1 Push-down Automata and Context-Free Languages

Lemma 1.1 (3.4.1) *The class of languages recognized by push-down automata is the same as the class of context-free languages.*

This result is interesting because sometimes it is easier to see how to construct a push-down automaton than a context-free language. Also, pda are helpful for parsing cfl.

Given a context-free grammar $G = (V, \Sigma, R, S)$, one can construct a push-down automaton M such that $L(M) = L(G)$ as follows:

$M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$ where Δ has the rules

(1)	$((p, \epsilon, \epsilon), (q, S))$	
(2)	$((q, \epsilon, A), (q, x))$	if $A \to x$ is in R
		(do leftmost derivation on the stack)
(3)	$((q, a, a), (q, \epsilon))$	for each $a \in \Sigma$
		(remove matched symbols)

Here is an example of this construction. Consider the grammar G given by the following rules:

$$S \to aSb$$

$$S \to \epsilon$$

Then the push-down automaton M has the following transitions:

(1)	$((p, \epsilon, \epsilon), (q, S))$	
(2)	$((q, \epsilon, S), (q, aSb))$	from $S \to aSb$
	$((q, \epsilon, S), (q, \epsilon))$	from $S \to \epsilon$
(3)	$((q, a, a), (q, \epsilon))$	because $a \in \Sigma$
	$((q, b, b), (q, \epsilon))$	because $b \in \Sigma$

We note that $aabb \in L(G)$ by the derivation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

.

Corresponding to this there is the following accepting computation of the push-down automaton M:

$$(p, aabb, \epsilon) \vdash (q, aabb, S) \vdash (q, aabb, aSb) \vdash (q, abb, Sb) \vdash (q, abb, aSbb) \vdash (q, bb, Sbb)$$

$$\vdash (q, bb, bb) \vdash (q, b, b) \vdash (q, \epsilon, \epsilon)$$

.

For this derivation, for all configurations in the state q,

- the part of the input already read,
- followed by the stack,
- is always derivable from the start symbol,

so that one can always recover a cfg derivation from a pda computation. In this case we obtain the following sequence from the pda computation:

$$S, aSb, a : Sb, a : aSbb, aa : Sbb, aa : bb, aab : b, aabb : \epsilon$$

where the colon separates the part already read from the stack.
In general,

- it is possible to convert any context-free derivation in G into a computation on the pda M in this way, and back,
- so that $L(M) = L(G)$.

The same construction works for any context-free grammar G, showing that every context-free language can be recognized by some push-down automaton.

The reverse direction of the proof, going from a push-down automaton to a context-free grammar, is much harder, and the main part of the argument is given in Lemma 3.4.2 of the text.

Lemma 1.2 (3.4.2) *If a language is recognized by a push-down automaton, then it is a context-free language.*

The basic idea of the proof is to construct a context-free grammar G from a push-down automaton M, satisfying $L(G) = L(M)$.

- The nonterminals of G are triples of the form $\langle q, A, p\rangle$ where q and p are states of M and A is a stack symbol or ϵ.
- Then G has productions guaranteeing that $\langle q, A, p\rangle \Rightarrow^* x$ if and only if $(q, x, A) \vdash_M^* (p, e, e)$ where (q, x, A) and (p, e, e) are configurations of M and $\vdash$ is the "yields" relation.
- Intuitively, the nonterminal $\langle q, A, p\rangle$ derives x if the push-down automata can remove A from the stack while reading x and going from state q to state p.

The context-free grammar G has

- productions of the form $\langle q, B, p\rangle \to a\langle r, C, p\rangle$ for all states $p \in K$ and for each transition of M of the form $((q, a, B), (r, C))$, and
- productions of the form $\langle q, B, p\rangle \to a\langle r, C_1, p'\rangle\langle p', C_2, p\rangle$ for all $p, p' \in K$ and for each transition of M of the form $((q, a, B), (r, C_1C_2))$.

Intuitively, to remove B from the stack, one can first put C_1C_2 on the stack, then remove C_1, then remove C_2, or one can put C on the stack and then remove it.

This proof is more complicated than lemma 3.4.1 because context-free grammars are simple but push-down automata are complex. It is easier to simulate something simple by something complex, as in lemma 3.4.1, but harder to simulate something complex by something simple, as in lemma 3.4.2.