CSE 322
Intro to Formal Models in CS
Shift/Reduce Parsing
An Alternate Proof of Lemma 2.21

W. L. Ruzzo 14 Nov 10

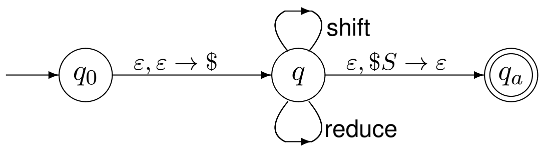
This handout sketches an alternate proof of Lemma 2.21, via a "bottom-up" or "shift/reduce" method, used in many compilers and other text-processing applications.

The book's proof of Lemma 2.21 creates a "top-down" parser: build a parse tree for a given input w by starting from the root, S, of the tree and working downward, nondeterministically guessing rules to apply at each step. One slightly tricky bit is that the tree traversal order and data structure need to be carefully chosen so that the PDA's pushdown is sufficient. It turns out that a pre-order tree traversal, corresponding to a leftmost derivation of w, suffices. The bottom-up construction below instead builds the tree from the leaves up to the root, again liberally using nondeterminism to choose the right rules to use. Again, some care is needed to do all this on a PDA, and it turns out that the right choice is a post-order traversal, which corresponds to a rightmost derivation of w (but in reverse order; see the example below).

Lemma 2.21: If a language is context-free, then some PDA recognizes it.

SHORTHAND: It's convenient to assume the PDA can have transitions where the "stack" part looks like $bcd \to E$, meaning "if bcd are the top three symbols on the stack, d topmost, pop them and push E." It's straightforward to simulate such a move by the standard model by adding a few intermediate states, similar to figure 2.23, popping one symbol at a time. These details are omitted.

CONSTRUCTION: Let $G = (V, \Sigma, R, S)$ be a CFG generating the language. Construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ as follows. Q has 3 states $\{q_0, q, q_a\}$ (plus the intermediate states implied by the shorthand defined above). $F = \{q_a\}$. $\Gamma = V \cup \Sigma \cup \{\$\}$. And δ is:



q_0
$\varepsilon, \varepsilon \to \$$
q
shift
reduce
$\varepsilon, \$S \to \varepsilon$
q_a

where "shift" is the set of transitions $\delta(q, a, \epsilon) = (q, a)$ for all $a \in \Sigma$, ("$\mathsf{a}, \varepsilon \to \mathsf{a}$" in the diagram) and "reduce" is the set of transitions $\delta(q, \epsilon, \beta) = (q, A)$ for all rules $A \to \beta$ in G ("$\varepsilon, \beta \to A$"). In words, M can shift any input letter onto its stack, and whenever it has the right-hand-side of some rule on the top of its stack, it can "reduce" it to (i.e., replace it by) the variable on the left-hand-side of that rule.

NOTATION: For any $q \in Q, \gamma \in \Gamma^*$, and $w \in \Sigma^*$, let $[q, \gamma, w]$ denote a *configuration* of the PDA, i.e., a specification of its current state, stack contents, and the remaining (unread) input. The stack is the *middle* component of the triple, and the top of stack is the *right*most symbol of γ. Furthermore, $[s, \gamma, w] \vdash^k [s', \gamma', w']$ means the PDA can move from the first configuration to the second in exactly k steps. (Assume $k = 1$ if omitted.) Finally, for $\alpha, \beta \in (V \cup \Sigma)^*$, the notation $\alpha \Rightarrow_R \beta$ ($\alpha \Rightarrow_R^k \beta$) means that β can be derived from α in G by a one-step (or k-step, respectively) rightmost derivation, i.e., one in which the rightmost variable is rewritten at each step.

See example on the next page for an illustration of the construction and execution of the parser on a sample grammar/input.

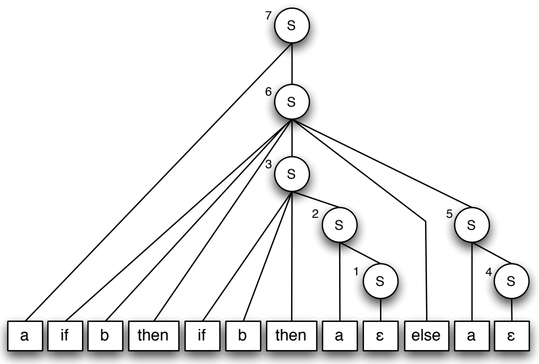
Example: (Note the correspondence between the postorder numbering of the internal nodes of the tree and the like-numbered steps in the derivation and the reduce moves in the PDA computation.)

G:

$$\begin{aligned}
S &\rightarrow \text{a } S\\
S &\rightarrow \text{if b then } S\\
S &\rightarrow \text{if b then } S \text{ else } S\\
S &\rightarrow \varepsilon
\end{aligned}$$

A Rightmost derivation:

$$\begin{aligned}
S &\Rightarrow_R \text{a } S && (7)\\
&\Rightarrow_R \text{a if b then } S \text{ else } S && (6)\\
&\Rightarrow_R \text{a if b then } S \text{ else a } S && (5)\\
&\Rightarrow_R \text{a if b then } S \text{ else a} && (4)\\
&\Rightarrow_R \text{a if b then if b then } S \text{ else a} && (3)\\
&\Rightarrow_R \text{a if b then if b then a } S \text{ else a} && (2)\\
&\Rightarrow_R \text{a if b then if b then a else a} && (1)
\end{aligned}$$



Accepting PDA Computation:

[q_0,	ε,	a if b then if b then a else a]	$\vdash$	
[q,	\$,	a if b then if b then a else a]	$\vdash$	
[q,	\$ a,	if b then if b then a else a]	$\vdash$	
[q,	\$ a if,	b then if b then a else a]	$\vdash$	
	⋮ 6 more shifts			
[q,	\$ a if b then if b then a,	else a]	$\vdash$	(1)
[q,	\$ a if b then if b then a S,	else a]	$\vdash$	(2)
[q,	\$ a if b then if b then S,	else a]	$\vdash$	(3)
[q,	\$ a if b then S,	else a]	$\vdash$	
[q,	\$ a if b then S else,	a]	$\vdash$	
[q,	\$ a if b then S else a,	ε]	$\vdash$	(4)
[q,	\$ a if b then S else a S,	ε]	$\vdash$	(5)
[q,	\$ a if b then S else S,	ε]	$\vdash$	(6)
[q,	\$ a S,	ε]	$\vdash$	(7)
[q,	\$ S,	ε]	$\vdash$	
[q_a,	ε,	ε]		

The correctness of the construction is captured by the following assertion:
CLAIM: For all $\gamma \in (V \cup \Sigma)^*$ and all $w \in \Sigma^*$,

$$\gamma \Rightarrow_R^k w \text{ if and only if } [q, \epsilon, w] \vdash^{k+|w|} [q, \gamma, \epsilon].$$

Before proving the claim, look at the example again, and note that as a corollary,

$$S \Rightarrow_R^k w \text{ if and only if } [q, \epsilon, w] \vdash^{k+|w|} [q, S, \epsilon],$$

from which it's easy to see that $L(M) = L(G)$.

Proof of Claim: First, note that if the PDA moves from q to q reading w, then it must make exactly $|w|$ "shift" moves, since they're the only ones that read inputs; all other moves are "reduce" moves.

We prove the "only if" direction, by induction on k.

For the basis case, $k = 0$, we simply note that $\gamma \Rightarrow_R^0 w$ only if $\gamma = w$, which implies that $[q, \epsilon, w] \vdash^{|w|} [q, \gamma, \epsilon]$, via $|w|$ consecutive shift moves.

For the induction step, suppose the claim is true for some $k \geq 0$, and suppose that $\gamma \Rightarrow_R^{k+1} w$. This derivation must have a first step, say using rule $A \to \beta$. So, there must be some $\alpha \in (V \cup \Sigma)^*$ and some $x, y \in \Sigma^*$ such that $\gamma = \alpha A y \Rightarrow_R \alpha\beta y \Rightarrow_R^k xy = w$. So $\alpha A \Rightarrow_R \alpha\beta \Rightarrow_R^k x$, and so by the induction hypothesis, $[q, \epsilon, x] \vdash^{k+|x|} [q, \alpha\beta, \epsilon]$ and $[q, \alpha\beta, \epsilon] \vdash [q, \alpha A, \epsilon]$ by one "reduce" move, so $[q, \epsilon, xy] \vdash^{k+|x|} [q, \alpha\beta, y] \vdash^1 [q, \alpha A, y] \vdash^{|y|} [q, \alpha A y, \epsilon]$. Thus $[q, \epsilon, w] \vdash^{k+1+|w|} [q, \gamma, \epsilon]$ as desired.

Proof of the converse direction (showing that $[q, \epsilon, w] \vdash^{k+1+|w|} [q, \gamma, \epsilon]$ implies $\gamma \Rightarrow_R^{k+1} w$) is similar, and is left as an exercise. Hint: consider the last "reduce" step in the PDA's computation.

QED.

Exercise: The grammar above is *ambiguous* (the famous "dangling else" ambiguity) and the example string has another rightmost derivation and corresponding parse tree, postorder numbering, and PDA computation. Find them.

Remarks: Note that the PDA built above is fundamentally nondeterministic: a shift move is always possible as long as there is any remaining input, and one or more reduce moves may also be possible in many circumstances. With a carefully designed grammar, and by being able to "peek" ahead at the next input symbol, it is often possible to tell deterministically which action to take. The CFG's for which this is possible are called LR(1) grammars, and are important for programming language design. Every language accepted by a deterministic PDA has an LR(1) grammar, but not all grammars for a given language are LR(1), and for some CFL's *no* grammar is LR(1).