construction Algorithm 5 is taken as follows.

***Example* 1.** Let DFA be $M=\langle\{A,B,f\},\{0,1,2\},\delta,A,\{f\}\rangle$ which is equivalent to regular expression 02(102)* where δ satisfies $\delta(A,0)=B$, $\delta(B,2)=f$ and $\delta(f,1)=A$. The state transition diagram of M is shown in **Figure 1**. Now we can construct a left linear grammar $G_L=\langle\{0,1,2\},\{A,B,f,q\},q,\Psi\rangle$ equivalent to M where

$$\Psi=\{q\to f,f\to B2,\ B\to A0,\ A\to f1,\ A\to\varepsilon\}$$

holds.

In **Figure 1**, we can reduce G_L to $\langle\{0,1,2\},\{A,B,f\},f,\Psi\rangle$ because of only one final state f here where $\Psi=\{f\to B2,B\to A0,\ A\to f1,\ A\to\varepsilon\}$ holds. Furthermore, we can also get rid of ε from $A\to\varepsilon$ for A is not a start symbol in G_L, and then $\Psi=\{f\to B2,B\to 0\,|\,A0,A\to f1\}$ is obtained.

5. Related Work

The known proofs that the equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata are PSPACE-complete that depends upon consideration of highly unambiguous expressions, grammars and automata. R. E. Stearns and H. B. Hunt III [5] proved that such dependence is inherent. Deterministic polynomial-time algorithms are presented for the equivalence and containment problems for unambiguous regular expressions, unambiguous regular grammars and unambiguous finite automata. The algorithms are then extended to ambiguity bounded by a fixed k. Their algorithms depend upon several elementary observations on the solutions of systems of homogeneous linear difference equations with constant coefficients and their relationship with the number of derivations of strings of a given length n by a regular grammar.

V. Laurikari [6] proposed a conservative extension to traditional nondeterministic finite automata (NFAs) to keep track of the positions in the input string for the last uses of selected transitions, by adding "tags" to transitions. The resulting automata are reminiscent of nondeterministic Mealy machines. A formal semantics of auto-

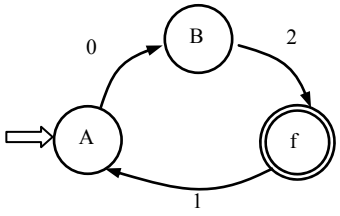

Figure 1. The state transition diagram of *M*.

mata with tagged transitions is given. An algorithm is given to convert these augmented automata to the corresponding deterministic automata, which can be used to process strings efficiently. The application to regular expressions is discussed, explaining how the algorithms can be used to implement, for example, substring addressing and a look ahead operator, and an informal comparison to other widely-used algorithms is made.

Cyril Allauzen, *et al.* [7] presented a general weighted grammar software library, the *GRM Library*, that can be used in a variety of applications in text, speech, and biosequence processing. The underlying algorithms were designed to support a wide variety of semirings and the representation and use of very large grammars and automata of several hundred million rules or transitions. They described several algorithms and utilities of this library and pointed out in each case their application to several text and speech processing tasks.

Several observations were presented on the computational complexity of regular expression problems [8]. The equivalence and containment problems were shown to require more than linear time on any multiple tape deterministic Turing machine. The complexity of the equivalence and containment problems was shown to be "essentially" independent of the structure of the languages represented. Subclasses of the regular grammars, that generated all regular sets but for which equivalence and containment were provably decidable deterministically in polynomial time, were also presented. As corollaries several program scheme problems studied in the literature were shown to be decidable deterministically in polynomial time.

Anne Brüggemann-Klein [9] showed that the Glushkov automaton can be constructed in a time quadratic in the size of the expression, and that this is worst-case optimal. For deterministic expressions, their algorithm has even linear run time. This improves on the cubic time methods.

Motivated by Li and Pedrycz's work on fuzzy finite automata and fuzzy regular expressions with membership values in lattice-ordered monoids and inspired by the close relationship between the automata theory and the theory of formal grammars, Xiuhong Guo [10] established a fundamental framework of *L*-valued grammar. It was shown that the set of *L*-valued regular languages coincides with the set of *L*-languages recognized by nondeterministic *L*-fuzzy finite automata and every *L*-language recognized by a deterministic *L*-fuzzy finite automaton is an *L*-valued regular language.

Formal construction of deterministic finite automata (DFA) based on regular expression was presented [11] as a part of lexical analyzer. At first, syntax tree is described based on the augmented regular expression. Then formal description of important operators, checking nullability and computing first and last positions of internal nodes of the tree is described. Next, the transition diagram is described from the follow positions and converted into deterministic finite automata by defining a relationship among syntax tree, transition diagram and DFA. Formal specification of the procedure is described using Z notation and model analysis is provided using Z/Eves toolset.

Sanjay Bhargava, *et al.* [12] described a method for constructing a minimal deterministic finite automaton (DFA) from a regular expression. It is based on a set of graph grammar rules for combining many graphs (DFA) to obtain another desired graph (DFA). The graph grammar rules are presented in the form of a parsing algorithm that converts a regular expression R into a minimal deterministic finite automaton M such that the language accepted by DFA M is same as the language described by regular expression R.

6. Concluding Remarks

The conversion algorithm can be realized from regular grammar to finite automata for the equivalence exists between the language regular grammar G describes and that finite automata M identifies and vice versa. In fact, the conversion between them is the very conversion between generation rules of grammar and mapping function of finite automata. The simplified forms of the conversion algorithms which are a little complicated and their proofs are given. And an algorithm about the equivalent conversion from finite automata to left linear grammar is presented as well as its correctness proof. Additionally, a relevant example is expounded.

7. Acknowledgements

This work was financially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61262010 and the Jiangxi Provincial Natural Science Foundation of China under Grant No. 2010GQS 0048.

REFERENCES

[1] H. W. Chen, C. L. Liu, Q. P. Tang, K. J. Zhao and Y. Liu, "Programming Language: Compiling Principle," 3rd Edition, National Defense Industry Press, Beijing, 2009, pp. 51-53.

[2] A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, "Compilers: Principles, Techniques, and Tools," 2nd Edition, Addison-Wesley, New York, 2007.

[3] J. E. Hopcroft, R. Motwani and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, New York, 2007.

[4] P. Linz, "An Introduction to Formal Languages and Automata," 5th Edition, Jones and Bartlett Publishers, Inc., Burlington, 2011.

[5] R. E. Stearns and H. B. Hunt III, "On the Equivalence and Containment Problems for Unambiguous Regular Expressions, Regular Grammars and Finite Automata," *SIAM Journal on Computing*, Vol. 14, No. 3, 1985, pp. 598-611. doi:10.1137/0214044

[6] V. Laurikari, "NFAs with Tagged Transitions, Their Conversion to Deterministic Automata and Application to Regular Expressions," *Proceedings of the 7th International Symposium on String Processing Information Retrieval*, IEEE CS Press, New York, 2000, pp. 181-187.

[7] C. Allauzen, M. Mohri and B. Roark, "A General Weighted Grammar Library," *Implementation and Application of Automata*, LNCS 3317, 2005, pp. 23-34. doi:10.1007/978-3-540-30500-2_3

[8] H.B. Hunt III, "Observations on the Complexity of Regular Expression Problems," *Journal of Computer and System Sciences*, Vol. 19, No. 3, 1979, pp. 222-236. doi:10.1016/0022-0000(79)90002-3

[9] A. Brüggemann-Klein, "Regular Expressions into Finite Automata," *Theoretical Computer Science*, Vol. 120, No. 2, 1993, pp. 197-213. doi:10.1016/0304-3975(93)90287-4

[10] X. H. Guo, "Grammar Theory Based on Lattice-ordered Monoid," *Fuzzy Sets and Systems*, Vol. 160, No. 8, 2009, pp. 1152-1161. doi:10.1016/j.fss.2008.07.009

[11] N. A. Zafar and F. Alsaade, "Syntax-Tree Regular Expression Based DFA Formal Construction," *Intelligent Information Management*, Vol. 4, No. 4, 2012, pp. 138-146. doi:10.4236/iim.2012.44021

[12] S. Bhargava and G. N. Purohit, "Construction of a Minimal Deterministic Finite Automaton from a Regular Expression," *International Journal of Computer Applications*, Vol. 15, No. 4, 2011, pp. 16-27.

