CS310

Parsing with Context Free Grammars

Today's reference:

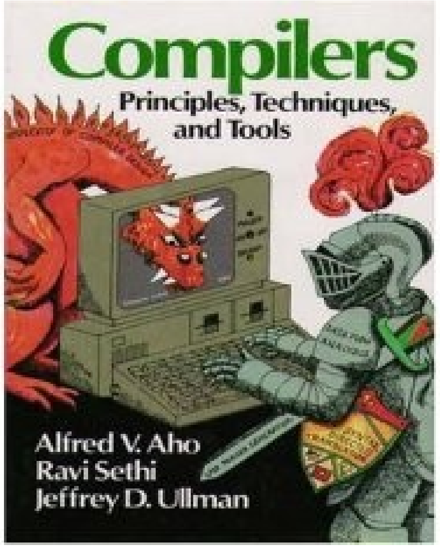
Compilers: Principles, Techniques, and Tools

by: Aho, Sethi, Ullman

aka: The Dragon Book

Section 4.4

October 29, 2010

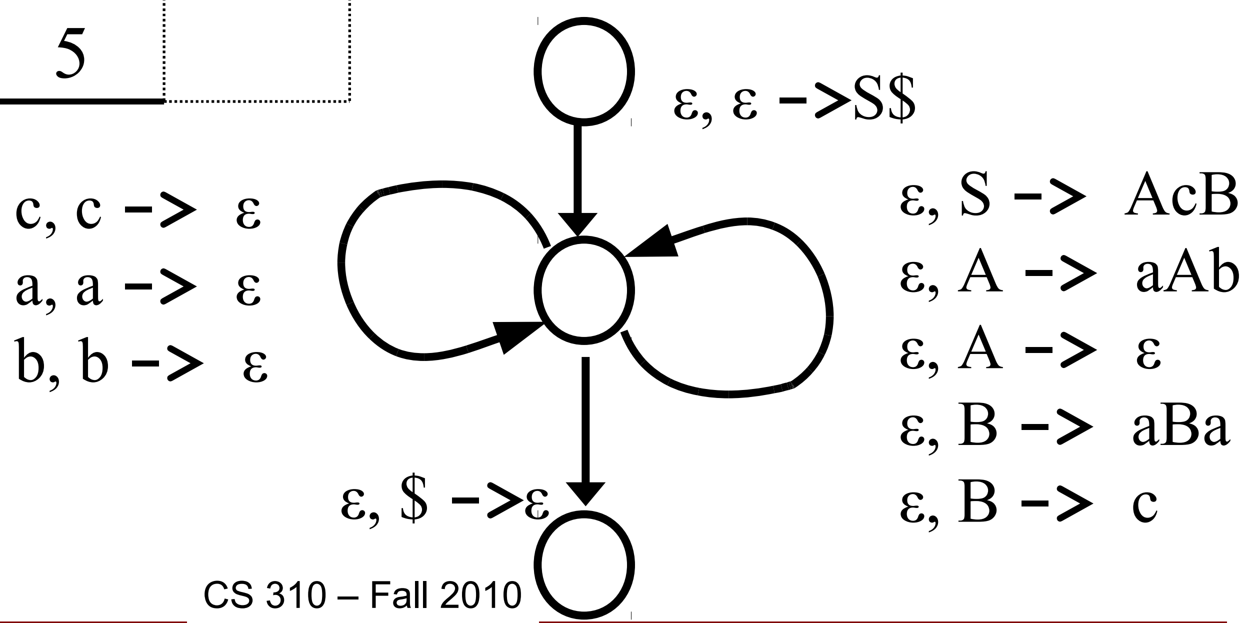


Parse Tables

(1) S -> AcB
(2) A -> aAb
(3) A -> ε
(4) B -> aBb
(5) B -> c

Input / Stack	a	b	c	$
S	1	-	1	
A	2	3	3	
B	4	-	5	

PDA



FIRST

FIRST(X) = terminals that can begin strings derivable from X

X **-*>** av X **-*>** ε

FIRST(X) = {a, ε}

Algorithm

– x is a terminal or ε, FIRST(x) = {x}

– x is nonterminal x **->** $x_1 \mid x_2 \mid .. \mid x_n$

FIRST(x) = U_k FIRST(x_k)

3) x = $x_1x_2..x_n$ (concatenation)

FIRST(x) = FIRST(x_1) U FIRST(x_2) U FIRST(x_3)

if x_1 **-*>** ε if x_2 **-*>** ε

Example

- FIRST(A) =
- FIRST(B) =
- FIRST(S) =
- FIRST(AcB) =
- FIRST(aAb) =
- FIRST(ε) =
- FIRST(aBa)
- FIRST(c) =

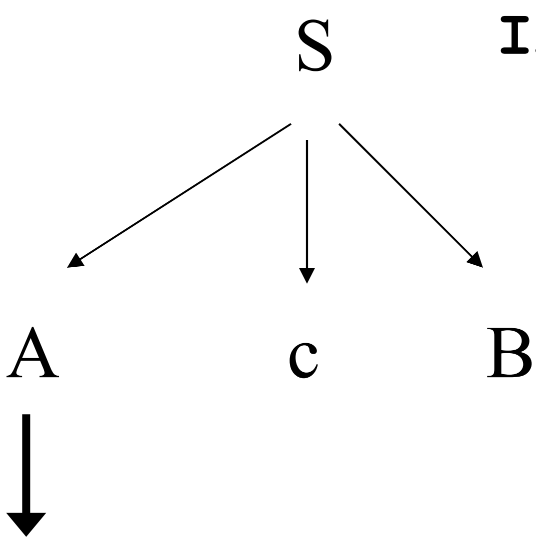
S -> AcB
A -> aAb
A -> ε
B -> aBb
B -> c

FOLLOW

When do we choose which rule to use to expand A?

(And avoid lots of backtracking.)

S -> AcB
A -> aAb
A -> ε
B -> aBb
B -> c



`Input: abcc`

Base our choice on what could possibly follow A

FOLLOW

- For $A \in V$, FOLLOW(A) consists of *terminals,* immediately following A in any intermediate step in a derivation
  - ε is never in FOLLOW
- Algorithm

  1) A = S or A is rightmost symbol in any intermediate step then $\$ \in$ FOLLOW A.

  2) Q **->** aAB, $B \in \{T, V\}^*$, **$A \in V$**, $Q \in V$

     a) B begins with a terminal x, add x to FOLLOW(A)

     b) B = ε or B **-*>** ε include FOLLOW(Q) in FOLLOW(A)

Example

- FOLLOW(A) =
- FOLLOW(B) =
- FOLLOW(S) =

S -> AcB
A -> aAb
A -> ε
B -> aBb
B -> c

Constructing Parse Tables

- For X on the stack and input *a,* select a righthand replacement that begins with *a* or can lead to something beginning with *a*
- Algorithm

  For each X -> B

  a) For each a $\in$ FIRST(B), add B to Parse(X, a)

  b) If $\varepsilon \in$ FIRST(B), add B to Parse(X, b) for each b $\in$ FOLLOW(X)

  c) If \$ $\in$ FOLLOW(X), Parse(X, \$) = B

Parse Tables

(1) S -> AcB
(2) A -> aAb
(3) A -> ε
(4) B -> aBb
(5) B -> c

- Algorithm

  For each X -> B

  a) For each a ∈ FIRST(B), add B to Parse(X, a)

  b) If ε ∈ FIRST(B), add B to Parse(X, b) for each b ∈ FOLLOW(X)

Stack \ Input	a	b	c	$
S	1	-	1	
A	2	3	3	
B	4	-	5	

Example

- Parse: abcacb

(1) S -> AcB
(2) A -> aAb
(3) A -> ε
(4) B -> aBb
(5) B -> c

Top-Down Parsing

- LL(1) parser
  - parse from Left to right
  - produces a Leftmost derivation
    - always replace the left-most nonterminal first
  - with 1 lookahead symbol
- LL(1) Grammars
  - FIRST and FOLLOW uniquely determine which productions to use to parse a string
  - not all grammars are LL(1)
    - common prefixes
    - left recursion

Common Prefixes

- Lead term not sufficient to decide how to expand a nonterminal

S -> if E then S else S | if E then S | E

Parse: if E then if E then E else if E then E else E

```
if E then
    if E then
        E
else
    if E then
        E
    else
        E
```

or

```
if E then
    if E then
        E
    else
        if E then
            E
        else
        E
```

Remove Common Prefixes

Remove: $A \rightarrow aB_1 \mid aB_2 \mid Y$

Add: $A \rightarrow aT \mid Y$

$T \rightarrow B_1 \mid B_2$