RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

CS415 Compilers
Syntax Analysis
Top-Down Parsing

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University

Announcements

- **Midterm on Wednesday, March 8**
  - closed book, closed notes
  - covers everything up to (and including) top-down parsing
  - No recitation on Tuesday, March 8
  - Q&A session second half of Thursday 3/3 lecture – come prepared!
- Project 1 deadline extended, code Feb 26th, report Feb 28th
- No class this Thursday, Feb 2/25

Top-Down Parsing (Syntax Analysis)

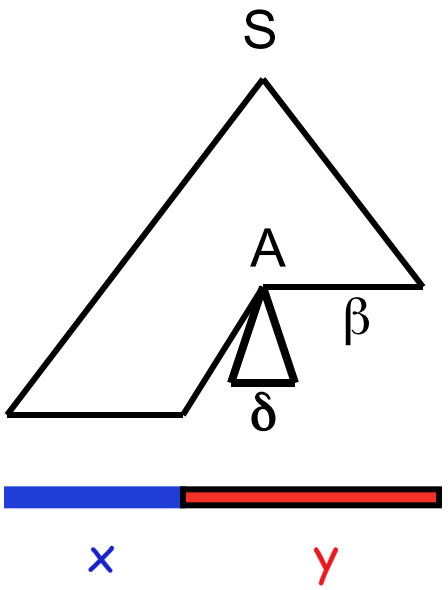
EAC Chapters 3.1 – 3.3

Parsing Techniques: Top-down parsers

LL(1), recursive descent

- 1 input symbol lookahead
- construct leftmost deriviation (forwards)
- input: read left-to-right

$$S \Rightarrow^*_{lm} x\, A\, \beta \Rightarrow_{lm} x\, \delta\, \beta \Rightarrow^*_{lm} x\, y$$



Predictive Parsing

Basic idea

Given $A \rightarrow \alpha \mid \beta$, *the parser should be able to choose between* α & β

FIRST sets

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

That is, $a \in$ FIRST(α) *iff* $\alpha \Rightarrow^* a\,\gamma$, for some γ

The FIRST Set

$a \in \text{FIRST}(\alpha)$ *iff* $\alpha \Rightarrow^* a\,\gamma$, for some γ

To build FIRST(X) for all grammar symbols X:

1. if X is a terminal (token), FIRST(X) := { X }
2. if X ::= ε, then ε ∈ FIRST(X)

3. iterate until no more terminals or ε can be added to any FIRST(X):

   if $X ::= Y_1\, Y_2 \ldots Y_k$ then

   $a \in$ FIRST(X) if $a \in$ FIRST(Y_i) and ε ∈ FIRST(Y_j) for all $1 \le j < i$

   ε ∈ FIRST(X) if ε ∈ FIRST(Y_i) for all $1 \le i \le k$

   end iterate

Note: if ε ∉ FIRST(Y_1), then FIRST(Y_i) is irrelevant, for $1 < i$

The FIRST Set

$a \in \text{FIRST}(\alpha)$ *iff* $\alpha \Rightarrow^* a\,\gamma$, for some γ

To build FIRST(α) for $\alpha = X_1 X_2 \ldots X_n$:

1. $x \in \text{FIRST}(\alpha)$ if $x \in \text{FIRST}(X_i)$ and
   $\varepsilon \in \text{FIRST}(X_j)$ for all $1 \le j < i$

2. $\varepsilon \in \text{FIRST}(\alpha)$ if $\varepsilon \in \text{FIRST}(X_i)$ for all $1 \le i \le n$

Predictive Parsing

Basic idea

Given $A \rightarrow \alpha \mid \beta$, *the parser should be able to choose between* α & β

FIRST sets

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

That is, $a \in$ FIRST(α) *iff* $\alpha \Rightarrow^* a\,\gamma$, for some γ

The LL(1) Property

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ both appear in the grammar, we would like

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \varnothing$$

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

This is almost correct, but not quite

The FOLLOW Set

For a non-terminal A, define FOLLOW(A) as

> *FOLLOW(A) := 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 has no FOLLOW set

The FOLLOW Set

To build FOLLOW(X) for all non-terminal X:

1. Place eof in FOLLOW(<*goal*>)

   iterate until no more terminals or ε can be added to any FOLLOW(X):

2. If $A \rightarrow \alpha B \beta$ then
   put {FIRST(β) - ε} in FOLLOW(B)
3. If $A \rightarrow \alpha B$ then
   put FOLLOW(A) in FOLLOW(B)
4. If $A \rightarrow \alpha B \beta$ and $\varepsilon \in$ FIRST(β) then
   put FOLLOW(A) in FOLLOW(B)

Predictive Parsing

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ and $\varepsilon \in \text{FIRST}(\alpha)$, then we need to ensure that $\text{FIRST}(\beta)$ is disjoint from $\text{FOLLOW}(A)$, too

Define $\text{FIRST}^{+}(\delta)$ for rule $A \rightarrow \delta$ as

- $(\text{FIRST}(\delta) - \{ \varepsilon \}) \cup \text{FOLLOW}(A)$, if $\varepsilon \in \text{FIRST}(\delta)$
- $\text{FIRST}(\delta)$, otherwise

Predictive Parsing

The LL(1) Property

A grammar is *LL(1)* iff $A \rightarrow \alpha$ and $A \rightarrow \beta$ implies

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

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

Question: Can there be two rules $A \rightarrow \alpha$ and $A \rightarrow \beta$ in a *LL(1)* grammar such that $\varepsilon \in \text{FIRST}(\alpha)$ and $\varepsilon \in \text{FIRST}(\beta)$?

Predictive Parsing

Given a grammar that has the *LL(1)* property

- Problem: NT A needs to be replaced in next derivation step
- Assume $A \rightarrow \beta_1 \mid \beta_2 \mid \beta_3$, with

  $FIRST^+(\beta_1) \cap FIRST^+(\beta_2) = \varnothing$, $FIRST^+(\beta_1) \cap FIRST^+(\beta_3) = \varnothing$, and

  $FIRST^+(\beta_2) \cap FIRST^+(\beta_3) = \varnothing$ (pair-wise disjoint sets)

```
/* find rule for A */
if (current token ∈ FIRST+ (β1))
    select A → β1
else if (current token ∈ FIRST+(β2))
    select A → β2
else if (current token ∈ FIRST+(β3))
    select A → β3
else
    report an error and return false
```

Grammars with the *LL(1)* property are called *predictive grammars* because the parser can "predict" the correct expansion at each point in the parse.

Parsers that capitalize on the *LL(1)* property are called *predictive parsers*.

One kind of predictive parser is the *recursive descent* parser. *The other is a table-driven parser table-driven parser.*

LL(1) Parser Example

Is the following grammar LL(1)?

S ::= a S b | ε

LL(1) Parser Example

Is the following grammar LL(1)?

S ::= a S b | ε

First(aSb) = { a }
First(ε) = { ε }

First$^+$(aSb) = { a }
First$^+$(ε) = (First (ε) - { ε }) ∪ Follow (S) = { eof, b }

LL(1)?

LL(1) Parser Example

Is the following grammar LL(1)?

S ::= a S b | ε

First(aSb) = { a }
First(ε) = { ε }

First$^+$(aSb) = { a }
First$^+$(ε) = (First (ε) - { ε }) ∪ Follow (S) = { eof, b }

LL(1)? YES, since { a } ∩ { eof, b } = ∅

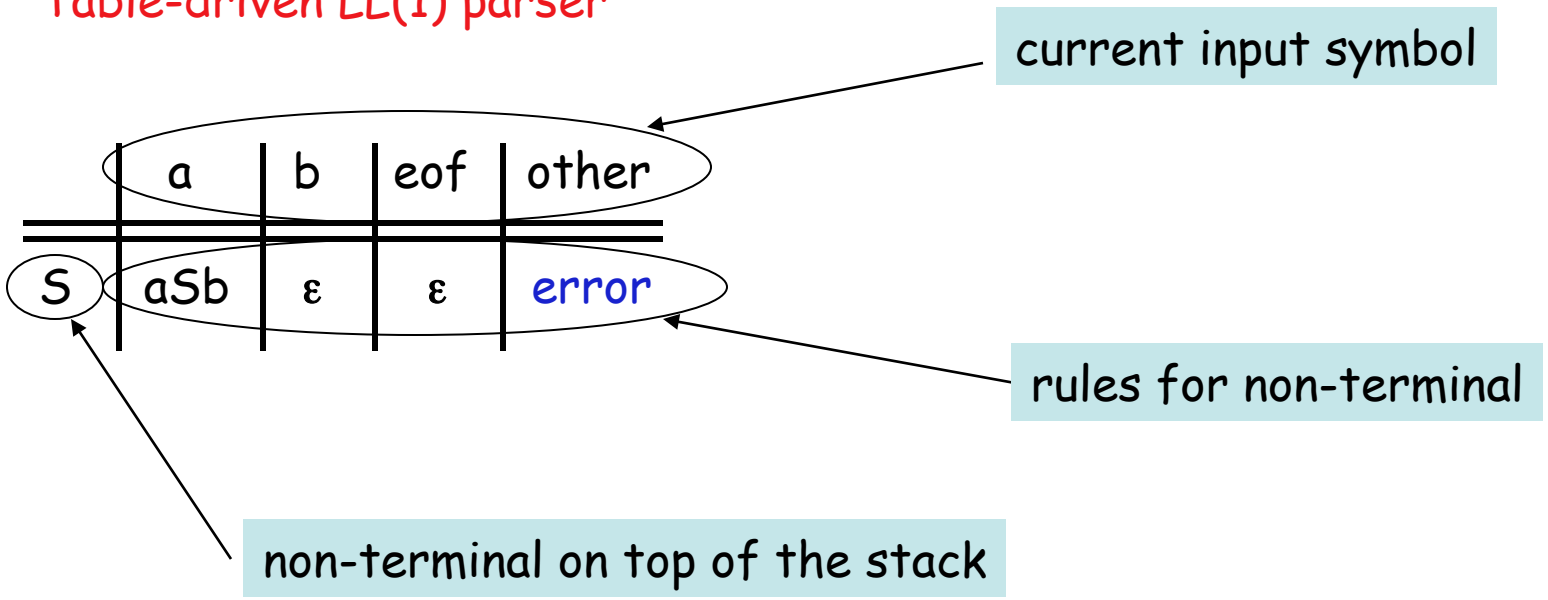
LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

LL(1) Parser Example

Table-driven LL(1) parser



Building Top Down Parsers

Building the complete table

- Need a row for every *NT* & a column for every *T*
- Need an algorithm to build the table

Filling in TABLE[X,y], X ∈ *NT*, y ∈ *T*

- entry is the rule *X ::= β*, if y ∈ FIRST+(*β*)
- entry is error otherwise

If any entry is defined multiple times, *G* is not *LL(1)*

This is the *LL(1)* table construction algorithm

LL(1) Skeleton Parser

```
token ← next_token()
push EOF onto Stack
push the start symbol, S, onto Stack
TOS ← top of Stack
loop forever
  if TOS = EOF and token = EOF then
     break & report success
  else if TOS is a terminal then
     if TOS matches token then
        pop Stack                    // recognized TOS
        token ← next_token()
     else report error looking for TOS
  else                               // TOS is a non-terminal
     if TABLE[TOS,token] is A→ B1B2…Bk then
        pop Stack                    // get rid of A
        push Bk, Bk-1, …, B1         // in that order
     else report error expanding TOS
  TOS ← top of Stack
```

exit on success

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

How to parse input a a a b b b ?

Describe action as sequence of states

(PDA stack content, remaining input, next action)

PDA stack content: [X, … Z], where Z is the TOS
next actions: rule or next input+pop or error or accept

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒
([eof, b, b, S, a], aabbb, next input+pop) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒
([eof, b, b, S, a], aabbb, next input+pop) ⇒
([eof, b, b, S], abbb, aSb) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒
([eof, b, b, S, a], aabbb, next input+pop) ⇒
([eof, b, b, S], abbb, aSb) ⇒
([eof, b, b, b, S, a], abbb, next input+pop) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒
([eof, b, b, S, a], aabbb, next input+pop) ⇒
([eof, b, b, S], abbb, aSb) ⇒
([eof, b, b, b, S, a], abbb, next input+pop) ⇒
([eof, b, b, b, S], bbb, ε) ⇒

LL(1) Parser Example

Table-driven LL(1) parser

	a	b	eof	other
S	aSb	ε	ε	error

([eof, S], aaabbb, aSb) ⇒
([eof, b, S, a], aaabbb, next input+pop) ⇒
([eof, b, S], aabbb, aSb) ⇒
([eof, b, b, S, a], aabbb, next input+pop) ⇒
([eof, b, b, S], abbb, aSb) ⇒
([eof, b, b, b, S, a], abbb, next input+pop) ⇒
([eof, b, b, b, S], bbb, ε) ⇒
([eof, b, b, b], bbb, next input+pop) ⇒ ([eof, b, b], bb, next input+pop) ⇒
([eof, b], b, next input+pop) ⇒ ([eof], eof, accept)

Announcements

- **Midterm on Wednesday, March 8**
  - closed book, closed notes
  - covers everything up to (and including) top-down parsing
  - No recitation on Tuesday, March 8
  - Q&A session second half of Thursday 3/3 lecture – come prepared!
- Project 1 deadline extended, code Feb 26th, report Feb 28th
- No class this Thursday, Feb 2/25

Next class

More Syntax Analysis (bottom-up)

Read EaC: Chapter 3.4