



TOP-DOWN PARSING

CMPT 379 Lecture 14b

Lecture Overview

- FIRST, FOLLOWS, FIRST⁺
- Parsing Tables
- LL(1) Parsing
- Recursive Descent

Analyzing a Grammar

- Suppose we have a grammar G with productions P . We wish to tell if there is an automatic way to parse G .
- We will use the following example grammar:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid - T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid / F T' \mid \varepsilon$$

$$F \rightarrow i \mid c \mid (E)$$

FIRST

- For any grammar symbol B (terminal, nonterminal, ϵ , or $\langle \text{eof} \rangle$) we define $\text{FIRST}(B)$ to be the set of **terminals** that can appear as the first word in a string derived from B.
- If B is a terminal, ϵ , or $\langle \text{eof} \rangle$, then $\text{FIRST}(B) = B$.

$\text{FIRST}(E) = \{ i, c, (\}$

$\text{FIRST}(E') = \{ +, -, \epsilon \}$

$\text{FIRST}(T) = \{ i, c, (\}$

$\text{FIRST}(T') = \{ *, /, \epsilon \}$

$\text{FIRST}(F) = \{ i, c, (\}$

FIRST OVER STRINGS

For any grammar symbol string $s = B_1 B_2 B_3 \dots B_k$ we define $\text{FIRST}(s)$ to be the set of **terminals** that can appear as the first word in a string derived from s . It is the union of FIRST sets for $B_1 B_2 \dots B_n$, where B_n is the first symbol whose FIRST set does not contain ϵ . ϵ is in $\text{FIRST}(s)$ iff it is in $\text{FIRST}(B_i)$ for all $i = 1$ to k .

$$\text{FIRST}(EE') = \{i, c, (\}$$

$$\text{FIRST}(E'ET) = \{+, -, i, c, (\}$$

$$\text{FIRST}(T'E') = \{*, /, +, -, \epsilon\}$$

FOLLOW

- For any nonterminal N we define FOLLOW(N) to be the set of **terminals** that can appear to the immediate right of a string derived from N.

FOLLOW(E) = {<eof>,) }

FOLLOW(E') = {<eof>,) }

FOLLOW(T) = {<eof>, +, -,) }

FOLLOW(T') = {<eof>, +, -,) }

FOLLOW(F) = {<eof>, +, -, *, /,) }

FIRST⁺

- For any production $A \rightarrow \beta$, where β is a string of terminals and nonterminals, we define FIRST⁺($A \rightarrow \beta$) to be

$$\begin{cases} \text{FIRST}(\beta) \cup \text{FOLLOW}(A) & \text{if } \varepsilon \in \text{FIRST}(\beta) \\ \text{FIRST}(\beta) & \text{otherwise} \end{cases}$$

$$\text{FIRST}^+(E' \rightarrow + T E') = \{+\}$$

$$\text{FIRST}^+(E' \rightarrow - T E') = \{-\}$$

$$\text{FIRST}^+(E' \rightarrow \varepsilon) = \{<\text{eof}>,)\}$$

FIRST⁺

In a **predictive grammar**, if there are two productions $A \rightarrow \beta_1$ and $A \rightarrow \beta_2$, then it must be that $\text{FIRST}^+(A \rightarrow \beta_1)$ and $\text{FIRST}^+(A \rightarrow \beta_2)$ are disjoint.

In this case, the FIRST⁺ sets completely encode the decisions needed for parsing. We can express them in a table called a **LL(1) parsing table**.

First we must number our (BNF) productions.

LL(1) Parsing Table

1. $E \rightarrow T E'$
2. $E' \rightarrow + T E'$
3. $E' \rightarrow - T E'$
4. $E' \rightarrow \epsilon$
5. $T \rightarrow F T'$
6. $T' \rightarrow * F T'$
7. $T' \rightarrow / F T'$
8. $T' \rightarrow \epsilon$
9. $F \rightarrow i$
10. $F \rightarrow c$
11. $F \rightarrow (E)$

	+	-	*	/	i	c	()	eof
E	-	-	-	-	1	1	1	-	-
E'	2	3	-	-	-	-	-	4	4
T	-	-	-	-	5	5	5	-	-
T'	8	8	6	7	-	-	-	8	8
F	-	-	-	-	9	10	11	-	-

Parsing Example

	+	-	*	/	i	c	()	eof
E	-	-	-	-	1	1	1	-	-
E'	2	3	-	-	-	-	-	4	4
T	-	-	-	-	5	5	5	-	-
T'	8	8	6	7	-	-	-	8	8
F	-	-	-	-	9	10	11	-	-

1. $E \rightarrow T E'$
2. $E' \rightarrow + T E'$
3. $E' \rightarrow - T E'$
4. $E' \rightarrow \epsilon$
5. $T \rightarrow F T'$
6. $T' \rightarrow * F T'$
7. $T' \rightarrow / F T'$
8. $T' \rightarrow \epsilon$
9. $F \rightarrow i$
10. $F \rightarrow c$
11. $F \rightarrow (E)$

stack

eof E
 eof E' T
 eof E' T' F
 eof E' T' c
 eof E' T'
 eof E' T +
 eof E' T
 eof E' T' F
 eof E' T' c
 eof E' T'
 eof E' T' F *
 eof E' T' F
 eof E' T' i
 eof E' T'
 eof E'
 eof

input

c + c * i eof
 c + c * i eof
 c + c * i eof
 c + c * i eof
 + c * i eof
 + c * i eof
 + c * i eof
 c * i eof
 c * i eof
 c * i eof
 * i eof
 * i eof
 i eof
 i eof
 eof
 eof
 eof

prod.

1
 5
 10
 match
 8
 2
 match
 5
 10
 match
 6
 match
 9
 match
 8
 4
 match

Leftmost Derivation

If we write the matched terminals followed by the stack contents (from top to bottom), we get an interesting sequence of strings

matched + stack

E

T E'

F T' E'

c T' E'

c T' E'

c E'

c + T E'

c + T E'

c + F T' E'

c + c T' E'

c + c T' E'

c + c T' E'

c + c * F T' E'

c + c * F T' E'

c + c * i T' E'

c + c * i T' E'

c + c * i E'

c + c * i

This is a **leftmost derivation** in the grammar. In each step, the **leftmost** nonterminal is replaced by one of its productions' right-hand sides.

LL(1) Parsing Algorithm

```
token = NextToken()
create stack T
T.push eof
T.push(S)           // start symbol
```

loop forever:

```
    focus = T.top()    // not T.pop()
    if focus = token = eof
        report success; exit loop;
    else if focus is a terminal or eof
        if focus matches token
            T.pop()
            token = nextToken()
        else
            report error looking for token
            at top of the stack
```

else

```
if Table(focus, token) is
     $A \rightarrow B_1 B_2 \dots B_k$ 
```

```
T.pop()
```

```
for i = k to 1 step -1
```

```
    if  $B_i$  is not  $\epsilon$ 
```

```
        T.push( $B_i$ )
```

else

```
    report an error expanding
    focus
```

LL(1) Table Creation

build **FIRST, FOLLOW, FIRST⁺** sets.

for each nonterminal A

for each terminal w

 // including eof

 Table[A, w] = error;

for each production p of the form $A \rightarrow \beta$

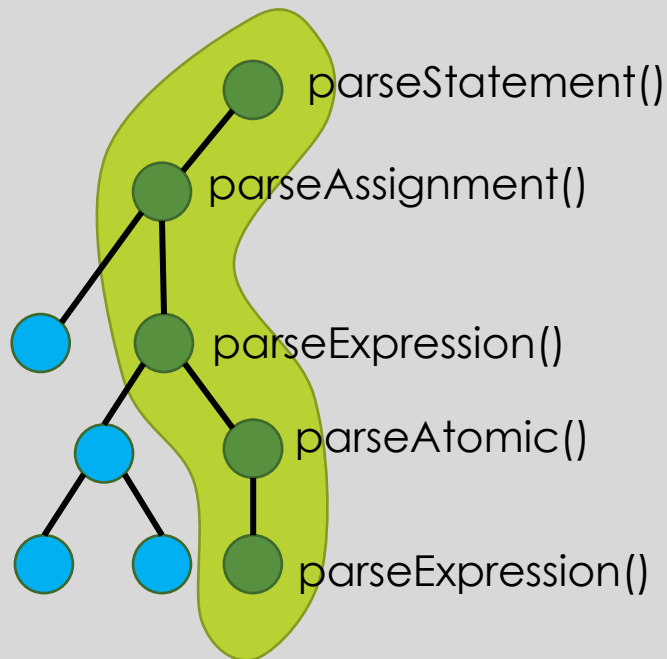
for each terminal w in FIRST⁺($A \rightarrow \beta$)

 // including eof

 Table[A, w] = p

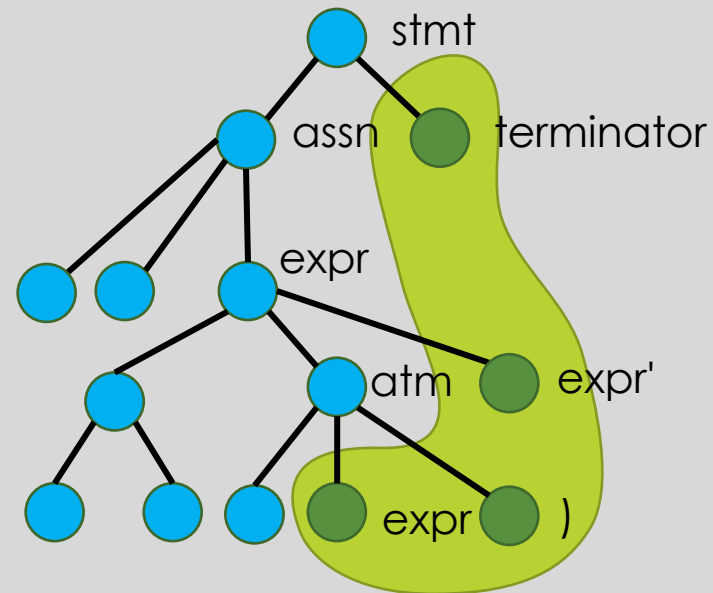
Top-Down Parsing

Recursive Descent



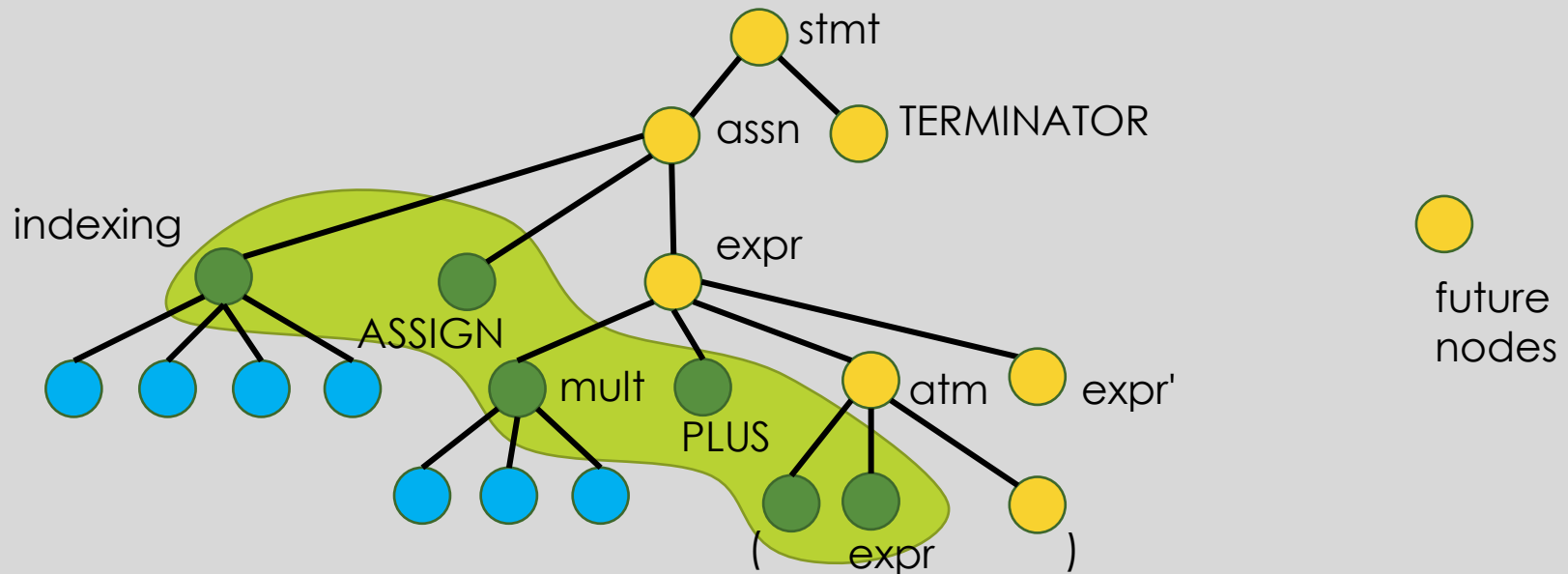
Recursion stack contains path from root to current node

LL(1) Table Parsing



Stack contains right children of path from root to current node

Bottom-up Parsing



Stack contains roots of discovered subtrees