ONE-STACK AUTOMATA AS ACCEPTORS OF CONTEXT-FREE LANGUAGES[*]

Pradip Peter Dey, Mohammad Amin, Bhaskar Raj Sinha and Alireza Farahani
National University
3678 Aero Court
San Diego, CA 92123
{pdey, mamin, bsinha, afarahan}@nu.edu

ABSTRACT

This paper presents one-stack automata as acceptors of context-free languages; these are equivalent to Pushdown Automata which are well known in automata theory. As equivalence relations such as equivalence of Turing Machines and two-stack Pushdown Automata help in learning general properties of formal modeling, the equivalence relation of Pushdown Automata and one-stack automata also helps in learning general properties of context-free language modeling. One-stack automata are helpful to students for several reasons including: (1) their contrast with two-stack Pushdown Automata and multi-stack automata is revealing for computability; (2) their computer animation is helpful for learning their salient features; (3) their graphical representation is more easily obtained by augmenting Non-Deterministic Finite Automata for regular languages which usually precede context-free language acceptors in the logical sequence of ideas.

INTRODUCTION

The most elegant models of computation are the mathematically defined automata including Turing Machines (TMs), two-stack Pushdown Automata (2PDA), Linear Bounded Automata (LBA), Pushdown Automata (PDA) and Finite Automata (FA). These models are usually studied in the fields of theory of computation, automata theory and computability. TMs define the most powerful automata class for processing the most complex sets, namely, recursively enumerable sets. TMs are equivalent to two-stack

[*] Copyright © 2010 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

automata or 2PDA as proven by Minsky [7]. Classical forms of Pushdown Automata (PDA) are required to have exactly one stack and they are non-deterministic unless otherwise explicitly stated. PDA are acceptors of the class of Context-Free Languages (CFL's) or sets. They are less powerful than TMs; they cannot accept non-CFL's. Finite Automata (FA) define a proper subset of CFL's called regular languages denoted by regular expressions; they are equivalent to Non-deterministic FA (NFA). The above narrative information with some additional information is usually presented in a tabular form called the Chomsky hierarchy of grammars and languages as shown in Table 1 [1].

The Chomsky Hierarchy of Grammars and Languages		
Type	**Language/Grammar**	**Acceptor**
0	Recursively Enumerable	Turing Machines = 2PDA = Post Machines
1	Context-Sensitive	Linear Bounded Automata (LBA) = Turing Machines with bounded tape.
2	Context-Free	Pushdown Automata (PDA)
3	Regular	Finite Automata (FA) = NFA = Transition Graphs

Table 1: The Chomsky Hierarchy

Unlike other automata classes, PDA are not shown to be equivalent to any other acceptors [1]. This paper proves that one-stack automata are equivalent to PDA and suggests that this equivalence relation helps in learning general properties of CFL acceptors. In addition, computer animation of one-stack automata (www.asethome.org/onestack) demonstrates their revealing features [3]. Due to numerous recent research activities with multi-stack automata [2, 6] the name and the representation of one-stack automata are carefully chosen to avoid possible confusions. This paper presents essential features of one-stack automata and describes how these features and their animation promote learning CFL modeling.

ONE-STACK AUTOMATA

One-stack automata are designed as acceptors of CFL's; they are unable to accept any non-CFL. For example, $L_1 = \{ a^nb^n : \text{ where } n >= 0 \}$ is a well-known CFL [1] which can be accepted by a one-stack automaton. The definition of the automata is presented below followed by two examples.

Definition of One-Stack Automata

A one-stack automaton is composed of 5 components:

1. Σ: a finite non-empty set of symbols called alphabet from which input strings are formed.
2. S: a finite non-empty set of states including
2a. One start state (marked by -) ;
2b. Some (may be none) final states (marked by +) ;
3. Ω: an optional pushdown STACK, infinite in one direction. Initially, the STACK is empty containing all blanks (Δ).

4. Γ: an optional finite set of symbols called the STACK SYMBOLS. If the STACK is included then the STACK SYMBOLS are also included.
5. T: a finite set of transitions (edge labels) including:
5a. A finite set of transitions that show how to go from some states to some others, based on reading a specified symbol from input or the null string, ^ .
5b. Some (may be none) special transitions called push transitions each of which reads a symbol from input and inserts a symbol onto the top of the STACK.
5c. Some (may be none) special transitions called pop transitions each of which reads a symbol from input and pops a symbol from the top of the STACK.
5d. Some (may be none) special transitions each of which examines if the stack is empty by reading and popping Δ.

EXAMPLES AND EXPLANATIONS

One-stack automata are augmented from ^-NFA (NFA with null-transitions) to CFL acceptors with an optional stack and optional push and pop transitions. On the other hand, every PDA is required to have a stack and every transition in a PDA is specified with reference to the stack [4, 5, 8]. Every NFA is not a PDA, whereas every NFA is a one-stack automaton. The latter proposition helps student's understanding of the Chomsky Hierarchy, because every regular grammar is a CFG. Two examples of one-stack automata are presented below.

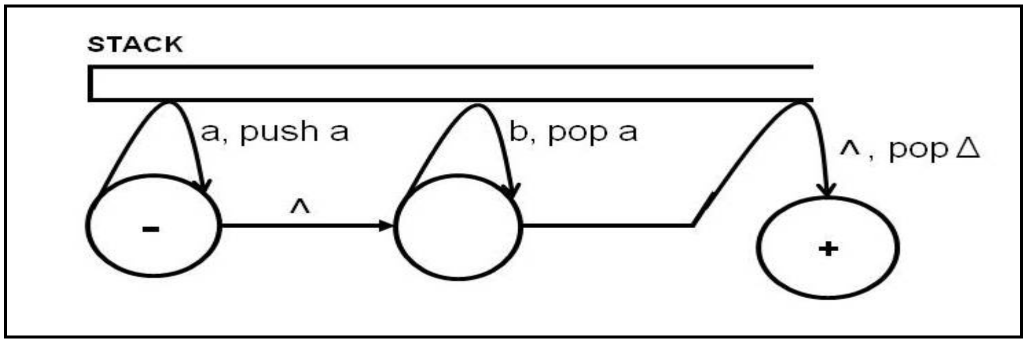


Figure 1: A one-stack automaton for { $a^n b^n$: where n >= 0 }

An example of a one-stack automaton for L_1 = { $a^n b^n$: where n >= 0 } is shown in Figure 1, where Σ is given by {a, b}; S is given by three states each of which is represented by a circle; the start state is represented by a circle marked by -; the final state is represented by a circle marked by +; Ω is given by a horizontal stack; Γ is given by {a }; T is given by a set of transitions each of which is represented by an arrow. If the string, **aabb**, is given as an input to the automaton of Figure 1 then it would be accepted by the following sequence of steps: (i) by starting the machine at the start state (marked by -) and scanning the first **a** of the input string while traversing the push transition marked by **a, push a** which pushes **a** into the stack; (ii) by scanning the second **a** while taking the same push transition marked by **a, push a** again which pushes another **a** into the stack; (iii) by taking the null-transition marked by ^ but not consuming any symbol from the input; (iv) by scanning the third symbol **b** from the input while taking the loop transition marked by **b, pop a**; (v) by scanning the final symbol **b** from the input while taking the loop transition marked by **b, pop a**; (vi) by taking the transition marked by ^, **pop** Δ. After the preceding sequence of actions the automaton meets the acceptance conditions given below and, therefore, accepts the input. The input is accepted if the following conditions are met: (a) starting with the start state (marked by -) the automaton reaches one of the accept or final states (marked by +); (b) the stack is empty; (c) the input is consumed or completely scanned.

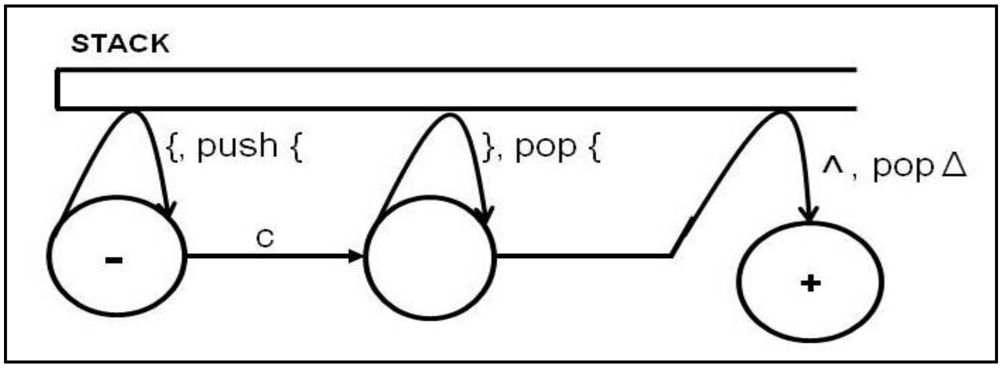


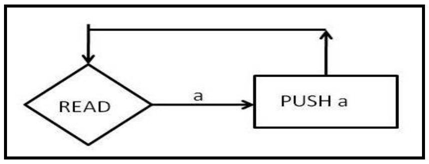
Figure 2: A one-stack automaton for $\{ \quad \{^n c\}^n : \text{ where } n >= 0 \quad \}$

The one-stack automaton of Figure 2 is similar to that of Figure 1; however, it accepts strings with matching number of {'s and }'s separated by one **c**; programming languages such as C++ have a similar string pattern. This machine will accept **{{c}}** and reject **{{{c}**.

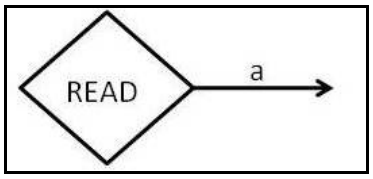
That one-stack automata are equivalent to PDA can be proven with the following two theorems. It is assumed that the reader is familiar with PDA given in textbooks such as [1]; the proofs of the theorems are given with brevity.

Theorem 1: For every one-stack automaton there is a PDA.

Proof: An arbitrary one-stack automaton can be converted into a PDA. The alphabet, Σ, is the same in both. The stack symbols and the stack are also the same in both. If the start state of the one-stack automaton has incoming transitions then using null-transitions it is mapped to a new start state without any incoming transitions. The single start state of the one-stack automaton is converted to the START state of the target PDA. If any final states of the one-stack automaton have out-going transitions then these final states are mapped to new final states without out-going transitions using null-transitions. The final states of the one-stack automaton are then converted to ACCEPT states of the PDA. Each of the push transitions of the one-stack automaton is converted to a sequence of READ-PUSH states; if a push transition of the one-stack automaton has **a, push a** then it is converted to:



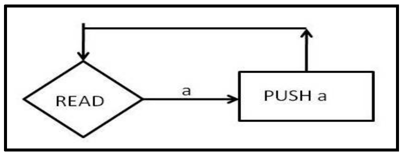
Similarly, each pop transition of the one-stack automaton is also converted to a sequence of READ-POP states of the PDA. All other transitions of the one-stack automaton are converted to the READ states of the PDA. That is a transition labeled **a** is converted to:



Null-transitions of the one-stack automaton are eliminated by creating sequences of PDA states.

Theorem 2: For every PDA there is a one-stack automaton.

Proof: Any arbitrary PDA can be converted to a one-stack automaton. The alphabet, Σ, is the same in both. The stack symbols and the stack are also the same in both. The START and ACCEPT states of the PDA are converted to the corresponding start and final states of the one-stack automaton and if necessary null-transitions are added. A sequence of READ-PUSH states is converted to a push transition of one-stack automaton; if



is given in the PDA then it is converted to a single loop push transition labeled by **a, push a**. Similarly, a sequence of READ-POP states of the PDA is converted to a single loop pop transition. Other READ states are converted to a corresponding transition with the same label. The REJECT states of the PDA are deleted; one-stack automata do not have REJECT states; they just crash if no path can be found for the letter read from the input buffer as in FA, NFA and TMs [1].

Since PDA are equivalent to one-stack automata, every formal aspect proven for PDA would apply to one-stack automata also. Thus, one-stack automata are also equivalent to CFGs, since PDA are proven to be equivalent to CFG's [1, 4, 5, 8]. Equivalence relations are often used in textbooks as learning exercises. Animation of one-stack automata clearly shows use of stack for matching strings without counting [3]. In a survey, 22 out of 25 respondents (88%) found one-stack automata helpful in learning CFL processing.

CONCLUDING REMARKS

One-stack automata are equivalent to PDA. This equivalence may be considered in a learning environment because learners may get a better understanding by comparing related features of the two. The equivalence relation of PDA and one-stack automata helps in learning general properties of CFL modeling. Some learners may also like the stack related generalization that zero-stack automata are equivalent to regular grammars; they accept regular languages; they do not accept non-regular languages. One-stack automata are equivalent to CFG's; they accept CFL's; they do not accept non-CFL's. Two or multi-stack automata are equivalent to un-restricted phrase structure grammars and they accept recursively enumerable sets.*

REFERENCES

[1] Cohen, D., *Introduction to Computer Theory*, 2nd Edition, New York, NY: John Wiley & Sons, 1997.

[2] Dassowa, J., Mitranab, V., Stack cooperation in multistack pushdown automata, *Journal of Computer and System Sciences*, 58 (3), 611-621, 1999.

[3] Dey, P. P., Farahani, A., One-Stack Automata Animation, http://www.asethome.org/onestack, retrieved April 12, 2010.

[4] Hopcroft, J. E., Motwani, R., Ullman, J., *Introduction to Automata Theory, Languages, and Computation*, Pearson Education, 2007.

[5] Lewis, H. R., Papadimitriou, C. H., *Elements of the Theory of Computation*, 2nd Edition, New Jersey: Prentice_Hall, 1998.

[6] Limaye, N., Mahajan, M., Membership Testing: Removing Extra Stacks from Multi-stack Pushdown Automata, *Language and Automata Theory and Applications*, LNCS, Springer, 2009.

[7] Minsky, M. L, Recursive insolvability of Post's problem of 'Tag' and other topics in theory of Turing Machines, *Annals of Mathematics*, 437-55, 1961.

[8] Sipser, M., *Introduction to the Theory of Computation*, 2nd Edition, PWS Publishing, 2006.

* Comments of some anonymous CCSC/NW reviewers were very useful.