Lecture 17

Homework #18: 3.3.1, 3.3.2, 3.3.3. Hand in 3.3.2 b and 3.3.3.
Note: In my copy of the text, 3.3.1 has two typos. $\in$ should be $\notin$ and vice versa.
For 3.3.3, give the construction and explain why it is correct.

Just as finite automata are recognizers of regular languages, we can construct recognizers of context-free languages.

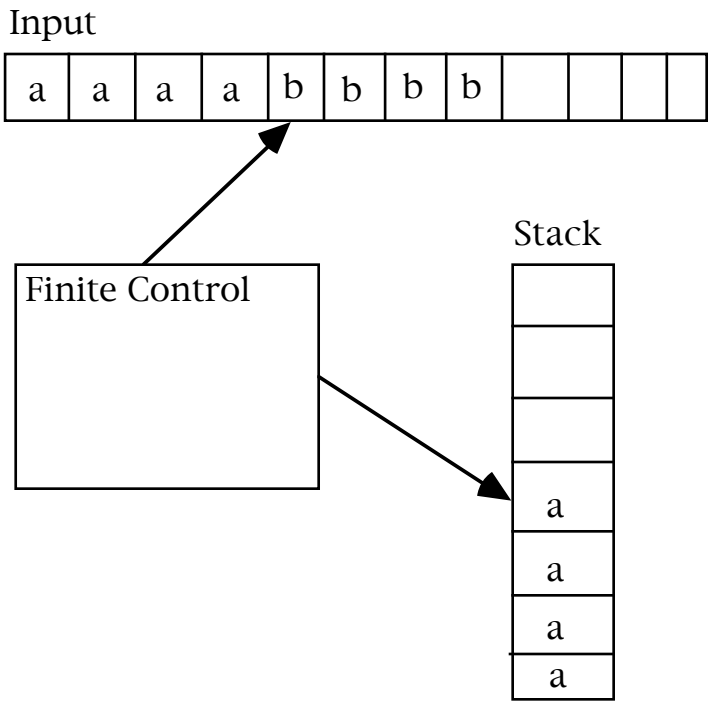
Clearly, finite automata are not adequate.
So what do we need to add to finite automata to make them sufficiently powerful to handle CFLs?

$\Rightarrow$ **memory**

Example. consider $a^n b^n$

The automaton must have a way to remember a^n (or at least n), so that it can check the number of b's.

This can be accomplished by the following extended automaton:



Every time you see an a in the input, place a copy on the stack; once you begin to encounter b's, check them off against the a's.

A **Pushdown Automaton (PDA)** is:

- an automaton with a stack;

  In general, a PDA operates as follows:
  in a given state, consider the input and the top of the stack.
  based upon what you see, "eat" the input, do something with the stack (push and/or pop), and change state.

- the languages accepted by PDAs are precisely the Context Free Languages. (we'll prove this later)

Let's consider why this type of machine makes sense, and specifically consider why a stack is an appropriate structure for memory.

We'll do this by looking at a particular context free language, $\{ww^R \mid w \in \{a, b\}^*\}$. This language is generated by the following CFG $G = (V, \Sigma, R, S)$, where R contains the rules

$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow e$

Say that we have the string abbbba, and we want to build a machine to recognize it as part of the language:
Upon seeing the first a, we want to remember it for later, in order to check it against what we see at the end of the string. Next we need to remember the symbol b, so that we can check it as well. Finally, we want to remember the next b. Once we're ready to check against the end of the string, we need to have the symbols to check in the order that they will appear in the string, i.e., "bba". That means that the first symbol we encountered, "a", must be the last item checked. A stack is a perfect data structure for holding data in such an order.

While this gives us a general understanding of the workings of Pushdown Automata, there's more.

Formally:

Def. A pushdown automaton is a sextuple
$M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

K is a finite set of states
Σ is an alphabet (the input symbols)
Γ is an alphabet (the stack symbols)
$s \in K$ is the start state
$F \subseteq K$ is a set of final states

Δ, the transition relation, is a finite subset of

$$(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$$

If $((p,u,\beta),(q,\gamma)) \in \Delta$:
in state p, with β at the top of the stack,
read u from the input, replace β with γ,
and go into state q.

Note that the book is seemingly more restrictive: the transition relation is a finite subset of
$$(K \times (\Sigma \cup \{e\}) \times \Gamma^*) \times (K \times \Gamma^*)$$
Why doesn't it matter which definition we follow?

Since we're working with a stack, we'll use the usual terminology of pushing and popping:

((p,u,e),(q,a))	pushes a on the stack
((p,u,a),(q,e))	pops a from the stack

Looking at "e" at the top of the stack does not mean that the stack is empty! It simply means that the transition is ignoring the top of the stack. (It might be empty; but it might not be empty.)

Note that a **PDA is nondeterministic**!
But this makes sense if you think about CFGs.
Consider ww^r
A PDA can't count the number of symbols to determine where the halfway point will be. It can only guess.

What if you try to simulate all possibilities in parallel (a la DFAs = NFAs)? What happens to the stack?

Now, for the obligatory terminology, notation, and stuff:

A **configuration** is an element of $K \times \Sigma^* \times \Gamma^*$

(in general, a configuration is a concise way of representing the "state" of the machine)

K = state
Σ^* = remaining input
Γ^* = the stack, ***top to bottom***

so if we have the configuration (q, aa, abb), our PDA is in state q, with aa remaining as input. The stack holds the symbols abb, with a at the top of the stack and b at the bottom. Each cell in the stack can hold only one symbol from the stack alphabet.

$\vdash$ is **yields in 1 step**

$(p, ux, \beta\alpha) \vdash (q, x, \gamma\alpha)$

when $((p,u,\beta), (q,\gamma)) \in \Delta$.

[If it wasn't obvious already, we're using lowercase Greek symbols for the stack to distinguish them from the input alphabet]

$\vdash^*$ is the reflexive transitive closure of $\vdash$.

M **accepts** $s \in \Sigma^*$ iff
$(s, w, e) \vdash^* (f, e, e), f \in F$.

if c_i denotes a configuration, then
$c_0 \vdash c_1 \vdash c_2$ etc denotes a **computation**

L(M) is the **language accepted by** M.

And now for some ***examples***

(1) $\{a^n c b^n, n \geq 0\}$

Let M = (K, Σ, Γ, Δ, s, F), where

Δ contains ((s,a,e),(s,a))
((s,c,e),(f,e))
((f,b,a),(f,e))

K = {s, f}
F = {f}
Σ = {a, b, c}
Γ = {a}

Do a mini-trace.

(2) $\{a^n b^n, n \geq 1\}$

Let M = (K, Σ, Γ, Δ, s, F), where

Δ contains ((s,a,e),(s,a))
((s,b,a),(f,e))
((f,b,a),(f,e))

Do a mini-trace.

(3) $\{w \in \{a,b\}^*: w = w^R\}$

Let M = (K, Σ, Γ, Δ, s, F), where

Δ contains ((s,a,e),(s,a))
((s,b,e),(s,b))

((s,a,e),(f,e))
((s,b,e),(f,e)) **nondeterministically select**
((s,e,e),(f,e)) **the center of the string**

((f,a,a),(f,e))
((f,b,b),(f,e))

K = {s,f}
F = {f}
Σ = {a, b}
Γ = {a, b}

(4) $\{a^n b^m, m \geq n\}$

Let M = (K, Σ, Γ, Δ, s, F), where

Δ contains ((s,e,e),(p,c)) adding a bottom of stack marker gives a sense of determinism

((p,a,e),(p,a))
((p,b,a),(q,e))
((p,b,c),(q,c))
((q,b,a),(q,e))
((q,b,c),(q,c))
((q,e,c),(f,e))

K = {s,p,q,f}
F = {f}
Σ = {a, b}
Γ = {a, c}