$\boxed{\textbf{Pushdown Automata}}$

1 Definition

A Pushdown Automaton(PDA) is similar to a non deterministic finite automaton with the addition of a stack. A stack is simply a last in first out structure, easily visualized as a Pez Dispenser or a stack of plates. The stacks in pushdown automata are allowed to grow indefinitely and as a result of that, these machines can recognize languages that cannot be recognized by NFAs.

It is also provable that the set of languages defined by PDA are exactly the context free languages.

Formally, a pushdown automata is defined as a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is the alphabet of the language. Also a finite set.
- Γ is the stack alphabet. It is also a finite set and is allowed to have elements that are not in Σ. In particular, in JFLAP, the initial state of the stack is assumed to be the start symbol Z.
- δ is the transition function. The transition function is the critical piece in designing a PDA. The function takes 3 arguments - a state q, an input alphabet σ and a character γ that is on top of the stack. What is returned is the state that you go to as well as the symbol that is put on top of the stack in place of γ.
- q_0 as usual is used to denote the start state
- $F \subseteq Q$ is a set of final states

2 Example

We will now learn how to use JFLAP to build a PDA that recognizes the following language $L_1 = \{0^n 1^{2n} | n > 0\}$.

That is, a block of 0 followed by twice the number of 1s.

3 Intuition behind the solution

Before we dive into JFLAP, here is some intuition behind the solution. Clearly, we need something that will help us keep track of the number of 0s. The only structure we really have at our disposal is the stack. Note that since we have to accept $0^n 1^{2n}$ for any positive n, we cannot just keep making states.

To keep track of the bottom of the stack, we need to insert some initial symbol. If you are using Sipser's book, the $ symbol is used. JFLAP on the other hand reserves Z for the initial symbol that is at the bottom of the stack.

How do we count the number of 0s? Just keep pushing them onto the stack. Then the moment we get a 1, we know we have to switch to a different mode - that of matching a pair of 1s with every single 0 that is on the stack. To do this, we can pop out a 0 for every pair of 1s that are observed.

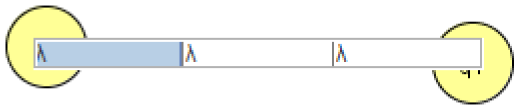
Finally, when the end of the input is reached, the stack should have only the symbol Z. That symbol can be popped out and we can transition to the final state.

Note that in this process, we also need to ensure that if the input string is not in L_1 then either we end up in a non-final state or we get stuck.

4 How to use JFLAP for PDAs

To make this PDA in JFLAP begin by making the state q_0. Now we would like to start pushing 0s on the stack.

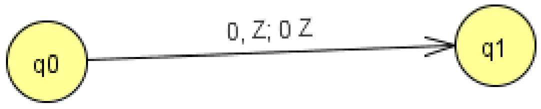
If you put down another state q_1 and make a transition from q_0 to q_1 (please see the finite automata examples if you do not recall how to do this), you will see the following, where you have to enter 3 things in order to specify the transition.



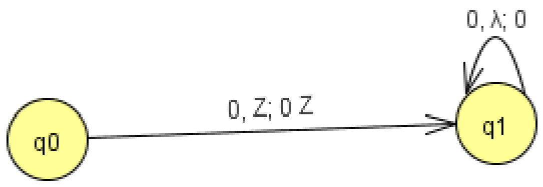
In order, the three things to enter are

1. What is the input symbol
2. What is on top of the stack (call this x)
3. What do you want on top of the stack in place of x after the transition is made.

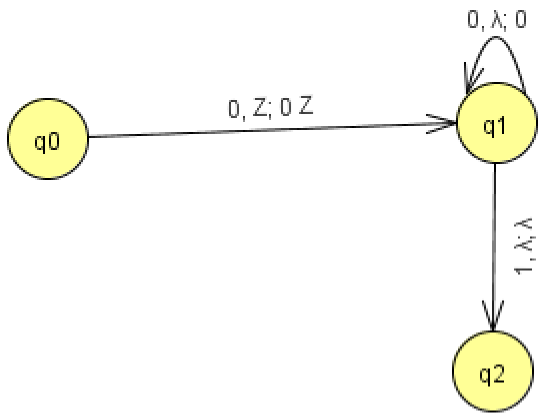
Now, we want the input string to start with 0s, so the answer to the first question is 0. The top of the stack at the beginning is always going to be Z. And at the end of the transition we want the 0 to be put on top of the Z. So the transition is going to look like this.



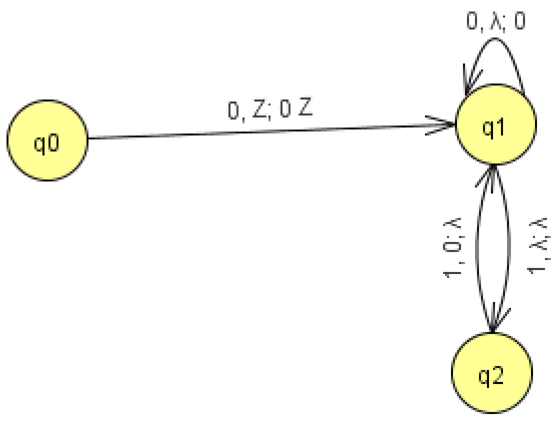
For the block of 0s that follow the first one, we just want to keep pushing them on to the stack. Note that to push something on the stack without popping anything out, you will always have λ as the second thing that you enter in your transtion. To push 0s therefore, we just have a self-loop with $0, \lambda; 0$ being the label.



Now, the moment a 1 is encountered, we need the PDA to go into a different 'mode'. A mode of counting pairs of 1s and trying to match them up with 0s. This would mean a change of state.



Note that we are not in a position to pop out a 0 yet, because at this stage we have only counted a single 1. It is for every 2 1s that we want to be popping out a 0. But when in state q_2, if we see another 1, we know that now it is time to match this pair of 1s with 0s on the stack. Hence the following transition



At this point, observe that whenever the automaton is in state q_2 it has seen an odd number of 1s and whenever it is in state q_1 it has seen as even number of 1s. If you recall building a DFA or an NFA to recognize a string with an even number (or odd number) of 1s, this should look familiar.

The last step is to accept the string when the 1s and 0s have been matched up properly and there is no more input to read.

Try it yourself!

5 Testing the PDA

Similar to the way an NFA or a DFA can be tested, a PDA can be tested in JFLAP either by clicking the multiple run option under the Input menu.

Try it yourself! - enter the following strings in the left column

```
0 1 1
1 0
0 1
0 0 1 1
0 0 1 1 1 1
λ
0 0 0 1 1 1 1
1 0 1 0
1 0 1 1 1 0
```

What results do you get? Are they consistent with what this PDA should and should not be accepting?

6 Q & A

A few questions to improve your understanding of PDAs and how they work in JFLAP.

1. How do we change this PDA if we want it to accept the language $L_2 = \{0^n 1^n\}$?
2. Show how $a^m b^n$ for (any m and n) - We note that this can be done with a stack or without. So this is both a regular language and a context free language.
3. Consider the language $a^n b^n a^m$ - A PDA can handle this. Why? Try and do it in JFLAP.
4. Consider the language $a^n b^n a^n$ - a PDA cannot handle this. Why? Try and do it in JFLAP.