LL(1) Parsing

- A Push-Down Automaton (PDA) is a finite state machine augmented with a stack.
  - In addition to changing state on input, you can also look at top of stack and push/pop/chg top element of stack. NPDA ≠ DPDAs unlike NFA = DFA.
- Can use PDAs to build parsers for CFGs
  - Can use a DPDA to parse LL(1) grammars (deterministic parser to avoid backtracking)
  - The stack contains the unprocessed part of the derivation.
- By analyzing an LL(1) grammar, we can build a Prediction Table:
  - Predict(A, x) = the set of rules we can apply when expanding an A if the next symbol is x.
  - If when building the table, we get > 1 entry in a slot, then the grammar is not LL(1).
  - If the set is ∅, then we have an error during the parse.
- Table entries get built using the First sets, and if there are A → ε rules, also the Follow sets
  - With rule A → α (where α ≠ ε), if x ∈ First(α), then rule A → α ∈ Predict(A, x).
  - Put off A → ε handling for a bit.

Example: Grammar for \{ a^n c b^n | n ≥ 0 \}

- S' → S $
- S → a S b
- S → c
- Calculate First(S') = \{a, c\}, First(S) = \{a, c, ε\}
- Expand nonterminal A by A → α if next symbol ∈ First(α).
  - S' using S' → S $ rule if next symbol is a or $ since First(S' $) = \{a, $\}.
  - S using S → a S b rule if next symbol is a, First(a S b) = \{a\}.
  - S using S → c rule if next symbol is c, since First(c) = \{c\}.
- Prediction table (number indicates rule to use: Predict(S', a) = \{ rule 1 \} = \{ S' → S $ \})

**Prediction for Grammars with ε Productions**

- Say we have S ⇒* w A x ... and a rule S → ε.
  - Should we apply it and get S ⇒* w A x... ⇒ w x ...? 
  - If w x doesn't begin any legal sentence in the language, then no, we shouldn't
  - When expanding A, if the next symbol ∈ Follow(A), applying A → ε is reasonable.
- General rules (x is next symbol)
  - If x ∈ First(α) (where α ≠ ε), then Predict(A, x) includes A → α.
If \( x \in \text{Follow}(A) \), then \( \text{Predict}(A, x) \) includes \( A \to \epsilon \).

**Example: Grammar for \( \{ a^n b^n \mid n \geq 0 \} \)**

- \( S' \to S \$
- \( S \to a S b \)
- \( S \to \epsilon \) [not \( S \to c \)]

Calculate \( \text{First}(S') = \{a, \$, \}, \text{First}(S) = \{a, \epsilon\}, \text{Follow}(S') = \emptyset, \text{Follow}(S) = \{b, \$\} \).

- Expand
  - \( S' \) using \( S' \to S \$ \) rule if next symbol is \( a \) or \( \$ \) (as in \( a^n c b^n \) example)
  - \( S \) using \( S \to a S b \) rule if next symbol is \( a \) (as in \( a^n c b^n \) example)
  - \( S \) using \( S \to \epsilon \) rule if next symbol is \( b \) or \( \$ \), since \( \text{Follow}(S) = \{b, \$\} \).

- Predict table
  
<table>
<thead>
<tr>
<th>NT</th>
<th>a</th>
<th>b</th>
<th>$</th>
</tr>
</thead>
</table>
  | \( S' \) | 1 | . | 1 | [number indicates rule to use]  
  | \( S \) | 2 | 3 | 3 |

**Trace parse of input using prediction table**

- Stack contains terminals / nonterminals we hope to find
- As we match terminal symbols from the input, we remove them.
- As an example, take the \( a^n b^n \) language (has 3 rules: (1) \( S' \to S \$ \), (2) \( S \to a S b \), and (3) \( S \to \epsilon \)).

**Trace parse of \( a a b b \$ \)**

<table>
<thead>
<tr>
<th>Stack (Top to Left)</th>
<th>Rule # or = test</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S' )</td>
<td>1</td>
<td>( a a b b $ )</td>
</tr>
<tr>
<td>( S $ )</td>
<td>2</td>
<td>( a a b b $ )</td>
</tr>
<tr>
<td>( a S b $ )</td>
<td>( a = a )</td>
<td>( a a b b $ )</td>
</tr>
<tr>
<td>( S b $ )</td>
<td>2</td>
<td>( a b b $ )</td>
</tr>
<tr>
<td>( a S b b $ )</td>
<td>( a = a )</td>
<td>( a b b $ )</td>
</tr>
<tr>
<td>( S b b $ )</td>
<td>3</td>
<td>( b b $ )</td>
</tr>
<tr>
<td>( b b $ )</td>
<td>( b = b )</td>
<td>( b b $ )</td>
</tr>
<tr>
<td>( b $ )</td>
<td>( b = b )</td>
<td>( b $ )</td>
</tr>
<tr>
<td>( $ )</td>
<td>Success!</td>
<td>( $ )</td>
</tr>
</tbody>
</table>

- If the top of the stack is a nonterminal, we want to apply a rule.
- If the top of the stack is a terminal, it must match the first remaining input symbol.
Review for Calculating First and Follow *(added after class)*

**First Sets**
- First(A) = First(α₁) ∪ First(α₂) ∪ ... where A → α₁ | α₂ | ...
- First(x β) = {x}
- First(B β) = First(B) ∪ (if B ⇒* ε then First(β) else ø)
- First(ε) = {ε}
  - (So ε ∈ First(A) iff A ⇒* ε)

**Follow Sets**
- For each rule B → ... A α, Follow(A) ⊇ First(α) (less ε),
  and if (α is ε or) α ⇒* ε then Follow(B) ⊆ Follow(A).
  - (We're looking for contexts like S ⇒* ... B β ⇒ ......A β, so anything that begins a β can follow a B and also follow an A.)

Typos in notes during lecture: I wrote ⊆ where I needed ⊇. It should've been:
- For S → a S b, Follow(S) ⊇ First(b)
- For S' → S $, Follow(S) ⊇ First($)