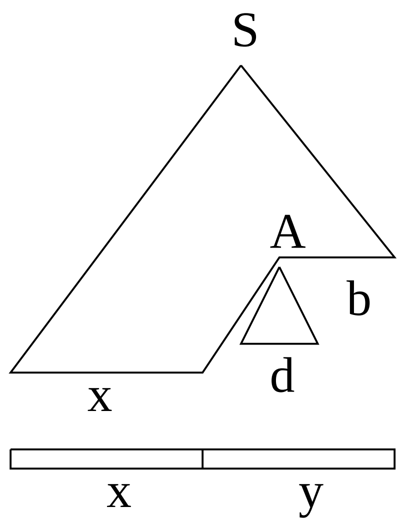
Class Information

- Second homework due on Friday, February 14, before class.

Top-Down Parsing - LL(1)



<u>Basic Idea:</u>

- The parse tree is constructed from the root, expanding **non-terminal** nodes on the tree's frontier following a left-most derivation
- The input program is read from left to right, and input tokens are read (consumed) as the program is parsed
- The next **non-terminal** symbol is replaced by one of its rules. The particular choice <u>has to be unique</u>, and uses parts of the input (partially parsed program), for instance the first **token** of the remaining input

Top-Down Parsing - LL(1) (cont.)

Example:

S ::= a S b | ϵ

How can we parse (automatically construct a left-most derivation) the input string **a a a b b b** using a PDA (push-down automaton) and only the first symbol of the remaining input?

INPUT: $\boxed{\text{a a a b b b eof}}$

Predictive Parsing

Basic idea:

> For any two productions $A ::= \alpha \mid \beta$, we would like a distinct way of choosing the correct production to expand.

For some *rhs* $\alpha \in G$, define **FIRST**(α) as the set of tokens that appear as the first symbol in some string derived from α.

That is
$x \in \text{FIRST}(\alpha)$ *iff* $\alpha \Rightarrow^* x\gamma$ for some γ, and
$\epsilon \in \text{FIRST}(\alpha)$ *iff* $\alpha \Rightarrow^* \epsilon$

For a non-terminal A, define **FOLLOW**(A) as the set of terminals that can appear immediately to the right of A in some sentential form.

Thus, a non-terminal's FOLLOW set specifies the tokens that can legally appear after it.

A terminal symbol has no FOLLOW set

$\boxed{\text{FIRST and FOLLOW sets can be constructed automatically}}$

Predictive Parsing (cont.)

Key Property:

Whenever two productions $A ::= \alpha$ and $A ::= \beta$ both appear in the grammar, we would like

- $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$, and
- if $\alpha \Rightarrow^* \epsilon$ then $FIRST(\beta) \cap FOLLOW(A) = \emptyset$
- Analogue case for $\beta \Rightarrow^* \epsilon$. Note: due to first condition, at most one of α or β can derive ϵ.

This would allow the parser to make a correct choice with a lookahead of only one symbol!

LL(1) Grammar

Define $FIRST^{+}(\delta)$ for rule $A ::= \delta$

- $FIRST(\delta)$ - $\{\epsilon\} \cup$ Follow(A), if $\epsilon \in FIRST(\delta)$
- $FIRST(\delta)$ otherwise

A grammar is LL(1) iff

$(A ::= \alpha$ and $A ::= \beta)$ implies

$$FIRST^{+}(\alpha) \cap FIRST^{+}(\beta) = \emptyset$$

Back to Our Example

S ::= a S b | ϵ

$FIRST$(aSb) = {a}

$FIRST(\epsilon) = \{\epsilon\}$

$FOLLOW$(S) = {eof, b}

$FIRST^{+}$(aSb) = {a}

$FIRST^{+}(\epsilon) = (FIRST(\epsilon)$ - { ϵ}) $\cup$ FOLLOW(S) =
{eof, b}

Is the grammar LL(1)?

Table-Driven LL(1) Parser

LL(1) parse table

Example:

S ::= a S b | ϵ

	a	b	eof	other
S	aSb	ϵ	ϵ	error

How to parse input **a a a b b b** ?

Table-driven predictive parsing algorithm

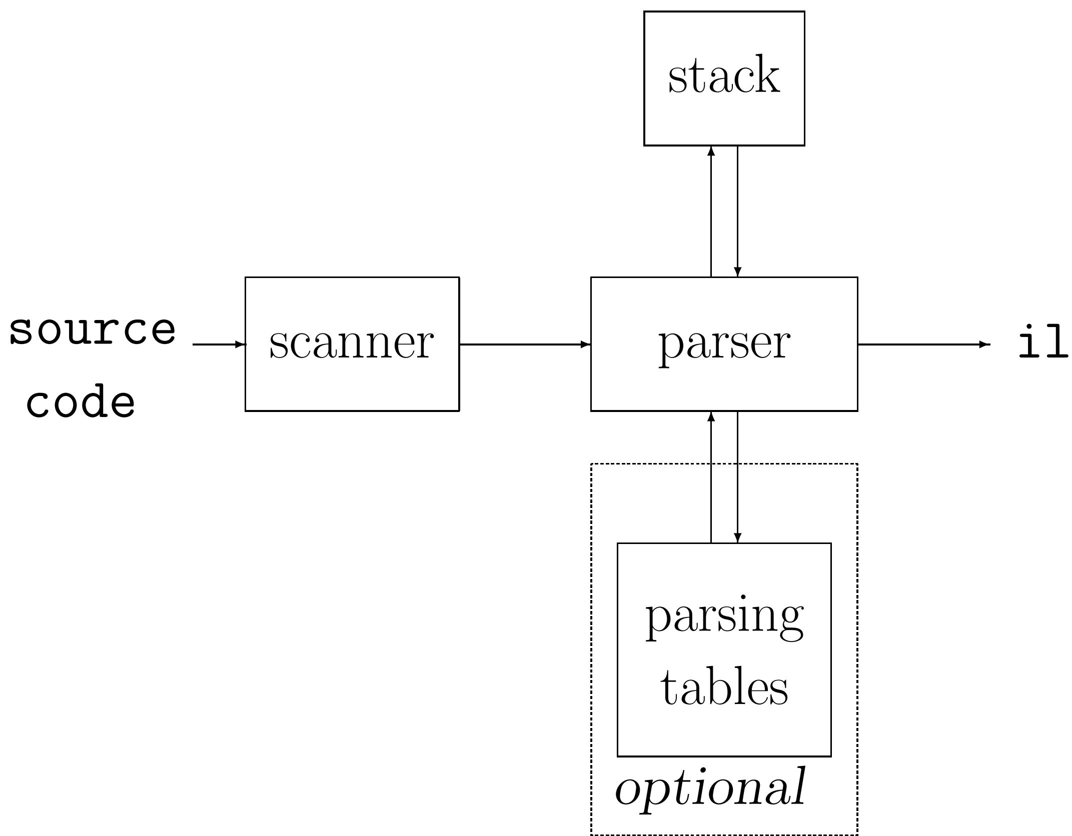
Input: a string w and a parsing table M for G

```
push eof
push Start Symbol
token ← next_token()
X ← top-of-stack
repeat
   if X is a terminal then
      if X = token then
         pop X
         token ← next_token()
      else error()
   else /* X is a non-terminal */
      if M[X,token] = X → Y1 Y2 ··· Yk then
         pop X
         push Yk, Yk−1, ···, Y1
      else error()
   X ← top-of-stack
until X = eof
if token ≠ eof then error()
```

See also Aho, Lam, Sethi, and Ullman, Figure 4.20, page 227

Predictive Parsing

Now, a predictive parser looks like:

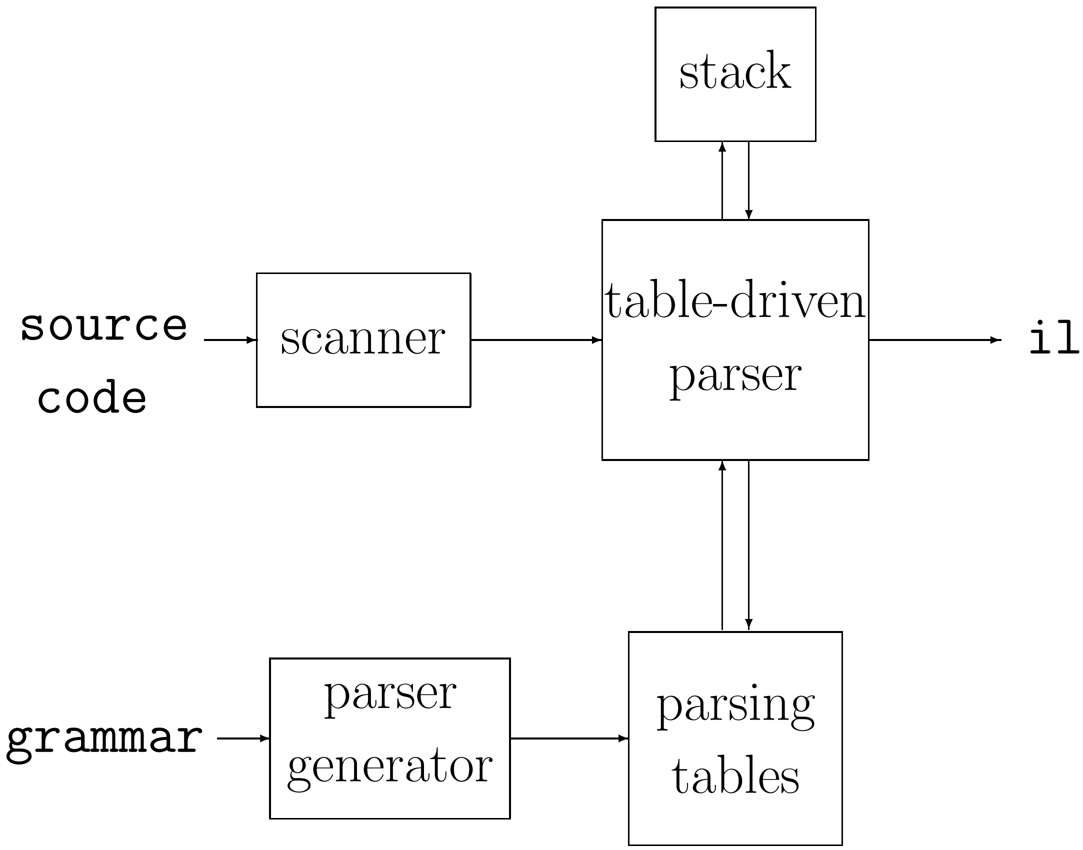


Rather than writing code, we build tables.

Building tables can be automated!

Generating a Table-Driven Parser

A parser generator system often looks like:



Next Lecture

Things to do:

Start programming in C. Check out the web for tutorials.

Next time:

- Recursive-descent parsers
- Syntax-directed translation schemes
- Imperative programming languages
- Pointers, basic types etc. in C
- Read Scott 5.1 - 5.3 (some background - chapter on CD)