

Some Sample PDAs

CS 2MJ3 Fall 2009

Nick James

October 15, 2009

NOTE: All the PDAs in this document are taken from notes I prepared during prior offerings of this course. Consequently, they are built using the PDA model covered by Kozen¹ (which is slightly different from the one Fernández uses). They're all the same, though. You can use either model. Kozen's model looks like this:

$$M = (Q, \Sigma, \Gamma, \delta, q, \perp, F)$$

where,

- Q is the set of states
- Σ is the alphabet
- Γ is the set of stack symbols
- δ is the state transition function
- q is the start state
- \perp is the initial stack symbol (the stack has this on it before the machine begins reading the string).
- F is the set of final (accepting) states. It is usually left empty when the machine is supposed to accept by empty stack.

Here are the primary differences between the two models:

- Kozen's PDAs all begin with an initial symbol sitting on the top of the stack before the PDA begins reading the string. Fernández's doesn't.
- Kozen's PDAs come in two varieties: those which accept by final state and those which accept by empty stack. Fernández has only PDAs which accept by final state. A PDA which accepts by final state accepts a string if there is a way to reach one of the final states while reading it (just like NFAs). A PDA which accepts by empty stack has no final states (more precisely, it needs no final states). It accepts a string if there is a way to read it that leaves the PDA's stack completely empty as soon as the string has been consumed (read).

¹"Automata and Computability," by Dexter C. Kozen, Springer, 1997

- Kozen’s PDAs MUST pop the symbol from the top of the stack during every state transition; if the stack is empty and the machine tries to read another character in the string, it “crashes.” Fernández allows the PDA to leave the stack untouched. His PDAs can freely read the string and change state even while the stack is empty.
- Kozen’s PDAs can push arbitrarily many symbols on the stack during a single state transition.

The last two items essentially concern the signature of the transition function, δ . Kozen’s δ looks like this:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$$

Fernández’s δ looks like this:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$$

1 PDA for $\{x \in \{a, b\}^* : x = \bar{x}\}$ (i.e. the set of palindromes over $\{a, b\}$)

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \emptyset)$$

δ -Transitions:

1. $(q, a, S) \rightarrow (q, SA)$
2. $(q, a, S) \rightarrow (q, \varepsilon)$
3. $(q, \varepsilon, S) \rightarrow (q, \varepsilon)$
4. $(q, b, S) \rightarrow (q, SB)$
5. $(q, b, S) \rightarrow (q, \varepsilon)$
6. $(q, a, A) \rightarrow (q, \varepsilon)$
7. $(q, b, B) \rightarrow (q, \varepsilon)$

The only state is q , so we can pretty much ignore states. We start with S on the stack. If the input string is ε , we must choose 3, leaving us with an empty stack (and thus we accept ε).

If we read a or b we can either get rid of the S (by using 2 or 5) or we can replace it with SA (when we read an “ a ”) or SB (when we read a “ b ”). In this stage we are recording the a ’s and b ’s we’ve read in the order they appeared.

S represents the middle of string. While it’s still there, we’re essentially reading the first half of the string and taking notes (on the stack) about what we read. When we hit the middle of the string, we get rid of S and begin the stage in which we check that the second half matches what we read in the first half. For an odd-length string, we use 2 or 5 to get rid of it. For an even-length string, we use 3 to get rid of it.

Once we get rid of S we’re in the checking phase. The stack contains the upper case version of the characters we expect to read, in the order we expect them. We use 6 and 7 entirely while reading this second part of the string.

2 PDA for $\mathcal{A} = \{x \in \{a, b\}^* : \#a(x) = \#b(x)\}$

Note that “ $\#a(x)$ ” means “the number of a 's in the string, x .” We'll make a machine, M , which accepts by empty stack:

$$M = (\{q\}, \{a, b\}, \{A, B, \perp\}, \delta, q, \perp, \emptyset)$$

δ -Transitions:

$$\begin{aligned}(q, a, \perp) &\rightarrow (q, A\perp) \\(q, b, \perp) &\rightarrow (q, B\perp) \\(q, a, A) &\rightarrow (q, AA) \\(q, b, B) &\rightarrow (q, BB) \\(q, a, B) &\rightarrow (q, \varepsilon) \\(q, b, A) &\rightarrow (q, \varepsilon) \\(q, \varepsilon, \perp) &\rightarrow (q, \varepsilon)\end{aligned}$$

So what's going on here...?

The initial stack symbol is \perp and the machine could read either an a or a b as the first character. The first two transitions are designed for this case. “If only \perp is on the stack and you read an a or a b , just make a note of which one you read by pushing the corresponding Γ symbol on the stack.”

These transitions could also be used if the machine has read only a substring of the input string that just happens to be in \mathcal{A} . For example, it might read, $abba$. After the first two characters (ab), the machine has read a portion of the input that would be in \mathcal{A} . So for all intents and purposes, that part of the input may as well not even be present.

In the third, fourth, fifth and sixth transitions, we're “stockpiling” the same characters or deleting ones that were already accumulated. When A is on the top of the stack, the only way to get rid of it is to read a b (transition 6), and vice versa (transition 5). So if A is on the top of the stack and we read a , that's just *another* a we'll have to save for later—hoping we find a b to go with it.

The last transition offers a way to clear that \perp off the stack when the input has been read. Without that, the machine could not accept any strings (because it could never empty its stack). Note that this last transition makes it a non-deterministic machine. It's the only transition that could be chosen over another, namely either of the first two transitions. If either of those first two transitions could be taken at some point, the last one could be used instead. So it *could* use that last transition anytime the stack has no A s or B s (e.g. before any input symbols have been read). That doesn't necessarily mean M would reject the string; it just means it made a misstep—a bad guess along the way.

We (as smart people) know that it shouldn't bother to apply that last transition until the input has been exhausted, but there's no way to teach the machine to behave this way. We can only offer it the option to use it whenever \perp is on the top of the stack, knowing that the machine will try everything possible before

deciding whether to accept the string.

Sorry, I'm not sure I explained that very well. Please write me if you're currently holding your head, muttering, "...the hell? I don't even... I mean, WHAT?!" Well, as usual, I guess. Always write me as soon as possible if the course material makes you do that.

3 PDA for $\mathcal{B} = \{a^n b^n : n \geq 0\}$

You'd think that this PDA should be almost identical to the last one since the languages are so similar. I mean \mathcal{B} is the same as \mathcal{A} , except all the a s have to come first. So how do we make such a picky machine?

I think the easiest way to approach it is to use states. We're allowed to use multiple states when building a PDA; nobody says we can have only one, so let's use them! We want one state, p , to represent a "reading a s" state, and the other state, q , to represent a "reading b s" state. The machine will start in state p , expecting to consume all the a s first. When the a s run out, the machine can switch over to state, q . The important thing is that if it's in state, q and suddenly it encounters an a (of all things!), it can put its foot down and reject the string.

So here's our machine:

$$M = (\{p, q\}, \{a, b\}, \{A, \perp\}, \delta, p, \perp, \emptyset)$$

δ -Transitions:

$$\begin{aligned}(p, \varepsilon, \perp) &\rightarrow (p, \varepsilon) \\(p, a, \perp) &\rightarrow (p, A\perp) \\(p, a, A) &\rightarrow (p, AA) \\(p, b, A) &\rightarrow (q, \varepsilon) \\(q, b, A) &\rightarrow (q, \varepsilon) \\(q, \varepsilon, \perp) &\rightarrow (q, \varepsilon)\end{aligned}$$

The first transition is to allow M to accept the empty string. That's it. It can be applied in only the first step because that's the only time you'll see \perp on top of the stack. So the machine will accept x via that transition if and only if $x = \varepsilon$. If $x \neq \varepsilon$, it could still apply it, of course, but in order to accept x , the stack must be empty AND the input string must be completely consumed (no deal unless both conditions are satisfied).

The second and third transitions start and keep the machine going, reading a s. It puts A on the stack whenever it reads an a so it can remember how many it has read. This is necessary for when it starts reading b s.

Ah! Now here's the "exciting" part (if, indeed, any excitement whatsoever could be derived from this exercise). So far, the machine has been in state, p , happily reading a s and keeping track of them. Suddenly it encounters its first b ! That means it must now switch over to the "reading b s" state (q), and delete the

topmost A from its stack (since it read a b to match it). From now on, no more as ! It will simply crash if it ever reads another horrible a . It said goodbye to its former, dreary life of reading as the moment it saw that first b , and it'll be damned if it's going to back out now. bs are the future! (Ok, ok, I'll stop. I promise.)

This vow to read only bs is evident from the fact that if it's in state q , there are no transitions that map (q, a, X) to *anything* (where $X \in \Gamma$). Transition 5 keeps it reading bs , and deleting the corresponding As from the stack. If it manages to exhaust the input without crashing and with only \perp remaining on the top of the stack, it can finally invoke the last transition—clearing the stack and thus accepting the string.

4 PDA for $L(a^*b^+)$

$M = (\{q\}, \{a, b\}, \{\perp, B\}, \delta, q, \perp, \emptyset)$ where δ is defined by the following transitions:

$$(q, a, \perp) \rightarrow (q, \perp)$$

$$(q, b, \perp) \rightarrow (q, B)$$

$$(q, b, B) \rightarrow (q, B)$$

$$(q, \varepsilon, B) \rightarrow (q, \varepsilon)$$