Syntax Analysis:

Context-free Grammars, Pushdown Automata and Parsing
Part - 3

Y.N. Srikant

Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Principles of Compiler Design

Outline of the Lecture

- What is syntax analysis? (covered in lecture 1)
- Specification of programming languages: context-free grammars (covered in lecture 1)
- Parsing context-free languages: push-down automata (covered in lectures 1 and 2)
- Top-down parsing: LL(1) and recursive-descent parsing
- Bottom-up parsing: LR-parsing

Testable Conditions for LL(1)

- We call strong LL(1) as LL(1) from now on and we will not consider lookaheads longer than 1
- The classical condition for LL(1) property uses *FIRST* and *FOLLOW* sets
- If α is any string of grammar symbols ($\alpha \in (N \cup T)^*$), then
  $FIRST(\alpha) = \{a \mid a \in T, \text{ and } \alpha \Rightarrow^* ax, \ x \in T^*\}$
  $FIRST(\epsilon) = \{\epsilon\}$
- If A is any nonterminal, then
  $$FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, \ \alpha, \beta \in (N \cup T)^*, \ a \in T \cup \{\$\}\}$$
- $FIRST(\alpha)$ is determined by α alone, but $FOLLOW(A)$ is determined by the "context" of A, i.e., the derivations in which A occurs

FIRST and *FOLLOW* Computation Example

- Consider the following grammar
  $S' \rightarrow S\$,\ S \rightarrow aAS \mid c,\ A \rightarrow ba \mid SB,\ B \rightarrow bA \mid S$
- $FIRST(S') = FIRST(S) = \{a, c\}$ because
  $S' \Rightarrow S\$ \Rightarrow \underline{c}\$$, and $S' \Rightarrow S\$ \Rightarrow \underline{a}AS\$ \Rightarrow \underline{a}baS\$ \Rightarrow \underline{a}bac\$$
- $FIRST(A) = \{a, b, c\}$ because
  $A \Rightarrow \underline{b}a$, and $A \Rightarrow SB$, and therefore all symbols in $FIRST(S)$ are in $FIRST(A)$
- $FOLLOW(S) = \{a, b, c, \$\}$ because
  $S' \Rightarrow \underline{S\$}$,
  $S' \Rightarrow^* aAS\$ \Rightarrow a\underline{S}BS\$ \Rightarrow aS\underline{b}AS\$$,
  $S' \Rightarrow^* a\underline{S}BS\$ \Rightarrow a\underline{S}SS\$ \Rightarrow aS\underline{a}ASS\$$,
  $S' \Rightarrow^* a\underline{S}SS\$ \Rightarrow aS\underline{c}S\$$
- $FOLLOW(A) = \{a, c\}$ because
  $S' \Rightarrow^* a\underline{A}S\$ \Rightarrow aA\underline{a}AS\$$,
  $S' \Rightarrow^* a\underline{A}S\$ \Rightarrow aA\underline{c}$

Computation of *FIRST*: Terminals and Nonterminals

```
{
 for each (a ∈ T) FIRST(a) = {a}; FIRST(ϵ) = {ϵ};
 for each (A ∈ N) FIRST(A) = ∅;
 while (FIRST sets are still changing) {
     for each production p {
        Let p be the production A → X1 X2...Xn;
        FIRST(A) = FIRST(A) ∪ (FIRST(X1) - {ϵ});
        i = 1;
        while (ϵ ∈ FIRST(Xi) && i ≤ n − 1) {
            FIRST(A) = FIRST(A) ∪ (FIRST(Xi+1 − {ϵ}); i + +;
        }
        if (i == n) && (ϵ ∈ FIRST(Xn))
           FIRST(A) = FIRST(A) ∪{ϵ}
 }
}
```

Computation of $FIRST(\beta)$: β, a string of Grammar Symbols

```
{ /* It is assumed that FIRST sets of terminals and nonterminals
    are already available /*
    FIRST(β) = ∅;
    while (FIRST sets are still changing) {
      Let β be the string X1 X2...Xn;
      FIRST(β) = FIRST(β) ∪ (FIRST(X1) - {ε});
      i = 1;
      while (ε ∈ FIRST(Xi) && i ≤ n − 1) {
         FIRST(β) = FIRST(β) ∪ (FIRST(Xi+1 − {ε}); i + +;
      }
      if (i == n) && (ε ∈ FIRST(Xn))
        FIRST(β) = FIRST(β) ∪{ε}
    }
}
```

FIRST Computation: Algorithm Trace - 1

- Consider the following grammar
  $S' \to S\$,\ S \to aAS \mid \epsilon,\ A \to ba \mid SB,\ B \to cA \mid S$
- Initially, FIRST(S) = FIRST(A) = FIRST(B) = $\emptyset$
- Iteration 1
  - FIRST(S) = $\{a, \epsilon\}$ from the productions $S \to aAS \mid \epsilon$
  - FIRST(A) = $\{b\} \cup$ FIRST(S) - $\{\epsilon\} \cup$ FIRST(B) - $\{\epsilon\}$ = $\{b, a\}$
    from the productions $A \to ba \mid SB$
    (since $\epsilon \in$ FIRST(S), FIRST(B) is also included;
    since FIRST(B)=ϕ, ϵ is not included)
  - FIRST(B) = $\{c\} \cup$ FIRST(S) - $\{\epsilon\} \cup \{\epsilon\}$ = $\{c, a, \epsilon\}$
    from the productions $B \to cA \mid S$
    (ϵ is included because $\epsilon \in$ FIRST(S))

FIRST Computation: Algorithm Trace - 2

- The grammar is
  $S' \rightarrow S\$,\ S \rightarrow aAS \mid \epsilon,\ A \rightarrow ba \mid SB,\ B \rightarrow cA \mid S$
- From the first iteration,
  FIRST(S) = $\{a, \epsilon\}$, FIRST(A) = $\{b, a\}$, FIRST(B) = $\{c, a, \epsilon\}$
- Iteration 2
  (values stabilize and do not change in iteration 3)
  - FIRST(S) = $\{a, \epsilon\}$ (no change from iteration 1)
  - FIRST(A) = $\{b\}$ $\cup$ FIRST(S) - $\{\epsilon\}$ $\cup$ FIRST(B) - $\{\epsilon\}$ $\cup\{\epsilon\}$
    = $\{b, a, c, \epsilon\}$ (changed!)
  - FIRST(B) = $\{c, a, \epsilon\}$ (no change from iteration 1)

Computation of *FOLLOW*

{ for each ($X \in N \cup T$) FOLLOW(X) = $\emptyset$;
  FOLLOW(S) = {\$}; /* S is the start symbol of the grammar */
  repeat {
    for each production $A \rightarrow X_1 X_2 ... X_n$ {/* $X_i \neq \epsilon$ */
      FOLLOW(X_n) = FOLLOW(X_n) $\cup$ FOLLOW(A);
      REST = FOLLOW(A);
      for i = n downto 2 {
        if ($\epsilon \in$ FIRST(X_i)) { FOLLOW(X_{i-1}) =
            FOLLOW(X_{i-1}) $\cup$ (FIRST (X_i) $- \{\epsilon\}$)$\cup$ REST;
            REST = FOLLOW(X_{i-1});
        } else { FOLLOW(X_{i-1}) = FOLLOW(X_{i-1}) $\cup$ FIRST (X_i) ;
                  REST = FOLLOW(X_{i-1}); }
      }
    }
  } until no FOLLOW set has changed
}

FOLLOW Computation: Algorithm Trace

- Consider the following grammar
  $S' \rightarrow S\$,\ S \rightarrow aAS \mid \epsilon,\ A \rightarrow ba \mid SB,\ B \rightarrow cA \mid S$
- Initially, $follow(S) = \{\$\}$; $follow(A) = follow(B) = \emptyset$
  $first(S) = \{a, \epsilon\}$; $first(A) = \{a, b, c, \epsilon\}$; $first(B) = \{a, c, \epsilon\}$;
- Iteration 1 /* In the following, $x \cup = y$ means $x = x \cup y$ */
  - $S \rightarrow aAS$: $follow(S) \cup = \{\$\}$; $rest = follow(S) = \{\$\}$
    $follow(A) \cup = (first(S) - \{\epsilon\}) \cup rest = \{a, \$\}$
  - $A \rightarrow SB$: $follow(B) \cup = follow(A) = \{a, \$\}$
    $rest = follow(A) = \{a,\$\}$
    $follow(S) \cup = (first(B) - \{\epsilon\}) \cup rest = \{a, c, \$\}$
  - $B \rightarrow cA$: $follow(A) \cup = follow(B) = \{a,\$\}$
  - $B \rightarrow S$: $follow(S) \cup = follow(B) = \{a, c,\$\}$
  - At the end of iteration 1
    $follow(S) = \{a, c,\$\}$; $follow(A) = follow(B) = \{a, \$\}$

FOLLOW Computation: Algorithm Trace (contd.)

- $first(S) = \{a, \epsilon\}$; $first(A) = \{a, b, c, \epsilon\}$; $first(B) = \{a, c, \epsilon\}$;
- At the end of iteration 1
  $follow(S) = \{a, c, \$\}$; $follow(A) = follow(B) = \{a, \$\}$
- Iteration 2
- $S \rightarrow aAS$: $follow(S) \cup = \{a, c, \$\}$;
  $rest = follow(S) = \{a, c, \$\}$
  $follow(A) \cup = (first(S) - \{\epsilon\}) \cup rest = \{a, c, \$\}$ (changed!)
- $A \rightarrow SB$: $follow(B) \cup = follow(A) = \{a, c, \$\}$ (changed!)
  $rest = follow(A) = \{a, c, \$\}$
  $follow(S) \cup = (first(B) - \{\epsilon\}) \cup rest = \{a, c, \$\}$ (no change)
- At the end of iteration 2
  $follow(S) = follow(A) = follow(B) = \{a, c, \$\}$;
- The *follow* sets do not change any further

LL(1) Conditions

- Let G be a context-free grammar
- G is LL(1) iff for every pair of productions $A \rightarrow \alpha$ and $A \rightarrow \beta$, the following condition holds
  - $dirsymb(\alpha) \cap dirsymb(\beta) = \emptyset$, where
    $dirsymb(\gamma)$ = if ($\epsilon \in first(\gamma)$) then
    $((first(\gamma) - \{\epsilon\}) \cup follow(A))$ else $first(\gamma)$
    (γ stands for α or β)
  - *dirsymb* stands for "direction symbol set"
- An equivalent formulation (as in ALSU's book) is as below
  - $first(\alpha.follow(A)) \cap first(\beta.follow(A)) = \emptyset$
- Construction of the LL(1) parsing table

```
for each production A → α
    for each symbol s ∈ dirsymb(α)
    /* s may be either a terminal symbol or $ */
        add A → α to LLPT[A, s]
Make each undefined entry of LLPT as error
```

LL(1) Table Construction using *FIRST* and *FOLLOW*

for each production $A \rightarrow \alpha$
    for each terminal symbol $a \in \mathit{first}(\alpha)$
        add $A \rightarrow \alpha$ to $LLPT[A, a]$
    if $\epsilon \in \mathit{first}(\alpha)$ {
      for each terminal symbol $b \in \mathit{follow}(A)$
          add $A \rightarrow \alpha$ to $LLPT[A, b]$
      if $\$ \in \mathit{follow}(A)$
          add $A \rightarrow \alpha$ to $LLPT[A, \$]$
    }
Make each undefined entry of *LLPT* as *error*

- After the construction of the LL(1) table is complete (following any of the two methods), if any slot in the LL(1) table has two or more productions, then the grammar is NOT LL(1)

Simple Example of LL(1) Grammar

- P1: $S \rightarrow$ *if* (*a*) *S else S* | *while* (*a*) *S* | *begin SL end*
  P2: $SL \rightarrow S\ S'$
  P3: $S' \rightarrow ;\ SL \mid \epsilon$
- {if, while, begin, end, a, (,), ;} are all terminal symbols
- Clearly, all alternatives of P1 start with distinct symbols and hence create no problem
- P2 has no choices
- Regarding P3, dirsymb(;SL) = {;}, and dirsymb(ϵ) = {end}, and the two have no common symbols
- Hence the grammar is LL(1)

LL(1) Table Construction Example 1

LL(1) Parsing Table for the original grammar

	if	id	else	a	\$
S'	$S' \rightarrow S\$$			$S' \rightarrow S\$$	
S	$S \rightarrow$ if id S $S \rightarrow$ if id S else S			$S \rightarrow a$	

Original Grammar

Grammar is not LL(1)

$S' \rightarrow S\$$
$S \rightarrow$ if id S |
if id S else S |
a

tokens: if, id, else, a

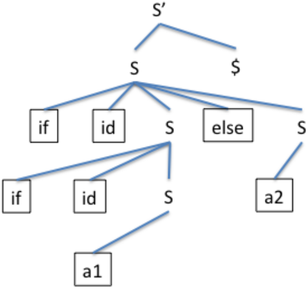
dirsymb(S\$) = {if,a}; dirsymb (a) = {a}
dirsymb(if id S) = {if}
dirsymb(if id S else S) = {if}

$$dirsymb(if\ id\ S) \bigcap dirsymb(a) = \varnothing$$

$$dirsymb(if\ id\ S\ else\ S) \bigcap dirsymb(a) = \varnothing$$

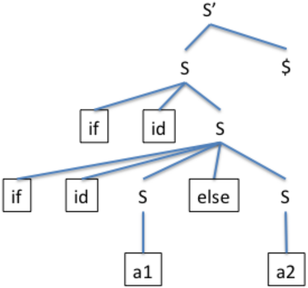
$$dirsymb(if\ id\ S) \bigcap dirsymb(if\ id\ S\ else\ S) \neq \varnothing$$

LL(1) Table Problem Example 1



string: if id (if id a1) else a2

parentheses are not part of the string



string: if id (if id a1 else a2)

parentheses are not part of the string

LL(1) Table Construction Example 2

Original Grammar

$S' \rightarrow S\$$
$S \rightarrow$ if id S |
if id S else S |
a

LL(1) Parsing Table for modified grammar

	if	else	a	$
S′	S′ → S$		S′ → S$	
S	S → if id S S1		S → a	
S1		S1 → ε S1 → else S		S1 → ε

dirsymb(S$) = {if, a}; dirsymb (a) = {a}
dirsymb(if id S S1) = {if}
dirsymb(else S) = {else}
dirsymb(ε) = {else, $}

Grammar is not LL(1)

Left-Factored Grammar

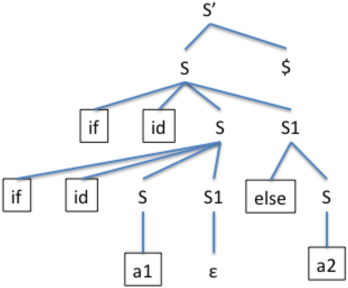
$S' \rightarrow S\$$
$S \rightarrow$ if id S $S1$ | a
$S1 \rightarrow \varepsilon$ | else S

tokens: if, id, else, a

$$dirsymb(if\ id\ S\ S1) \bigcap dirsymb(a) = \varnothing$$

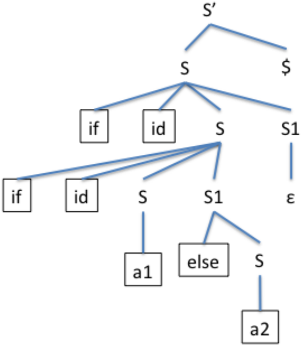
$$dirsymb(\varepsilon) \bigcap dirsymb(else\ S) \neq \varnothing$$

LL(1) Table Problem Example 2



string: if id (if id a1) else a2

parentheses are not part of the string



string: if id (if id a1 else a2)

parentheses are not part of the string

LL(1) Table Construction Example 3

$S' \rightarrow S\$$
$S \rightarrow aAS \mid c$
$A \rightarrow ba \mid SB$
$B \rightarrow bA \mid S$
Grammar is LL(1)

LL(1) Parsing Table

	a	b	c	$
S'	$S' \rightarrow S\$$		$S' \rightarrow S\$$	
S	$S \rightarrow aAS$		$S \rightarrow c$	
A	$A \rightarrow SB$	$A \rightarrow ba$	$A \rightarrow SB$	
B	$B \rightarrow S$	$B \rightarrow bA$	$B \rightarrow S$	

first(S) = {a,c}
first(A) = {a,b,c}
first(B) = {a,b,c}

$$dirsymb(aAS) \cap dirsymb(c) = \varnothing$$
$$dirsymb(ba) \cap dirsymb(SB) = \varnothing$$
$$dirsymb(bA) \cap dirsymb(S) = \varnothing$$

follow(S) = {a,b,c,$}
follow(A) = {a,c}
follow(B) = {a,c}

dirsymb(S$) = {a,c}
dirsymb(aAS) = {a}
dirsymb(c) = {c}

dirsymb(ba) = {b}
dirsymb(SB) = {a,c}
dirsymb(bA) = {b}
dirsymb(S) = {a,c}

LL(1) Table Construction Example 4

Left-Recursive Grammar for Statement List

S′ → SL $
SL → SL S | S
S → a

dirsymb(SL $) = {a}
dirsymb (a) = {a}
dirsymb(SL S) = {a}
dirsymb(S) = {a}

LL(1) Parsing Table for Left-Recursive Grammar

	a
S′	S′ → SL $
SL	SL → SL S SL → S
S	S → a

Grammar is not LL(1)

$$dirsymb(SL\ \ S) \bigcap dirsymb(S) \neq \varnothing$$

Right-Recursive Grammar for Statement List

S′ → SL $
SL → S A
A → S A | ε
S → a

LL(1) Parsing Table for Right-Recursive Grammar

	a	$
S′	S′ → SL $	
SL	SL → S A	
A	A → S A	A → ε
S	S → a	

dirsymb(SL $) = {a}
dirsymb (a) = {a}
dirsymb(S A) = {a}
dirsymb(ε) = {$}

Grammar is LL(1)

$$dirsymb(S\ A) \bigcap dirsymb(\varepsilon) = \varnothing$$

Elimination of Useless Symbols

Now we study the *grammar transformations*, elimination of useless symbols, elimination of left recursion and left factoring

- Given a grammar $G = (N, T, P, S)$, a non-terminal X is *useful* if $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$, where, $w \in T^*$
  Otherwise, X is useless
- Two conditions have to be met to ensure that X is useful
  1. $X \Rightarrow^* w, w \in T^*$ (X derives some terminal string)
  2. $S \Rightarrow^* \alpha X \beta$ (X occurs in some string derivable from S)
- Example: $S \rightarrow AB \mid CA,\ B \rightarrow BC \mid AB,\ A \rightarrow a,$
  $C \rightarrow aB \mid b,\ D \rightarrow d$
  1. $A \rightarrow a,\ C \rightarrow b,\ D \rightarrow d,\ S \rightarrow CA$
  2. $S \rightarrow CA,\ A \rightarrow a,\ C \rightarrow b$

Testing for $X \Rightarrow^* w$

G' = (N',T',P',S') is the new grammar
N_OLD = ϕ;
N_NEW = $\{X \mid X \rightarrow w, w \in T^* \}$
while N_OLD $\neq$ N_NEW do {
  N_OLD = N_NEW;
  N_NEW = N_OLD $\cup \{X \mid X \rightarrow \alpha, \alpha \in (T \cup N\_OLD)^*\}$
}
N' = N_NEW; T' = T; S' = S;
P' = $\{p \mid$ all symbols of p are in $N' \cup T'\}$

Testing for $S \Rightarrow^* \alpha X \beta$

G' = (N',T',P',S') is the new grammar
N' = {S};
Repeat {
  for each production $A \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$ with $A \in N'$ do
    add all nonterminals of $\alpha_1, \alpha_2, ..., \alpha_n$ to N' and
    all terminals of $\alpha_1, \alpha_2, ..., \alpha_n$ to T'
} until there is no change in N' and T'
P' = {$p \mid$ all symbols of p are in $N' \cup T'$}; S' = S