Programming a Universal Push-Down Automaton

Jim Harris
Department of Mathematics and Computer Science
University of North Carolina at Pembroke
Pembroke, NC 28372

Abstract

This paper involves a discussion of software that simulates a universal push-down automaton and shows how this software is used in a theory of computation class to teach problem solving and abstract concepts. The software described is available for downloading (Windows 95/98) at http://okraboy.uncp.edu.

Introduction

In teaching a course in the theory of computation, we cover four fundamental abstract machines, the finite automaton (FA), the push-down automaton (PDA), the linearly bounded automaton (LBA), and the turing machine {TM}. The PDA is unique in that it is known that non-deterministic push-down automata (NPDA) can represent a more general class of functions than can deterministic push-down automata (DPDA) (i.e. DPDA's can translate or recognize only a proper subset of the languages that can be recognized or translated by NPDA's.). This is not true for FA's and TM's and is not yet known for LBA's.

'99 ACM Southeast Regional Conference

In teaching the theory of compu- tation many students seem to have trouble with the concept of non-determinism. While many find it easy to tell the difference between a deterministic and non-deterministic machine, actually programm-ing a problem that has only a non-deterministic solution does not seem to be as easy. By the time students reach their junior year in college, most have a procedural mindset. Programming in Prolog helps to break this mindset. The drawback is the amount of syntax the students must learn in order to teach this concept. I have found PDA's to be a good way to have students think non-deterministically without as much overhead. In order to assist, I created a universal push-down automaton that is capable of running PDA programs (much like a universal turing machine). The universal PDA can operate in both deterministic and non-deterministic modes.

Using the Universal PDA

The universal PDA is written in VB5 and consists of four windows; a control window, a program window, a tape window and a stack window.

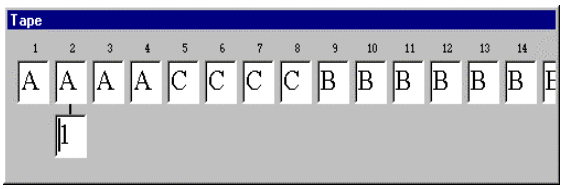

Figure 1: The Tape Window

The tape window contains an input string (entered by clicking the “Input” button on the control window). The tape cell extends well beyond the window view and scrolls automatically when the tape headmoves past the visible window. The same is true for the stack window. The tape head starts in cell one and moves one cell to the right after each instruction.

The programs are typed or loaded into the program window. Each line represents either a comment or an instruction. Comments start with a single quote.

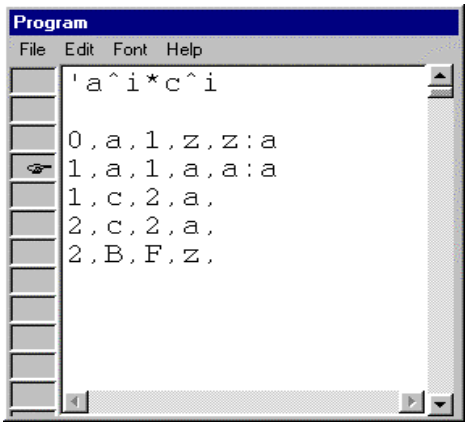

Figure 2: The Program Window

An instruction is of the form:

<start>,<input>,<next>,<pop>,<push>

where <push> is a string of symbols each separated by a colon (:) . For example, the first instruction 0, a, 1, z, z:a translates to “if in state zero with input symbol “a” and top of stack “z”, pop the “z” on the stack, go to state 1 and push a “z” then an “a” back onto the stack.. The stack initially starts with a symbol “z” marking the bottom of the stack. “B” represents a blank cell and “F” represents a final state. The symbols “#” and “&” act as single character wildcards. The <input>, <pop>, and <push> can optionally be left blank. As you can see, there is not much syntax required in order to be able to write a PDA program .

The Control Window allows the program to be executed at varying speeds or a line at a time. The direction of execution can be forwards or backwards. Breakpoints can be placed in the code by clicking the box next to the line of code. A pointer shows which line of code is currently executing. The universal PDA can be set to operate deterministically or non-deterministically. Backtracking is used in non-deterministic mode to search all paths.

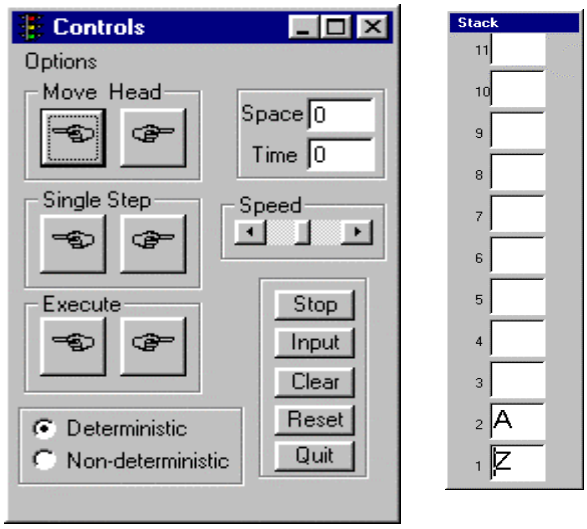

Figure 3: The Control and Stack Windows

The tape head and stack visually show the backtracking. A sample program is shown below:

'Adding four digit binary numbers. abcd + wxyz ‘are input as dzcybxaw. The result is on the stack
0,&,1,z,z:&
'There is no carry
0,B,F,Z,
0,&,1,#,#:&
1,0,0,0,0
1,0,0,1,1
1,1,0,0,1
1,1,2,1,1
'There is a carry
2,B,F,&,&:1
2,&,3,#,#:&

3,0,0,0,1
3,1,2,0,0
3,0,2,1,0
3,1,2,1,1

The PDA interpreter starts with the first instruction until it finds a match on the start state, input symbol, and top of the stack symbol. If it finds a match, it executes that instruction. The lines of code that are executed most often should be placed at the beginning of the program. This can depend on the input string, but in many cases, can be determined by analyzing the structure of the transition diagram representing the program. I usually have students draw a transition diagram showing the PDA before writing the program, much like flowcharting.

Some PDA problems

A sample PDA programming assignment is shown below:

If possible, write a deterministic PDA program to solve the following problems. If the problem has no deterministic solution, write a non-deterministic PDA program to solve the problem.

1. Accept strings of the form L1={$a^{2i}c^{3i}$, i>=1}
2. Accept strings where Σ = {a,c}and L2 = {strings that have three times as many a's as c's}
3. Accept strings where Σ = {a,c} and L3 = {strings that are odd length palindromes}
4. Evaluate a binary infix expression for Σ = {0,1,+,*} where + is bitwise "or" and * is bitwise "and" leaving the final result on the stack. Use single digits constants in the expression. (L4)
5. Accept strings of the form L5 = {$a^ib^jc^k$ | i=j or j=k}
6. Accept strings of the form L6 = {a^ib^j| i<>j}
7. Accept matching parenthesis (L7)

In the assignment above, students must recognize the problem to either have a deterministic solution or to be inherently non-deterministic. Most students recognize that problem #3 is non-deterministic since the first non-deterministic example covered in class is palindromes of even length. However, problem #5 is also non-deterministic and the solution is quite simple if you can force yourself to think non-deterministically. With just a single stack, you cannot remember if both i=j and j=k. If you could, then {$a^ib^ic^i$ | i>=1} could be recognized by a PDA and this is the classic example of a language that violates the pumping lemma for PDA's . The solution is simple if the student can get out of their "procedural" rut and let the machine decide if i=j or j=k. The problem reduces to two cases of accepting strings of the form {a^ib^i | i>=1}.

Conclusions

Having students program the universal PDA machine not only gives them a better understanding of non-determinism, but also allows them to better understand the limitations of PDA's in general and therefore better understand many inherent limitations of modern grammar-driven parsers. The simple syntax of PDA and TM languages allows even non CS majors to concentrate more on problem solving and logic under not only the constraints of the problem, but also the machine constraints. The universal PDA described in this program (along with a universal TM) can be downloaded from http://okraboy.uncp.edu.

References

Harris, James YATS – Yet Another Turing Machine Simulator, 1997 Proceedings of the Southeastern Small College Computing Conference.

Kozen, Dexter C. Automata and Computability, Springer-Verlag, 1997

Martin, John C. Introduction to Languages and Machines, McGraw-Hill , 1996