Cornerstones of Computer Science: *D*eterministic *P*ush*d*own *A*utomata
Evo—the DPDA Simulator

By Richard Born
Associate Professor Emeritus
Northern Illinois University
rborn2@niu.edu

Introduction

A pushdown automaton (PDA) is a language recognition device consisting of a finite control, input tape, and a stack (or pushdown store), as depicted in Figure 1. The input tape contains the input string which consists of characters from some input alphabet. The finite control can be in any one of a finite number of states, contains a read head that can sense one character from the input string, and has another head that may read the leftmost character (considered to be at the top of the stack) as well as write characters on the top of the stack as a replacement for the current top of stack character. In the idealized model, the stack can grow without bound to any desired size.

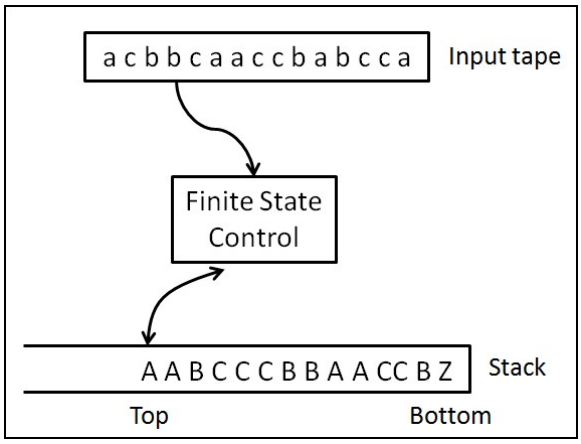

Figure 1

Although a general pushdown automaton is a nondeterministic device capable of recognizing an entire class of what are known as context-free languages, the PDA simulated by Evo in *Cornerstones of Computer Science* is deterministic and more restrictive. The students can, never-the-less, design deterministic pushdown automata (DPDAs) for recognizing a variety of interesting languages and then test the designs on input strings using Evo.

Moves for a DPDA may be of two types. The first type of move makes use of the current input character from the input string. Depending upon the current state, the input character, and the character on the top of the stack, the DPDA then goes to a next state, replaces the character on top of the stack with a (possible empty) string of characters, and then moves the input read head one character to the right. Such a move will be referred to as a real move. See Figure 2.

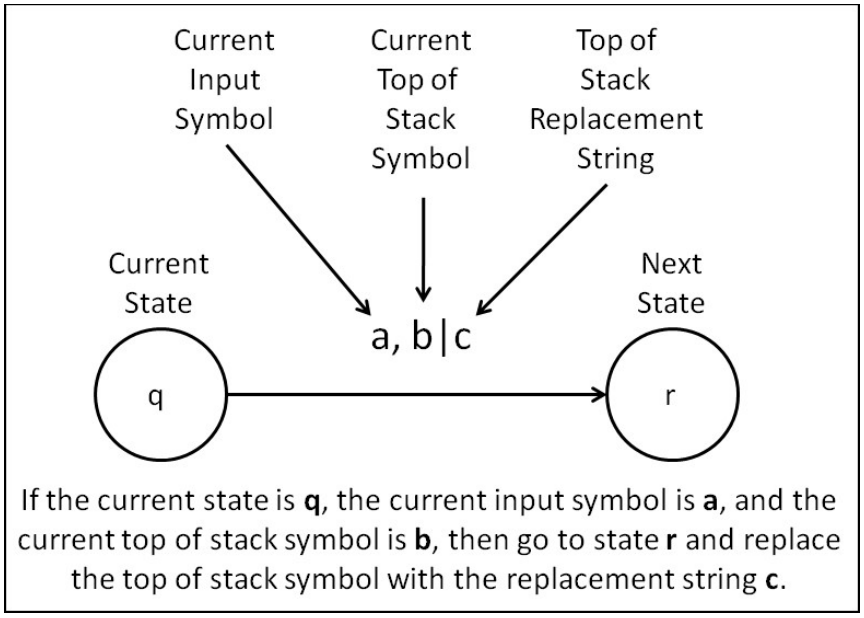

Figure 2

The other type of move is known as an epsilon-move and allows the DPDA to manipulate the stack without having to read any input characters. For this type of move, the input read head is not advanced after the move. This kind of move can also be done if the input has already been consumed—meaning that the input read head has moved to the right of the last character on the input string.

There are two common ways to define the acceptance of strings by a pushdown automaton. These are referred to as acceptance by ***empty stack*** and acceptance by ***final state***. In acceptance by empty stack, there must be a sequence of moves that causes the input string to be consumed and the pushdown automaton to empty its stack. In acceptance by final state, there must be a sequence of moves that causes the input string to be consumed and the pushdown automaton to be in what is called a ***final state***. *The Evo DPDA simulator makes use of acceptance by empty stack.* The ***language accepted by final state*** is the set of all strings accepted by final state. The ***language accepted by empty stack*** is the set of all strings accepted by empty stack.

A pushdown automaton can be loosely defined as a septuple **(Q, I, S, DELTA, q_0, Z, F)** where:

- **Q** is a finite set of states,
- **I** is an input alphabet,
- **S** is a stack alphabet,
- **q_0** is the start state,
- **Z** is the bottom-of-stack symbol (also a member of **S**),
- **F** is a set of final states (and a subset of **Q**), and
- **DELTA** (which we abbreviate as **D**) is the transition function defining the moves.

Note that no move is possible once the stack has been emptied. A DPDA will also stop if it reaches a configuration for which no move is defined.

A PDA is deterministic (*i.e.*, it is a DPDA) if two conditions are met. First, for a given character on top of the stack, a choice between a real move and an epsilon move is not allowed. Second, for any given current state, input symbol, and top-of-stack character, a choice of more than one move is not allowed.

A computation on a DPDA is commonly represented by a sequence of what are known as **configurations**. A configuration at any instant of time takes three things into account: (1) the current state, (2) the unread portion of the input tape, and the entire contents of the stack, read top-down. A first configuration **yields** a second configuration **in one step** if there is a transition that can achieve this. **Yields in one step** is denoted by the symbol → in *Cornerstones of Computer Science*.

Evo's Deterministic Pushdown Automaton

With all of the above discussion describing the theoretical model for a DPDA complete, we are now ready to look at the automaton that Evo simulates. Evo's septuple **(Q, I, S, D, q_0, Z, F)** is as follows:

- **Q** **States:** {0, 1, 2, 3}
- **I** **Input alphabet:** {1, 2, 3} plus an end-of-string marker **^**
- **S** **Stack alphabet:** {Z, A, B, C}
- **Z** **Bottom-of-stack symbol:** Z
- **q_0** **Start state:** 0
- **F** **Final states:** { }, the empty set (String acceptance is by empty stack.)
- **D** **User definable transition function** with up to 24 transitions

We see that the Evo DPDA allows up to four states numbered 0 through 3, and that the start state is always state 0. The input alphabet consists of the three characters 1, 2, and 3 plus an end of string marker ^. The stack alphabet consist of the four characters Z, A, B, and C, with the bottom-of-stack symbol Z the only symbol on the stack at simulation start. Since Evo's DPDA accepts strings by empty stack, the set of final states is immaterial. In this case, it is common to designate the set of final states by the empty set { }. Evo's stack has a maximum depth of 20 characters.

An Example DPDA

A common DPDA studied by students being introduced to pushdown automata recognizes the language that consists of all strings of the form 1^n2^n where *n* is any non-negative integer. Note that 1^n refers to a string of *n* 1's, not 1 to the *n*'th power. This DPDA therefore recognizes and accepts any string that consists of a specific number of 1's followed by the same number of 2's. Since *n* can be zero, the empty string is also accepted by this automaton.

When designing a DPDA, ***state diagrams*** are commonly used to represent the transitions, with notation like that discussed earlier in Figure 2. Figure 3 shows the state diagram for recognizing 1^n2^n and lists the transitions for the DELTA (D) function. The blank arrow on the far left simply points to the start state, which is always state 0 for the Evo DPDA simulation. All the remaining arrows show transitions. The red circled numbers in the state diagram and list of transitions are included to make it easier to discuss the purpose of each transition.

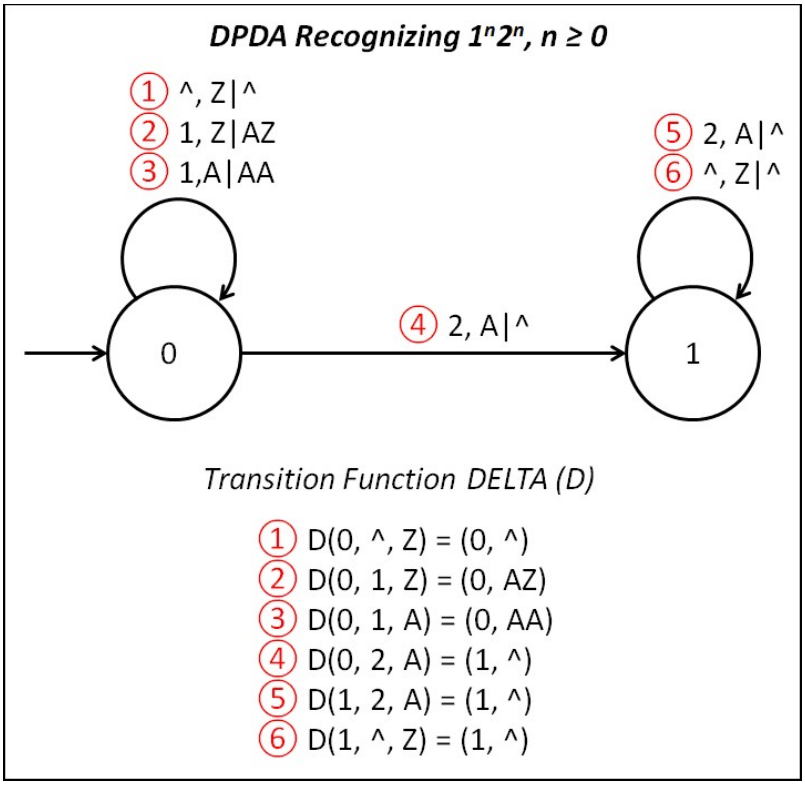

Figure 3

Transition ① accepts the empty string, *i.e.*, when the input tape is empty. This transition says "When in state 0, and at the end (^) of the input string with Z on the top of the stack, then go to state 0 and replace the top of the stack with the empty string ^. The input string has been consumed and the stack is empty. Therefore, the empty string has been **accepted by empty stack**.

The basic idea is to keep track of the number of 1's on the input tape by **pushing** an A to the top of the stack each time a 1 is read. Following the input of the 1's, each time a 2 is read, an A is **popped** from the stack. Transitions ② and ③ push an A to stack every time a 1 is read from the input tape. Transition ② says "When the DPDA is in state 0, reading a 1 with the character Z on top of the stack, then the DPDA is to go to state 0 and replace the character Z with the string AZ. Transition ④ pops the character on top of the stack when the first 2 is encountered in the input string. Transition ⑤ continues to pop an A from the stack each time a 2 is read from the input tape. Transition ⑥ empties the stack by popping the Z when the input is consumed. Transition ⑥ says "When in state 1 and the end (^) of the input string is encountered with a Z on top of the stack, then go to state 1 and replace Z with the empty string (^). The input string has been consumed and the stack is empty. Therefore, the input string has been **accepted by empty stack**.

It is important to note that the general technique is to *define transitions that will ultimately lead to acceptance of valid strings in the desired language*. Remember that when the DPDA encounters a configuration for which no transition is defined, the DPDA will stop and the input string will be rejected because the transition is not included in the DELTA function.

Now, as an example, let's take a look at the sequence of configurations that leads to acceptance of the input string 111222:

(0, 111222, Z)	→ (0, 11222, AZ)	by Transition ②
	→ (0, 1222, AAZ)	by Transition ③
	→ (0, 222, AAAZ)	by Transition ③
	→ (1, 22, AAZ)	by Transition ④
	→ (1, 2, AZ)	by Transition ④
	→ (1, ^, Z)	by Transition ⑤
	→ (1, ^, ^)	by Transition ⑥

Since (0, 111222, Z) "eventually" yields the configuration (1, ^, ^), then *this DPDA accepts the string 111222 by empty stack*.

Setting Up the Example DPDA Transitions in OzoBlockly

A program entitled *DPDA Template.ozocode* is an OzoBlockly file that accompanies this document. This program contains the transitions for our example DPDA program. It is also intended for inputting transitions for the automata of the exercises or automata of your own design. Finally, this program is used to test strings and debug your automaton logic, if necessary. The program is a large program that consumes a high percentage of the 2048 bytes of Evo's memory. However, you don't have to worry about any code other than that for your automaton's transitions. Your automaton can have up to 24 transitions, though none of the exercises uses more than about a dozen. In this section, you will learn how to set up the transitions from the example DPDA of the previous section of this document.

If you open *DPDA Template.ozocode* from within the OzoBlockly web site, you will find the transitions for the example DPDA are on the far left of the OzoBlockly workspace. Figure 4 shows these transitions along with their respective circled transition numbers discussed in the previous section of this document. Embedded in the *Define Transitions* subroutine are six calls to the subroutine *Create Transition with*. The five arguments for the *Create Transition* subroutine correspond to the five parameters of the DELTA (D) function: Current State, Current Input, Current Top of Stack, Next State, and Top of Stack Replacement String.

The Current State and Next State are always numbers 0, 1, 2, or 3 corresponding to the states is the state set Q = {0, 1, 2, 3}. These numbers can be adjusted as needed to reflect transitions in your DPDA. The Current Input is always one of the characters _1, _2, _3, or _e, corresponding to the input alphabet I = {1, 2, 3} and the end of string marker ^, respectively. Think of the underscore in the variable name as meaning *character*. For example, _1 is the character 1, and _e represents the end of string marker. These can be adjusted by clicking the down-arrow and selecting the desired character. Similarly, the Current Top of Stack is always one of the characters _Z, _A, _B, _C or _E, corresponding the stack alphabet S = {Z, A, B, C} and empty stack ^, respectively. These can also be adjusted by clicking the down-arrow and selecting the desired current top of stack character.

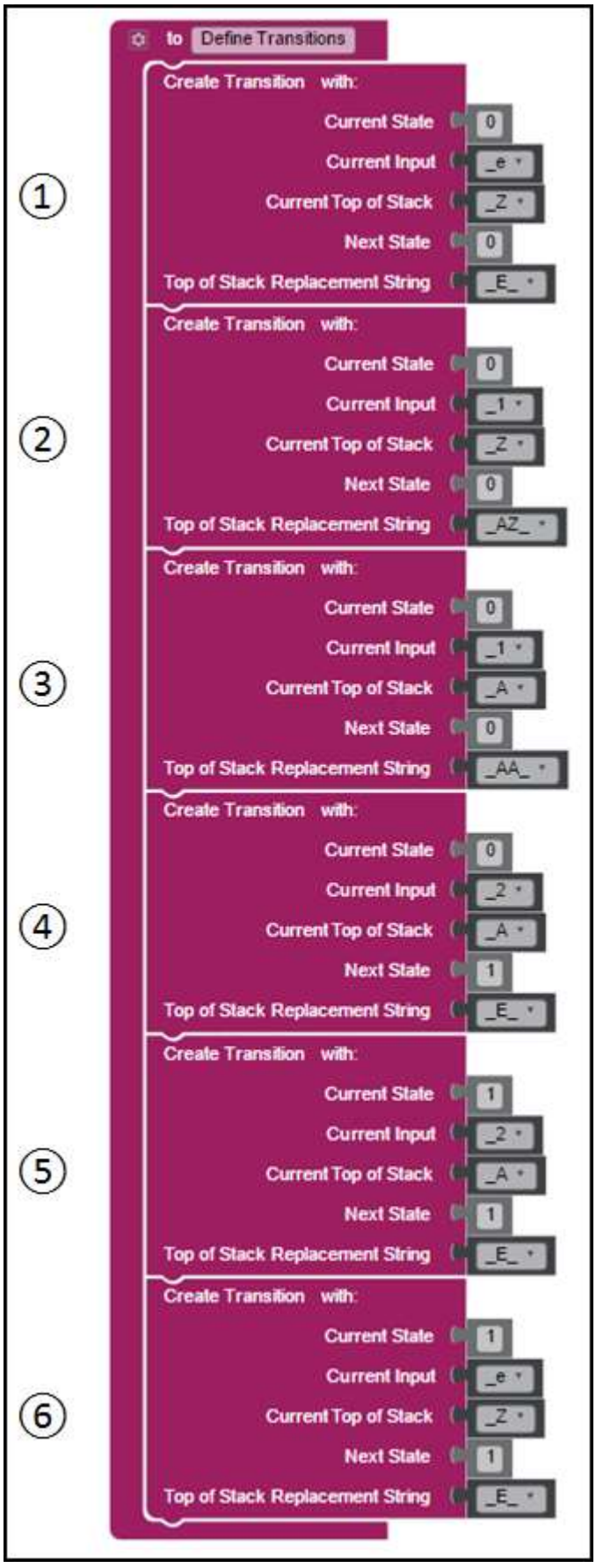
to Define Transitions
Create Transition with:
Current State 0
Current Input _e
Current Top of Stack _Z
Next State 0
Top of Stack Replacement String _E_
1
Create Transition with:
Current State 0
Current Input _1
Current Top of Stack _Z
Next State 0
Top of Stack Replacement String _AZ_
2
Create Transition with:
Current State 0
Current Input _1
Current Top of Stack _A
Next State 0
Top of Stack Replacement String _AA_
3
Create Transition with:
Current State 0
Current Input _2
Current Top of Stack _A
Next State 1
Top of Stack Replacement String _E_
4
Create Transition with:
Current State 1
Current Input _2
Current Top of Stack _A
Next State 1
Top of Stack Replacement String _E_
5
Create Transition with:
Current State 1
Current Input _e
Current Top of Stack _Z
Next State 1
Top of Stack Replacement String _E_
6

Figure 4

The Top of the Stack Replacement String is always represented by variables whose names ***begin and end*** with an underscore character. For example, _AZ_, _AA_, and _E_ are possible top of stack replacement strings. Think of the leading and trailing underscores as meaning *string*. Note than _E_ means to replace the current top of stack character with the *empty string*, i.e., a string containing no characters. The top of stack replacement strings can be adjusted by clicking the down arrow and selecting from a collection of 19 different replacement strings: _AA_, _AAA_, _AB_, _AC_, _AZ_, _BA_, _BB_, _AAZ_, _BC_, _BZ_, _CA_, _CB_, _CC_, _CZ_, _E_, _A_, _B_, _C_, and _Z_.

It is important to note that, for example, the variables _A and _A_ must not be confused. _A is the character A and _A_ is the string A. _A is used to refer to the top of stack character, while _A_ is a possible top of stack replacement string. Similarly, 1 and _1 must not be confused. 1 is a number referring to state 1, whereas _1 is the input character 1 from the input alphabet. Figure 5 emphasizes what each of the arguments in the "Create Transition" must be. If there is any misuse, results will likely be erroneous.

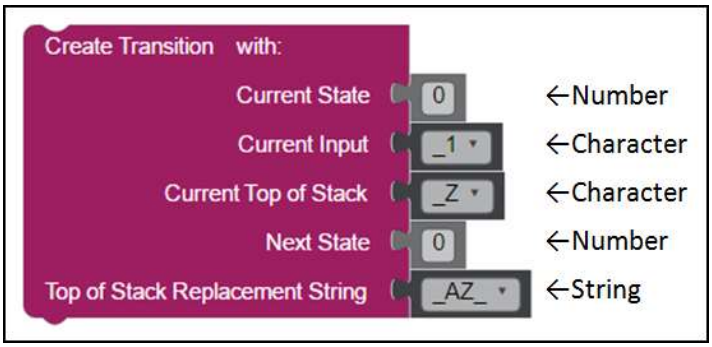

Figure 5

The suggested way to ***add new transactions*** is as follows. Right-click on any transition, and select *Duplicate* from the pop-up menu. Adjust the five arguments to the desired values. Drag the transition so that it is embedded in the *Define Transitions* subroutine. ***The order of transitions within the Define Transitions subroutine makes no difference.***

To ***delete a transaction***, right-click on the transition to be deleted, and select *Delete 6 Blocks* from the pop-up menu. To immediately undo the delete, click on the *clockwise circle-arrow* icon on the bottom right of the OzoBlockly screen.

This space is intentionally blank.

Working with the Evo DPDA Simulation Program

Let's assume that you have loaded the program *DPDA Template.ozocode* into Evo. This program will allow Evo to simulate acceptance or rejection of strings for the language 1^n2^n where *n* is any non-negative integer. For ready reference, a small copy of the Ozomap that is used for any and all DPDA's that you want to simulate using Evo is shown in Figure 6. A full page version, suitable for use with Evo, accompanies this lesson.

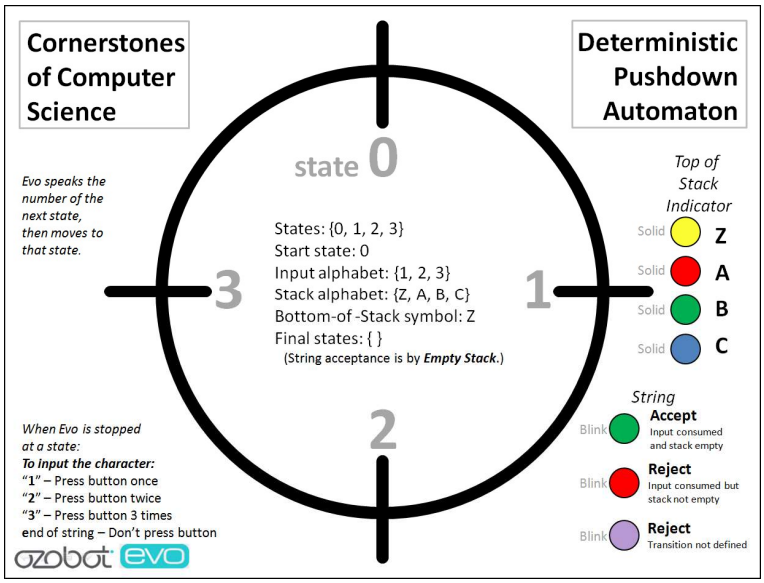

Figure 6

Evo is always initially placed on the maze centered on the intersection by the start state 0 and must be facing to the right. The information in the center of the circular path of states is a reminder of the definition of the DPDA simulated by Evo. When Evo is stopped at a state, the color of the top light indicates the symbol at the top of the stack, as detailed on the far-right center of the Ozomap: yellow for Z, red for A, green for B, and blue for C. Evo will remain stopped at any state for 8 seconds, giving you time to enter the next character from the input tape. As explained in the bottom left corner of the Ozomap, you would press Evo' button once to input the character 1, twice to input the character 2, and three times to input the character 3. Each time you press Evo's button, Evo will quickly flash white to let you know that it has recognized the button press. Make sure that you see the flash before pressing the button again. If you don't press the button at all, this is an indication that all characters on the input tape have been read. After the 8 seconds have elapsed, Evo will speak the number of the next state, and then move to that state.

If Evo accepts the string, his top light will blink green for a few seconds and Evo will then turn off. Acceptance occurs only if the input is consumed and the stack is empty. If Evo blinks red for a few seconds and then turns off, this indicates that the input string has been rejected because the input is consumed but the stack is not empty. If Evo blinks violet for a few seconds and turns off, this means that the input has been rejected because it has reached a situation for which a transition is not defined. This information is summarized in the bottom right corner of the Ozomap.

Running the Evo DPDA Simulation Program

1. Open the program *DPDA Template.ozocode* while at the OzoBlockly web site. After scrolling to the far left of the OzoBlockly workspace, you will see a subroutine called "Define Transitions". We'll assume here that you have the six transitions in this subroutine for recognizing the language 1^n2^n, where *n* is any non-negative integer. (These are the default transitions in this file, assuming you have made no changes.) Suppose that you want to test the string *12*, to determine if it is in the language 1^n2^n. This string corresponds to the case where $n = 1$.
2. Now load the program *DPDA Template.ozocode* into Evo, and then place Evo on intersection at state 0 on the Ozomap found on the final page of this document.
3. Assuming that Evo is currently turned off, press his button once to turn him on. The top and front lights will turn light blue. While they are light blue, quickly ***double-press*** Evo's button. This will start running the loaded OzoBlockly program. The top and front light will turn off for three seconds.
4. After the three seconds have elapsed, Evo' top light will turn solid yellow, indicating that Z is the current top-of-stack symbol. You will have eight seconds to enter the first character in your test string. Since the first character is 1, press Evo's button once. Evo will respond by quickly flashing his top light white to let you know that he has understood your button press.
5. When the eight-second interval has elapsed, Evo will speak the next state "1", and then move to that state on the Ozomap. His top light will turn red, indicating that A is now the current top-of-stack character. (Transition ② has just completed.)
6. The top light will remain red for eight seconds, during which you need to enter the next character in the test string. Since the next character is 2, press Evo's button twice. (Don't press twice too quickly; wait for the white flash acknowledgement after each flash.)
7. When the eight-second interval has elapsed, Evo will speak the next state "1", and then move completely around to that state again on the Ozomap. His top light will turn yellow, indicating that *A* has been replaced by the empty string and *Z* is once again the current top-of-stack character. (Transition ④ has just completed.)
8. The top light will remain yellow for eight seconds, during which you need to enter the next character in the test string. However, since you have already entered all the characters in the test string, to indicate the end of the string, don't press the button at all.
9. When the eight-second interval has elapsed, Evo will blink the top light green and then turn off. The blinking green light indicates that the string has been accepted because the input was consumed and the stack is empty. (Transition ⑥ has just completed.)
10. Test the empty string, i.e., the case where n = 0. This string will also be accepted.
11. Test the string *112*. This string will be rejected because a transition is not defined (violet flashing light).

Now that you have learned how to work with the Evo DPDA simulation program, you are ready to design state diagrams for languages in the Exercises section of this document, define your transitions in the program, and then load the program and test some strings using Evo. If you find that your design does not work correctly, you will need to put on that thinking cap a bit tighter and debug your design!

Exercises

Design state diagrams for the following languages, define your transitions in the Evo DPDA simulation program, and then load the program and test the suggested strings using Evo.

1. The language consisting of the set of all strings of *1*'s and *2*'s of the form $1^{n+1}2^n$, where $n \geq 0$.

Test strings:	^	reject	(^ is the empty string)
	1	accept	
	2	reject	
	112	accept	
	122	reject	
	11122	accept	

2. The language consisting of the set of all strings of *1*'s and *2*'s of the form 1^n2^{n+1}, where $n \geq 0$.

Test strings:	^	reject
	1	reject
	2	accept
	112	reject
	122	accept
	11	reject
	22	reject
	11222	accept

3. The language consisting of the set of all string of *1*'s and *2*'s of the form $1^{2n}2^n$, where $n \geq 0$.

Test strings:	^	accept
	112	accept
	122	reject
	1	reject
	2	reject
	111122	accept
	1121	reject
	21	reject

4. The language consisting of the set of all strings of *1*'s and *2*'s of the form 1^n2^{2n}, where $n > 0$.

Test strings:	^	reject
	1	reject
	2	reject
	122	accept
	112	reject
	112222	accept
	111122	reject
	1221	reject

5. The language consisting of the set of all strings of *1*'s and *2*'s of the form $(12)^n(21)^n$, where $n \geq 0$.

Test strings:	^	accept
	1221	accept
	1212	reject
	12122121	accept
	211	reject
	122112	reject

6. The language consisting of the set of all strings of *1*'s and *2*'s of the form $1^{2n+1}2^{2n+1}$, where $n \geq 0$.

Test strings:		
	^	reject
	1122	reject
	12	accept
	111222	accept
	11122	reject
	1	reject
	2	reject

7. The language consisting of the set of all strings of *1*'s and *2*'s of the form $1^{2n}2^{2n}$, where $n \geq 0$.

Test strings:		
	^	accept
	1122	accept
	12	reject
	211	reject
	11112222	accept
	112	reject

8. The language consisting of the set of all strings of *1*'s, *2*'s, and *3*'s of the form $(321)^n1^n$, where $n \geq 0$.

Test strings:		
	^	accept
	3211	accept
	321	reject
	32	reject
	32132111	accept

9. The language consisting of the set of all strings of *1*'s, *2*'s, and *3*'s of the form *w*3*w*', where *w* is any string (possibly empty) of *1*'s and *2*'s, and *w*' is "*w* spelled backwards".

Test strings:		
	^	reject
	131	accept
	21312	accept
	3	accept
	123321	reject
	12112321121	accept
	22131221	reject

10. The language consisting of the set of all strings of *1*'s and *2*'s for which the number of *1*'s is equal to the number of 2's.

Test strings:		
	^	accept
	121	reject
	1221	accept
	2121	accept
	1	reject
	22	reject
	11221221	accept
	211122	accept

11. The language consisting of the set of all strings of 1's, 2's, and 3's of the form $1^m2^m3^n$, where m ≥ 0 and n ≥ 0.

Test strings:							
^	accept	1	reject	23	reject	1122333	accept
12	accept	2	reject	1212	reject		
333	accept	3	accept	1122	accept		
123	accept	13	reject	11223	accept		

12. The language consisting of the set of all strings of balanced parentheses. Balanced means that each left-parenthesis has a matching right-parenthesis and that parentheses are properly nested. Since parentheses are not included in Evo's input alphabet, pretend that the character *1* is a left-parentheses (and that the character *2* is a right parenthesis).

Test strings:	^	accept
	(	reject
	)	reject
	(( )	reject
	( )	accept
	(( ) ( ))	accept
	(( ) (	reject
	((( )) ( ))	accept
	(( ) ( )))	reject

13. The language consisting of the set of all strings of 1's and 2's of the forms 1^n2^n OR 1^n, where $n \geq 0$.

Test strings:	^	accept
	1	accept
	22	reject
	111	accept
	12	accept
	112	reject
	122	reject
	111222	accept
	1122	accept
	11221	accept