Compilers (CS31003)

Lecture 06

Pralay Mitra

A grammar

$G =< T, N, S, P >$ is a (context-free) grammar where:

T	:	Set of terminal symbols
N	:	Set of non-terminal symbols
S	:	$S \in N$ is the start symbol
P	:	Set of production rules

Every production rule is of the form: $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$.

Symbol convention:

$a, b, c, \cdots$	Lower case letters at the beginning of alphabet	$\in T$
$x, y, z, \cdots$	Lower case letters at the end of alphabet	$\in T^+$
$A, B, C, \cdots$	Upper case letters at the beginning of alphabet	$\in N$
$X, Y, Z, \cdots$	Upper case letters at the end of alphabet	$\in (N \cup T)$
$\alpha, \beta, \gamma, \cdots$	Greek letters	$\in (N \cup T)^*$

A grammar

* G_1=({E}, {+,*,(,),Id}, P_1, E)
* P_1 E → E + E | E * E | (E) | Id

Id+Id

Left most derivation:

unambiguous grammar

E ➔ E + E ➔ Id + E ➔ Id +Id

Right most derivation

E ➔ E + E ➔ E + Id ➔ Id +Id

A grammar

* G_1=({E}, {+,*,(,),Id}, P_1, E)
* P_1 E → E + E | E * E | (E) | Id

Id*Id+Id

ambiguous grammar

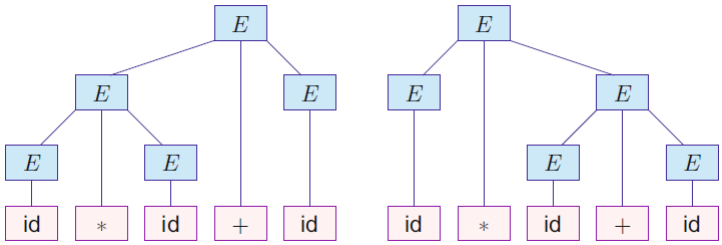
Left most derivation:
E ➔ E + E ➔ E * E + E ➔ Id * E + E ➔ Id * Id + E ➔ Id * Id + Id
E ➔ E * E ➔ Id * E ➔ Id * E + E ➔ Id * Id + E ➔ Id * Id + Id

Right most derivation
E ➔ E * E ➔ E * E + E ➔ E * E + Id ➔ E * Id + Id ➔ Id * Id + Id
E ➔ E + E ➔ E + Id ➔ E * E + Id ➔ E * Id + Id ➔ Id * Id + Id

A grammar

- G_1=({E}, {+,*,(,),Id}, P_1, E)
- P_1 E → E + E | E * E | (E) | Id

ambiguous grammar



Two grammars

- G_0=({E,T,F}, {+,*,(,),Id}, P_0, E)
- P_0 E → E + T | T
  T → T * F | F
  F → (E) | Id

- G_1=({E}, {+,*,(,),Id}, P_1, E)
- P_1 E → E + E | E * E | (E) | Id

Parsing Fundamentals

Derivation	Parsing	Parser	Remarks
Left-most	Top-Down	Predictive: Recursive Descent, LL(1)	No Ambiguity No Left-recursion Tool: Antlr
Right-most	Bottom-Up	Shift-Reduce: SLR, LALR(1), LR(1)	Ambiguity okay Left-recursion okay Tool: YACC, Bison

Recursive Descent Parser

$S \rightarrow c\ A\ d$
$A \rightarrow ab \mid a$

```
int main() {
    l = getchar();
    S(); // S is a start symbol

    // Here l is lookahead. If l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
S() { // Definition of S, as per the given production
    match('c');
    A();
    match('d');
}
A() { // Definition of A as per the given production
    match('a');
    if (l == 'b') { // Look-ahead for decision
        match('b');
    }
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: cad$ ($S \Rightarrow cAd \Rightarrow cad$), cabd$ ($S \Rightarrow cAd \Rightarrow cabd$), caad$

Recursive Descent Parser

$S \rightarrow c\ A\ d$
$A \rightarrow aAb \mid a$

```
int main() {
    l = getchar();
    S(); // S is a start symbol.

    // Here l is lookahead. if l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
S() { // Definition of S, as per the given production
    match('c');
    A();
    match('d');
}
A() { // Definition of A as per the given production
    match('a');
    if (l == 'a') { // Look-ahead for decision
        A();
        match('b');
    }
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: cad$ ($S \Rightarrow cAd \Rightarrow cad$), cabd$, caabd$ ($S \Rightarrow cAd \Rightarrow caAbd \Rightarrow caabd$)

Recursive Descent Parser

$E \rightarrow a\ E'$
$E' \rightarrow +\ a\ E' \mid \epsilon$

```
int main() {
    l = getchar();
    E(); // E is a start symbol.
    // Here l is lookahead. If l = $, it represents the end of the string
    if (l == '$') printf("Parsing Successful");
    else printf("Error");
}
E() { // Definition of E, as per the given production
    match('a');
    E'();
}
E'() { // Definition of E' as per the given production
    if (l == '+') { // Look-ahead for decision
        match('+');
        match('a');
        E'();
    }
    else return (); // epsilon production
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: a$ ($E \Rightarrow aE' \Rightarrow a$), a+a$ ($E \Rightarrow aE' \Rightarrow a + aE' \Rightarrow a + a$), a+a+a$ ($E \Rightarrow aE' \Rightarrow a + aE' \Rightarrow a + a + aE' \Rightarrow a + a + a$)

Recursive Descent Parser

$E \rightarrow E + E \mid a$

```
int main() {
    l = getchar();
    E(); // E is a start symbol.

    // Here l is lookahead. if l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
E() { // Definition of E as per the given production
    if (l == 'a') { // Terminate ? -- Look-ahead does not work
        match('a');
    }

    E();            // Call ?
    match('+');
    E();
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: a+a$, a+a+a$