Today(s)'s lectures: From PDAs to CFLs

Reading:

October 19: From PDAs to CFLs
Read: *Kozen* lecture 25 (or *Sipser* 2.2).

October 21: Deterministic PDAs
Read: *Kozen* lectures E and F.

October 24: Parsing
Read: *Kozen* lecture 26 (not in *Sipser*).

October 26: Midterm: In class

October 28: A Parsing Algorithm
Read: *Kozen* lecture 27 (not in *Sipser*).

October 31: Everything else about CFLs

November 2: Turing Machines and Effective Computability
Read: *Kozen* lecture 28 (or *Sipser* 3.1).

November 4: More Turing Machines
Read: *Kozen* lecture 29 (or *Sipser*3.1 again).

November 7: Modified Turing Machines
Read: *Kozen* lecture 30 (or *Sipser*3.2).

November 9: Diagonalization and the Halting Problem
Read: *Kozen* lecture 31 (or *Sipser*4.2).

November 14: Decidability
Read: *Kozen* lecture 32 (or *Sipser*4.1).

November 16: Reductions
Read: *Kozen* lecture 33 (or *Sipser*chapter 5).

November 18: Gödel's Theorem
Read: *Kozen* lecture 38 (or *Sipser*6.2).

November 21: Everything else about Turing Machines

November 23&25: Timed Automata

November 28&30: Theorem Proving

December 2: Something Fun

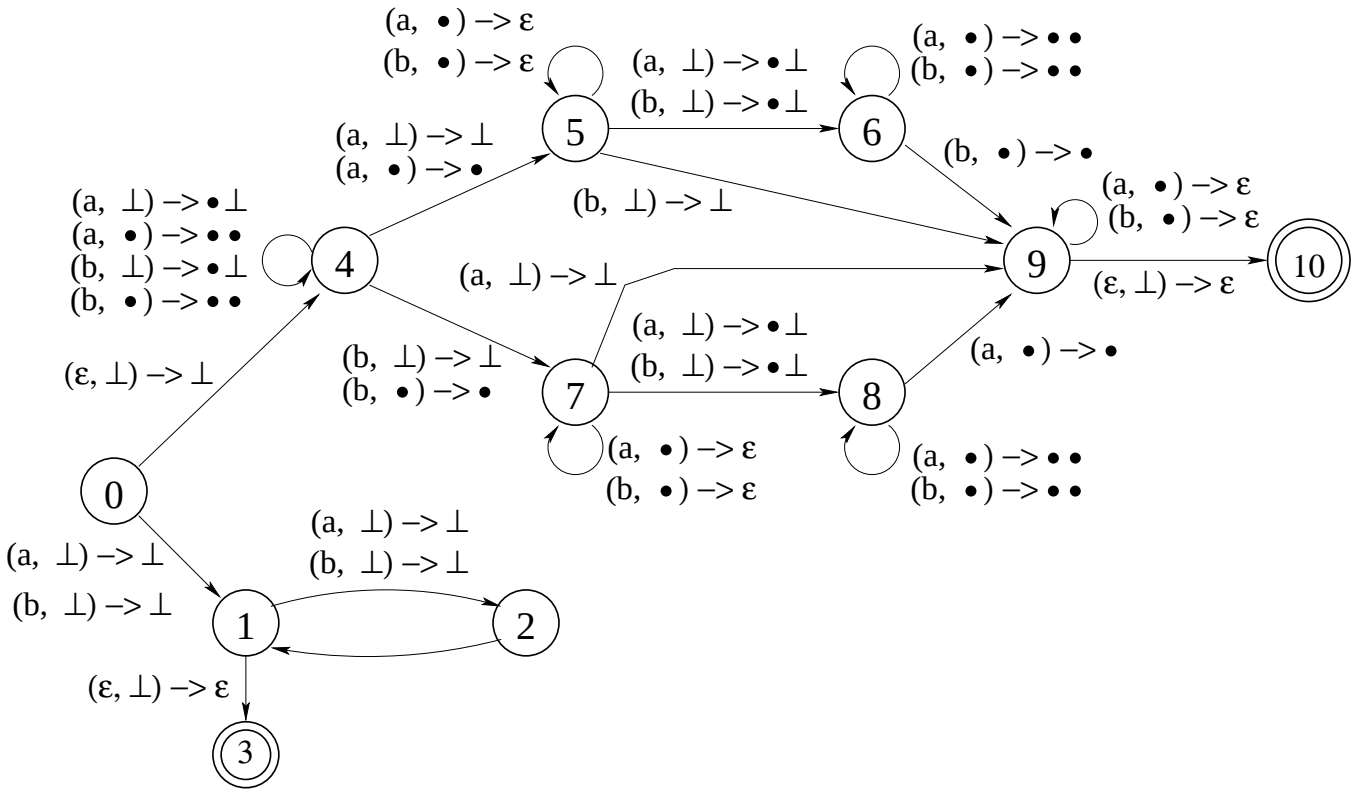

Figure 1: A NPDA for $\{x| \sim \exists w.\ x = ww\}$

I. A non-deterministic automaton with multiple states

A. Figure 1 shows a NPDA that recognizes the language $B = \{x| \sim \exists w.\ x = ww\}$.

B. Here are some sample executions:

1. $x = \texttt{aaa}$

a. This string has an odd length; therefore, it must be in the language B.

b. The machine transitions to state 1 on the first transition. From there on, the machine is in state 1 if the prefix that it has read is of odd length and state 2 if it's of even length. If the machine finishes reading the string and is in state 1, it makes an epsilon move to state 3 and accepts.

c. The sequence of configurations

step	input	state	stack	
0	aaa	0	$\perp$	
1	aa	1	$\perp$	
2	a	2	$\perp$	
3	ϵ	1	$\perp$	
4	ϵ	3	ϵ,	**accept**

2. $x = \texttt{abbaabaa}$

a. This string has an even length. Let $y = \texttt{abba}$ and $z = \texttt{abaa}$. Note that $x = yz$, $|y| = |z|$, and $y \neq z$. Thus, $x \in B$.

b. The machine transitions to state 4 on the first transition. It reads until it encounters a symbol of y that doesn't match the corresponding symbol of z. In this case, this is the third symbol of each string. This pair of symbols

proves that $y \neq z$. The machine moves to state 7 (to remember the b). The remaining actions in states 7, 8, and 9 count the number of symbols before and after these special ones to verify that they are in the same positions in y and z. See Kozen, example 23.2 (page 162-163).

c. The sequence of configurations

step	input	state	stack	
0	abbaabaa	0	$\perp$	
1	abbaabaa	4	$\perp$	
2	bbaabaa	4	$\bullet\ \perp$	
3	baabaa	4	$\bullet\bullet\ \perp$	
4	aabaa	7	$\bullet\bullet\ \perp$	
5	abaa	7	$\bullet\ \perp$	
6	baa	7	$\perp$	
7	aa	8	$\bullet\ \perp$	
8	a	9	$\bullet\ \perp$	
9	ϵ	9	$\perp$	
10	ϵ	10	ϵ,	**accept**

II. An equivalent NPDA with only one state –
The main idea: use the stack to record the state

A. First attempt: replace Γ with $Q \times \Gamma$

1. Let's try the example with abbaabaa again

2. The sequence of configurations

step	input	state	stack
0	abbaabaa	$*$	$(0, \perp)$
1	abbaabaa	$*$	$(4, \perp)$
2	bbaabaa	$*$	$(4, \bullet)$(**?**, $\perp$)

where $(0, \perp)$ means stack-symbol $\perp$ and state 0, and $*$ denotes the only state of the one-state machine. Note that from step 1 to step 2, the original machine popped $\perp$ off the stack and pushed $\bullet\ \perp$ onto the stack. Now, what state should we use to label each of these stack symbols? Obviously, we should label the $\bullet$ with state 4 because that's where the machine goes. But what about (**?**, $\perp$)? We won't care what its state is until it becomes the top of stack symbol. What will the state of the original machine be when we get there?

3. To figure out how to label symbols, we'll tag each stack symbol from I.B.2.c with the step when it was pushed onto the stack. Then, when the symbol becomes the top-of-stack one, we'll know what state the machine is in, and how the symbol should have been tagged when it was originally put on the stack.

step	input	state	stack	
0	abbaabaa	0	$\perp_0$	
1	abbaabaa	4	$\perp_1$	
2	bbaabaa	4	$\bullet_2\ \perp_2$	
3	baabaa	4	$\bullet_3\bullet_3\ \perp_2$	
4	aabaa	7	$\bullet_4\bullet_3\ \perp_2$	
5	abaa	7	$\bullet_3\ \perp_2$	
6	baa	7	$\perp_2$	
7	aa	8	$\bullet_7\ \perp_7$	
8	a	9	$\bullet_8\ \perp_7$	
9	ϵ	9	$\perp_7$	
10	ϵ	10	ϵ,	**accept**

Now we see that that the $\perp$ that was pushed onto the stack in the transition from step 1 to step 2 will be exposed on the top of the stack at step 6 when the original machine is in state 7. Thus we should mark it with state 7.
How does the NPDA know that it should choose 7 as the marker?
We'll use non-determinism. I'll fill-in the details below.

4. Continuing with the tagged version

step	input	state	stack	
0	abbaabaa	$*$	$(0, \perp)$	
1	abbaabaa	$*$	$(4, \perp)$	
2	bbaabaa	$*$	$(4, \bullet)(7, \perp)$	
3	baabaa	$*$	$(4, \bullet)(7, \bullet)(7, \perp)$	
4	aabaa	$*$	$(7, \bullet)(7, \bullet)(7, \perp)$	
5	abaa	$*$	$(7, \bullet)(7, \perp)$	
6	baa	$*$	$(7, \perp)$	
7	aa	$*$	$(8, \bullet)(9, \perp)$	
8	a	$*$	$(9, \bullet)(9, \perp)$	
9	ϵ	$*$	$(9, \perp)$	
10	ϵ	$*$	ϵ,	**accept**

5. How can we do this with non-determinism?

a. Let δ be the transition relation of the original machine, and δ' be the relation for the modified machine.

b. For each transition $((p, c, A), (q, B_1 B_2 \dots B_k)) \in \delta$, we could include the transitions:

$$((*, c, (A, p)), (q_1, B_1)(q_2, B_2) \dots (q_k, B_k))$$

in δ' for all possible combinations of $q_1, q_2, \dots q_k \in Q$.
But, this would let the machine transition to *any* state of the original machine for any transition. Not quite what we want.

c. We can start to solve this problem if we require that $q_1 = q$ in the construction above. However, we still have no guarantee that $q_2, \dots q_k$ correspond to states of the original machine.

B. The solution – keep track of two states in each stack symbol:

- The state that the original machine will be in when this symbol becomes the top-of-stack symbol.
- The state that the original machine will be in when the symbol below this one becomes the top-of-stack symbol.

I'll explain how to do this in just a moment.

1. But, first, let's try it with our ongoing example.

a. Replace Γ with $Q \times \Gamma \times Q$
where a stack symbol of (pAq) means that the original machine is in state p with A on the top-of-the-stack, and the original machine will be in state q when the symbol below this one is exposed.

b. Continuing with the tagged version

step	input	state	stack	
0	abbaabaa	$*$	$(0, \perp, 10)$	
1	abbaabaa	$*$	$(4, \perp, 10)$	
2	bbaabaa	$*$	$(4, \bullet, 7)(7, \perp, 10)$	
3	baabaa	$*$	$(4, \bullet, 7)(7, \bullet, 7)(7, \perp, 10)$	
4	aabaa	$*$	$(7, \bullet, 7)(7, \bullet, 7)(7, \perp, 10)$	
5	abaa	$*$	$(7, \bullet, 7)(7, \perp, 10)$	
6	baa	$*$	$(7, \perp, 10)$	
7	aa	$*$	$(8, \bullet, 9)(9, \perp, 10)$	
8	a	$*$	$(9, \bullet, 9)(9, \perp, 10)$	
9	ϵ	$*$	$(9, \perp, 10)$	
10	ϵ	$*$	ϵ,	**accept**

C. The transition relation for the one-state NPDA

1. Let $M = (Q, \Sigma, \Gamma, \delta, s, \perp, F)$ be an NPDA. We can assume that $F = \{t\}$, (i.e. M has a single final state), and that M empties its stack when it enters state t. Let $M' = (\{*\}, \Sigma, Q \times \Gamma \times Q, \delta', *, (s, \perp, t), \emptyset)$ be an NPDA with one state, $*$, that accepts by empty stack. All we have to do is construct δ', and show that the construction is correct.

2. Constructing δ':

a. $(p, c, A) \rightarrow (q, \epsilon) \in \delta$. This means that M pops A off the stack if it's in state p and will go to state q. The corresponding top-of-stack symbol for M' is (p, A, q) and we can pop it off the stack if the next input symbol is a c. In other words

$$(*, c, (p, A, q)) \rightarrow (*, \epsilon) \quad \in \quad \delta'$$

Note that we will construct δ' to ensure that whatever symbol is uncovered when we pop (p, A, q) off of the stack is of the form (q, B, r) for some $B \in \Gamma$ and some $r \in Q$ (or $q = t$ and the stack will be empty).

b. $(p, c, A) \rightarrow (q, B) \in \delta$. This means that M reads c, replaces A with B, and moves to state q. Note that whatever stack symbol was under A will be under B. Thus,

$$(*, c, (p, A, r)) \rightarrow (*, (q, B, r)) \quad \in \quad \delta'$$

In other words, we go from saying that the machine will reach state r when it's finished the derivation associated with A to saying that it will reach state r when it finishes the derivation associated with B.

c. $(p, c, A) \rightarrow (q_0, B_1, B_2, \ldots B_k) \in \delta$. Now, M' uses non-determinism to "guess" what state M will be in at the end of processing each of the B_i. Let's say that M' had the top-of-stack symbol (p, A, q_k); then we know that M will be in state q_0 after this transition. Thus, we'll translate B_1 to (q_0, B_1, q_1) for some $q_1 \in Q$. But this means that M must be in q_1 when B_2 is on the top of the stack, so, we translate B_2 to (q_1, B_2, q_2) for some $q_2 \in Q$, and so on. Finally, we note that when the derivation for B_k completes, we will expose whatever stack symbol was under (q, A, q_k). Thus, we translate B_k to (q_{k-1}, B_k, q_k), again for some $q_{k-1} \in Q$.
How does M' decide what $q_1, q_2, \ldots q_{k-1}$ should be? It makes a non-deterministic choice. We include all possible states from Q as choices for these intermediate states. We conclude,

$$\forall q_1, q_2, \ldots q_{k-1} \in Q$$
$$(*, c, (p, A, q_k)) \rightarrow (*, (q_0, B_1, q_1), (q_1, B_2, q_2), \ldots (q_{k-1}, B_{k-1}, q_k)) \quad \in \quad \delta'$$

D. Applying this to our ongoing example

1. We need to modify the machine to have a single accepting state. This is easy. States 3 and 10 are equivalent. We'll replace the arc from sate 1 to state 3 with an arc to state 10.

2. Transitions of M that push zero or one symbols onto the stack are straightforward. We get:

$$
\begin{array}{rcll}
\delta'_{01} = \{ & (*, \mathsf{a}, (0, \perp, 10)) & \rightarrow & (*, (1, \perp, 10)) \\
& (*, \mathsf{b}, (0, \perp, 10)) & \rightarrow & (*, (1, \perp, 10)) \\
& (*, \epsilon, (0, \perp, 10)) & \rightarrow & (*, (4, \perp, 10)) \\
& (*, \mathsf{a}, (1, \perp, 10)) & \rightarrow & (*, (2, \perp, 10)) \\
& (*, \mathsf{b}, (1, \perp, 10)) & \rightarrow & (*, (2, \perp, 10)) \\
& (*, \epsilon, (1, \perp, 10)) & \rightarrow & \epsilon, \quad \textbf{accept} \\
& (*, \mathsf{a}, (2, \perp, 10)) & \rightarrow & (*, (1, \perp, 10)) \\
& (*, \mathsf{b}, (2, \perp, 10)) & \rightarrow & (*, (1, \perp, 10)) \\
& (*, \mathsf{a}, (4, \perp, 10)) & \rightarrow & (*, (5, \perp, 10)) \\
& (*, \mathsf{a}, (4, \bullet, 5)) & \rightarrow & (*, (5, \bullet, 5)) \\
& (*, \mathsf{b}, (4, \perp, 10)) & \rightarrow & (*, (7, \perp, 10)) \\
& (*, \mathsf{b}, (4, \bullet, 5)) & \rightarrow & (*, (7, \bullet, 5)) \\
& (*, \mathsf{a}, (5, \bullet, 5)) & \rightarrow & \epsilon \\
& (*, \mathsf{b}, (5, \perp, 10)) & \rightarrow & (*, (9, \perp, 10)) \\
& (*, \mathsf{b}, (5, \bullet, 5)) & \rightarrow & \epsilon \\
& (*, \mathsf{b}, (6, \bullet, 9)) & \rightarrow & (*, (9, \bullet, 9)) \\
& (*, \mathsf{a}, (7, \perp, 10)) & \rightarrow & (*, (9, \perp, 10)) \\
& (*, \mathsf{b}, (7, \bullet, 5)) & \rightarrow & \epsilon \\
& (*, \mathsf{a}, (7, \bullet, 5)) & \rightarrow & \epsilon \\
& (*, \mathsf{a}, (8, \bullet, 9)) & \rightarrow & (*, (9, \bullet, 9)) \\
& (*, \epsilon, (9, \perp, 10)) & \rightarrow & \epsilon \\
& (*, \mathsf{a}, (9, \bullet, 9)) & \rightarrow & \epsilon \\
& (*, \mathsf{b}, (9, \bullet, 9)) & \rightarrow & \epsilon, \quad \textbf{accept} \\
\}
\end{array}
$$

3. Now, let's look at transitions that push two symbols onto the stack.

a. Consider the transition of the original machine: $(4, \mathsf{a}, \perp) \rightarrow (4, \bullet\ \perp)$. If the machine can accept on the input string, it will eventually transition to either state 5 or state 7, and the $\bullet$ that is pushed on the stack now will get popped of then. Thus we will include the transitions:

$$
\begin{array}{rcl}
(*, \mathsf{a}, (4, \perp, 10)) & \rightarrow & (*, (4, \bullet, 5)(5, \perp\ 10)) \\
(*, \mathsf{a}, (4, \perp, 10)) & \rightarrow & (*, (4, \bullet, 7)(7, \perp\ 10))
\end{array}
$$

in δ'. Note that we don't have to write all states in Q for the connector between the two stack symbols – we can determine from the operation of the machine that states 5 and 7 are the only ones that are needed. This reduces the amount of stuff we have to put into δ' and makes our solution shorter. On the other hand, including all such intermediate states, for example,

$$
(*, \mathsf{a}, (4, \perp, 10)) \quad \rightarrow \quad (*, (4, \bullet, 0)(0, \perp\ 10))
$$

also produces a correct machine. This is the approach taken in Kozen to show that every NPDA corresponds to a CFL. It's simpler to describe because it doesn't depend on the details of the particular machine.

We'll write δ'_2 to denote the transitions in δ' that push two symbols onto the stack. Here they are:

$$
\begin{array}{rcll}
\delta'_2 & = & \{ & (*, \mathsf{a}, (4, \perp, 10)) \rightarrow (*, (4, \bullet, 5)(5, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (4, \perp, 10)) \rightarrow (*, (4, \bullet, 7)(7, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (4, \bullet, 5)) \rightarrow (*, (4, \bullet, 5)(5, \bullet 5)) \\
 & & & (*, \mathsf{a}, (4, \bullet, 5)) \rightarrow (*, (4, \bullet, 5)(5, \perp\ 10)) \\
 & & & (*, \mathsf{b}, (4, \perp, 10)) \rightarrow (*, (4, \bullet, 5)(5, \perp\ 10)) \\
 & & & (*, \mathsf{b}, (4, \perp, 10)) \rightarrow (*, (4, \bullet, 7)(7, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (4, \bullet, 7)) \rightarrow (*, (4, \bullet, 7)(7, \bullet 5)) \\
 & & & (*, \mathsf{a}, (4, \bullet, 7)) \rightarrow (*, (4, \bullet, 7)(7, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (5, \perp, 10)) \rightarrow (*, (6, \bullet, 9)(9, \perp\ 10)) \\
 & & & (*, \mathsf{b}, (5, \perp, 10)) \rightarrow (*, (6, \bullet, 9)(9, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (6, \bullet, 9)) \rightarrow (*, (6, \bullet, 9)(9, \bullet 9)) \\
 & & & (*, \mathsf{b}, (6, \bullet, 9)) \rightarrow (*, (6, \bullet, 9)(9, \bullet 9)) \\
 & & & (*, \mathsf{a}, (7, \perp, 10)) \rightarrow (*, (8, \bullet, 9)(9, \perp\ 10)) \\
 & & & (*, \mathsf{b}, (7, \perp, 10)) \rightarrow (*, (8, \bullet, 9)(9, \perp\ 10)) \\
 & & & (*, \mathsf{a}, (8, \bullet, 9)) \rightarrow (*, (8, \bullet, 9)(9, \bullet 9)) \\
 & & & (*, \mathsf{b}, (8, \bullet, 9)) \rightarrow (*, (8, \bullet, 9)(9, \bullet 9)) \\
 & & \} &
\end{array}
$$

4. Combining these together we get:

$$\delta' = \delta'_{01} \cup \delta'_2$$

5. Here's the sequence of configurations that this machine goes through to accept the string `abbaabaa`:

step	input	state	stack	transition
0	abbaabaa	$*$	$(0, \perp, 10)$	$(*, \epsilon, (0, \perp, 10)) \rightarrow (*, (4, \perp, 10))$
1	abbaabaa	$*$	$(4, \perp, 10)$	$(*, \mathsf{a}, (4, \perp, 10)) \rightarrow (*, (4, \bullet, 7)(7, \perp, 10)$
2	bbaabaa	$*$	$(4, \bullet, 7)(7, \perp, 10)$	$(*, \mathsf{b}, (4, \bullet, 7)) \rightarrow (*, (4, \bullet, 7)(7, \bullet, 7))$
3	baabaa	$*$	$(4, \bullet, 7)(7, \bullet, 7)(7, \perp, 10)$	$(*, \mathsf{b}, (4, \bullet, 7)) \rightarrow (*, (7, \bullet, 7))$
4	aabaa	$*$	$(7, \bullet, 7)(7, \bullet, 7)(7, \perp, 10)$	$(*, \mathsf{a}, (7, \bullet, 7)) \rightarrow (*, \epsilon)$
5	abaa	$*$	$(7, \bullet, 7)(7, \perp, 10)$	$(*, \mathsf{a}, (7, \bullet, 7)) \rightarrow (*, \epsilon)$
6	baa	$*$	$(7, \perp, 10)$	$(*, \mathsf{b}, (7, \perp, 10)) \rightarrow (*, (8, \bullet, 9)(9, \perp, 10))$
7	aa	$*$	$(8, \bullet, 9)(9, \perp, 10)$	$(*, \mathsf{a}, (8, \bullet, 9)) \rightarrow (*, (9, \bullet, 9))$
8	a	$*$	$(9, \bullet, 9)(9, \perp, 10)$	$(*, \mathsf{a}, (9, \bullet, 9)) \rightarrow (*, \epsilon)$
9	ϵ	$*$	$(9, \perp, 10)$	$(*, \mathsf{a}, (9, \perp, 10)) \rightarrow (*, \epsilon)$
10	ϵ	$*$	ϵ	**accept**

The entry in the "transition" column shows which transition in δ' is taken to get to the next configuration.

E. Proving the construction correct.
See Kozen lecture 25.