THEORY OF COMPUTATION AND COMPILERS

Unit - II

CONTEXT FREE GRAMMARS AND PARSING

- Introduction
- Context-Free Grammars - Derivation, Parse trees, Ambiguity
- **Types of Parsers** 
- LL(K) grammars and LL(1) parsing
- Bottom-up Parsing - handle pruning
- LR Grammar Parsing
- LALR parsing
- Parsing ambiguous grammars
- Error Recovery in Parsing
- YACC programming specification

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Unit-II: Syntax Analysis (or) Parser
Top-Down Parsing

Outline:

- Top-Down Parsing-Introduction
- Recursive-Descent Parsing
- Recursive-descent parser with backtracking
- Difficulties in Top-Down Parser

Top-Down Parsing

Definition:

Construction of the **parse tree** starts at the root (from the start symbol) and proceeds towards leaves (tokens or terminals). i.e., construction of the parse tree is done from top to bottom of the tree.

(or)

Construction of the parse tree from the root (starting non-terminal) to the leaves in pre-order in depth-first search manner for the given input string.

Top-Down Parsing

- Thus, **top-down parsing** can be viewed as an attempt to find a **leftmost derivation** for an input string and constructing the **parse tree** for that derivation. The **parser** that uses this approach is called **top-down parsing**.

Top-Down Parsing

Example-1:

Show the **top-down parsing** process for the string **id+id*id** for the grammar given below:

E → E + E

E → E * E

E → (E)

E → id

Top-Down Parsing

Example: Solution

Leftmost Derivation (LMD):

Productions
E → E + E | E * E | (E) | id

$E \Rightarrow_{lm} E + E$

$E \Rightarrow_{lm} id + E[E \rightarrow id\ ]$

$E \Rightarrow_{lm} id + E * E\ [E \rightarrow E * E\ ]$

$E \Rightarrow_{lm} id + id * E\ [E \rightarrow id\ \ ]$

$E \Rightarrow_{lm} \mathbf{id + id * id}\ [E \rightarrow id\ \ ]$

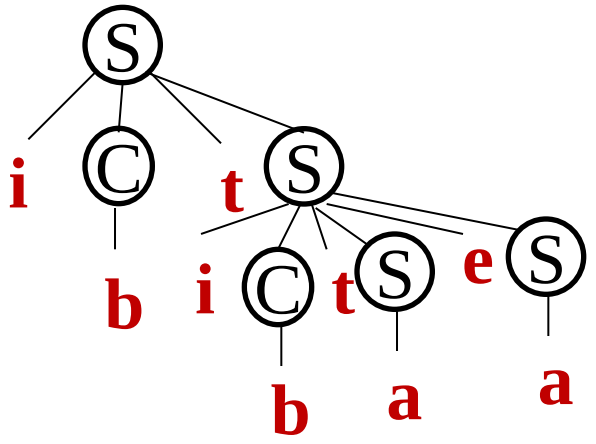


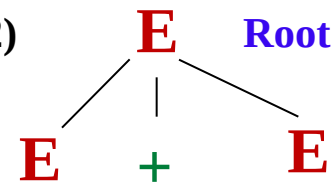
Fig: Parse tree for the LMD1 of string **ibtibtaea**

Top-Down Parsing

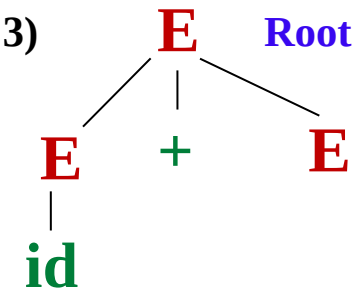
Example: Solution

1) E Root

2) E Root

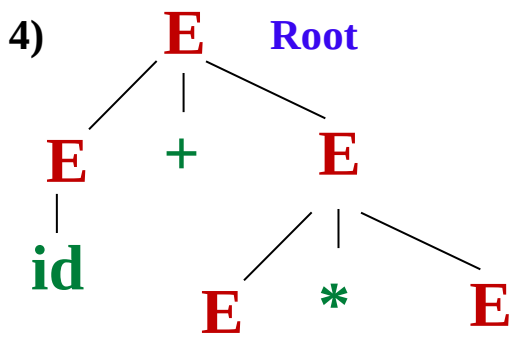


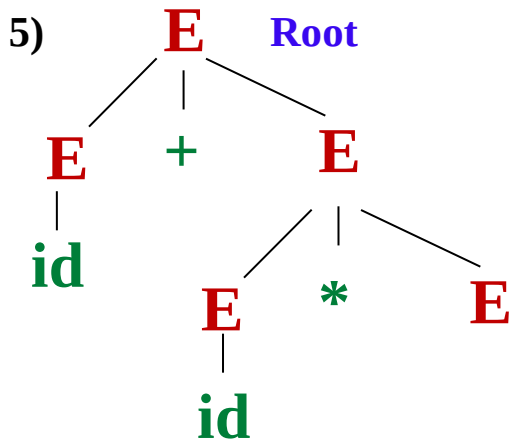
3) E Root



Top-Down Parsing

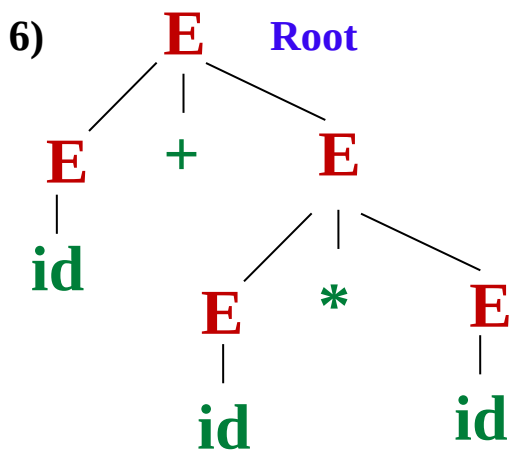
Example: Solution





Top-Down Parsing

Example: **Solution**



Top-Down Parsing

Practice Problem:

Show the **top-down parsing** process for the string **id+id*id** for the grammar given below:

E → TE′

E′ → +TE′ | ϵ

T → FT′

T′ → *FT′ | ϵ

F → (E)| id

Recursive-Descent Parsing

- A **recursive-descent parser** is a **top-down parser** in which **parse tree** is constructed from the top starting from **root node** and selecting the productions from the **left to right** (If two or more alternative productions exists).
- For every non-terminal there exist a **recursive procedure** and the right hand side of the production of that non-terminal is implemented as the body of the procedure.

Recursive-Descent Parsing

- The sequence of **terminals** and **non-terminals** on the right hand side of the production correspond to matching with input symbols and calls to other procedures while selecting the alternate production is implemented using **switch** or **if**-statements.
- For example, the procedure for the production **A → α** can be written as shown below:

```
Procedure A() //Function header
{
…    //body of the function
…
}
```

Recursive-Descent Parsing

General procedure for a recursive-descent parsing:

A typical procedure for a nonterminal in a top-down parser for the production $A \rightarrow X_1X_2X_3..X_k$ can be written as shown below (*backtracking is not supported*):

```
Procedure A() {
        Choose an A-production, A → X1X2X3..Xk
        for (i = 1 to k) {
            if (Xi is a nonterminal)
                call procedure Xi();
            else if (Xi equals the current input symbol a)
                advance the input to the next symbol;
            else /* an error has occurred */;
        }
}
```

Recursive-Descent Parsing

Example:

Write the recursive-descent parser procedure for the following grammar:

E → T

T → F

F → (E)| id

Recursive-Descent Parsing

Example: *Solution*

// Procedure corresponding to the production **E** → **T**

Procedure **E**()

{

T();

}

Recursive-Descent Parsing

Example: *Solution*

// Procedure corresponding to the production **T** → **F**

Procedure **T**()

{

F();

}

Recursive-Descent Parsing

Example: *Solution*

/* Procedure corresponding to the production **F → (E) | id** */

```
Procedure F() {
        if (input_symbol == '(')
        advance input_pointer;
        E();
        if (input_symbol == ')')
        advance input_pointer;
        else
                error();
        end if
        else if (input_symbol == 'id')
                advance input_pointer;
        else
        error();
        end if
        }
```

Recursive-Descent Parsing

Types of recursive-descent parsers:

1. A **`recursive-descent parser`** with **`backtracking`**
2. A **`recursive-descent parser`** without **`backtracking`** **(`Predictive Parser`)**

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

The **backtracking** is necessary for the following reasons:

1. During **parsing**, the productions are applied one by one. But if two or more alternative productions are there, they are applied in order from left to right one at a time.

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

2. When a particular production applied fails to expand the nonterminal properly, we have to apply the alternative production, it is necessary *undo* the activities done using the current production. This is possible only using **backtracking**.
3. But the **recursive-descent parser** with backtracking are not frequently used.

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example:

Show the steps involved in recursive-descent parser with backtracking for the input string **cad** for the following grammar:

S → cAd

A → ab | a

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example: *Solution*

Given grammar:

S → cAd

A → ab | a

String to be parsed:

cad

Parse tree:

S (root)

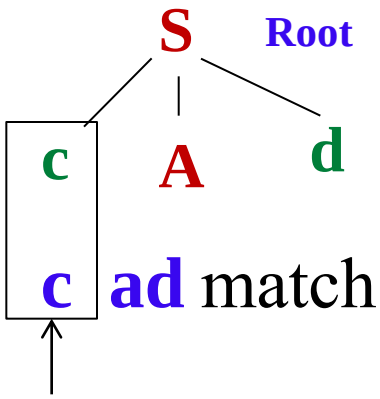
↑ *input pointer*

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example: *Solution*

Step-1:



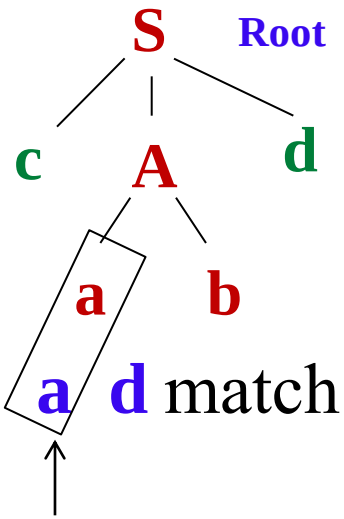
ad match '**c**' and increment *input_pointer*

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example: *Solution*

Step-2:



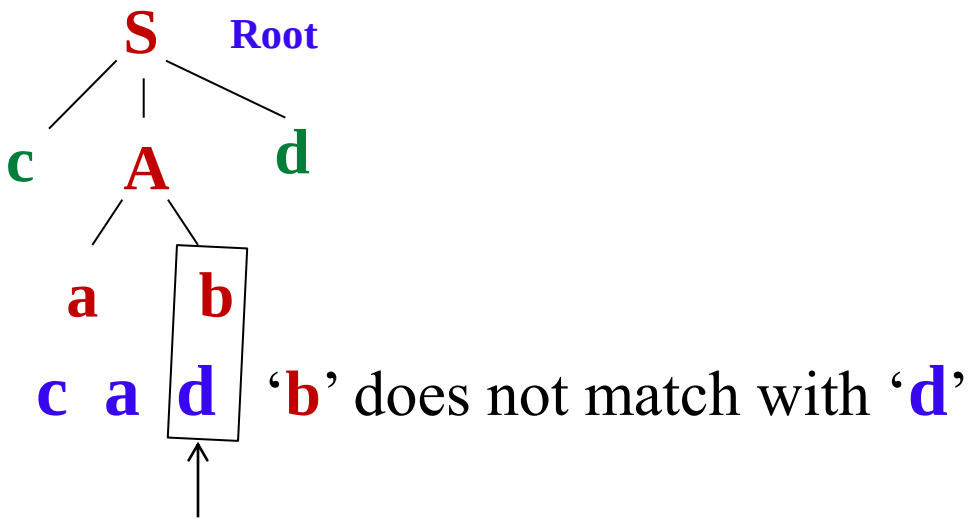
c a d match '**a**' and increment *input_pointer*

Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example: *Solution*

Step-3:



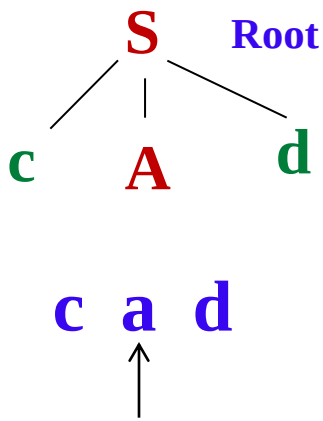
Recursive-Descent Parsing

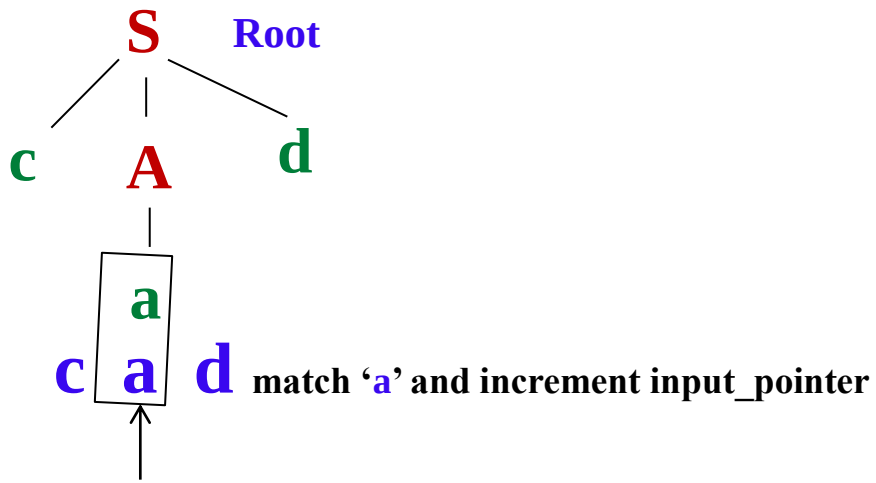
Recursive-descent parser with backtracking:

Example: *Solution*

Step-4:

By selecting **A → ab** the input string is not matched. So we have to reset the pointer to input symbol '**a**'. This is done using **backtracking** is shown below:





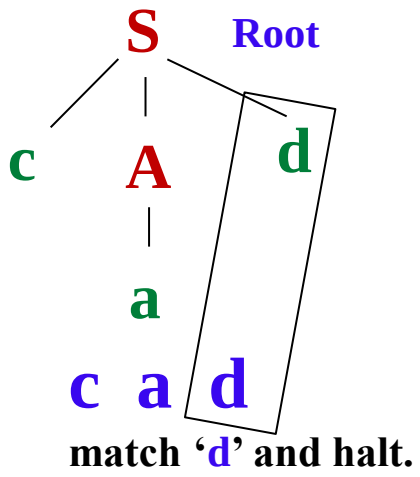
Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Example: *Solution*

Step-5:

Now the next symbol '**d**' in grammar is compared with '**d**' in the input and they *match*. Finally, we halt and announce **successful completion of parsing**.



Recursive-Descent Parsing

Recursive-descent parser with backtracking:

Recursive descent-parser cannot be constructed for a grammar having:

1. **Ambiguity:** The solution is to eliminate ambiguity from the grammar.
2. **Left recursion:** The solution is to eliminate left recursion from the grammar.
3. **Left factoring:** Two or more alternatives having a common prefix. The solution is to left factor the grammar.

Recursive-Descent Parsing

Difficulties in Top-Down Parser:

1. **Ambiguity:** Not suitable for top-down parser.
2. **Left recursion:** Not suitable for top-down parser.
3. **Non-left factored grammar:** Left factoring is must for top-down parser.
4. **Backtracking:** The backtracking is necessary for top-down parser.

Backtracking parsers are more powerful than **predictive parsers**, they are also much slower, requiring exponential time in general and therefore, **backtracking parsers** are not suitable for **practical compilers**.

Summary...

Top-Down Parsing : Recursive-Descent Parser

- Top-Down Parsing-Introduction
- Recursive-Descent Parsing
- Recursive-descent parser with backtracking
- Difficulties in Top-Down Parser

Reading: Aho2, Section 4.3

Next Lecture: Predictive parser (or) Non-Recursive-Descent Parser