Context free grammars

- Terminals
- Nonterminals
- Start symbol
- productions

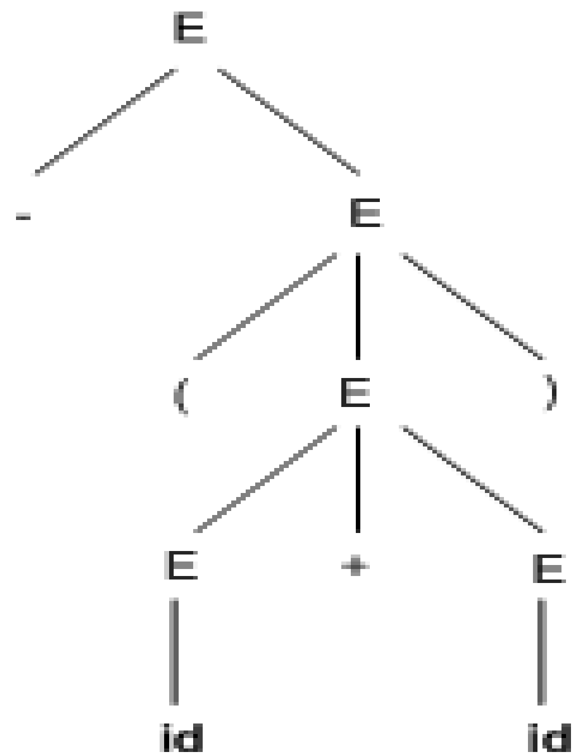
E --> E + T
E --> E – T
E --> T
T --> T * F
T --> T / F
T --> F
F --> (F)
F --> **id**

Derivations

- Productions are treated as rewriting rules to generate a string
- Rightmost and leftmost derivations
  - E --> E + E | E * E | -E | (E) | **id**
  - Derivations for **–(id+id)**
    - E => -E => **-**(E) => **-**(E+E) => **-**(**id**+E)=>**-**(**id**+**id**)

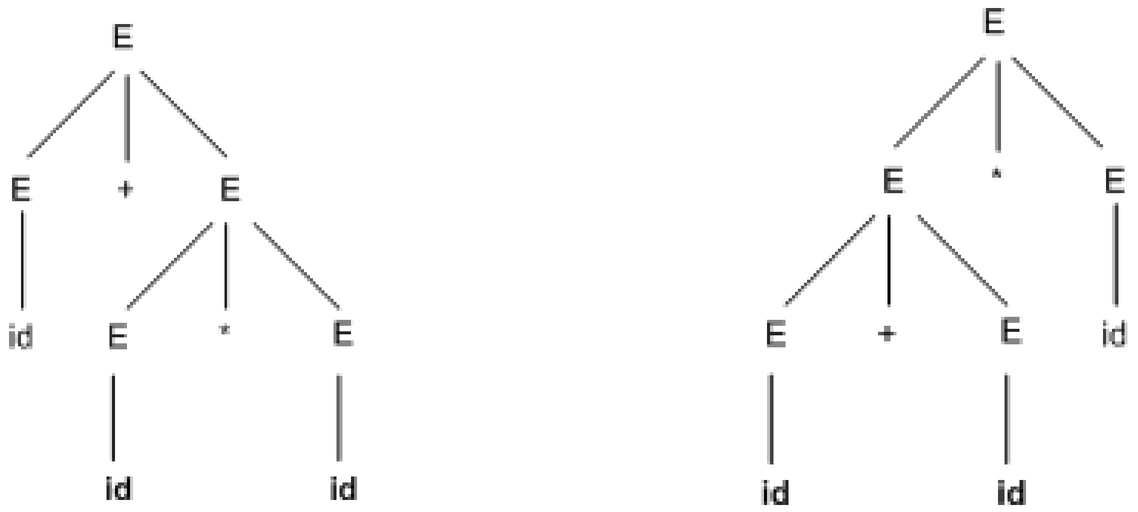
Parse trees

- **-(id+id)**
  - E => -E => -(E) => -(E+E) => -(**id**+E)=>-(**id**+**id**)



Ambiguity

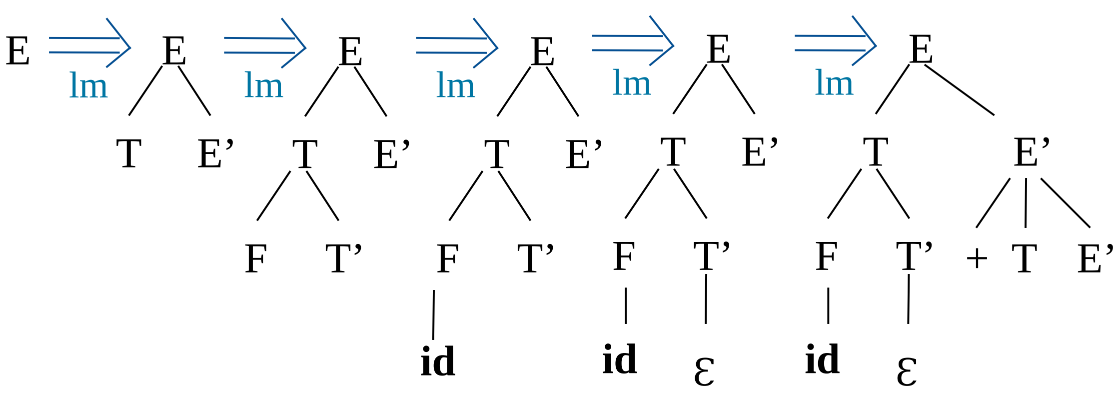
- For some strings there exist more than one parse tree
- Or more than one leftmost derivation
- Or more than one rightmost derivation
- Example: id+id*id



Introduction

- A Top-down parser tries to create a parse tree from the root towards the leafs scanning input from left to right
- It can be also viewed as finding a leftmost derivation for an input string
- Example: id+id*id

E -> TE'
E' -> +TE' | Ɛ
T -> FT'
T' -> *FT' | Ɛ
F -> (E) | **id**

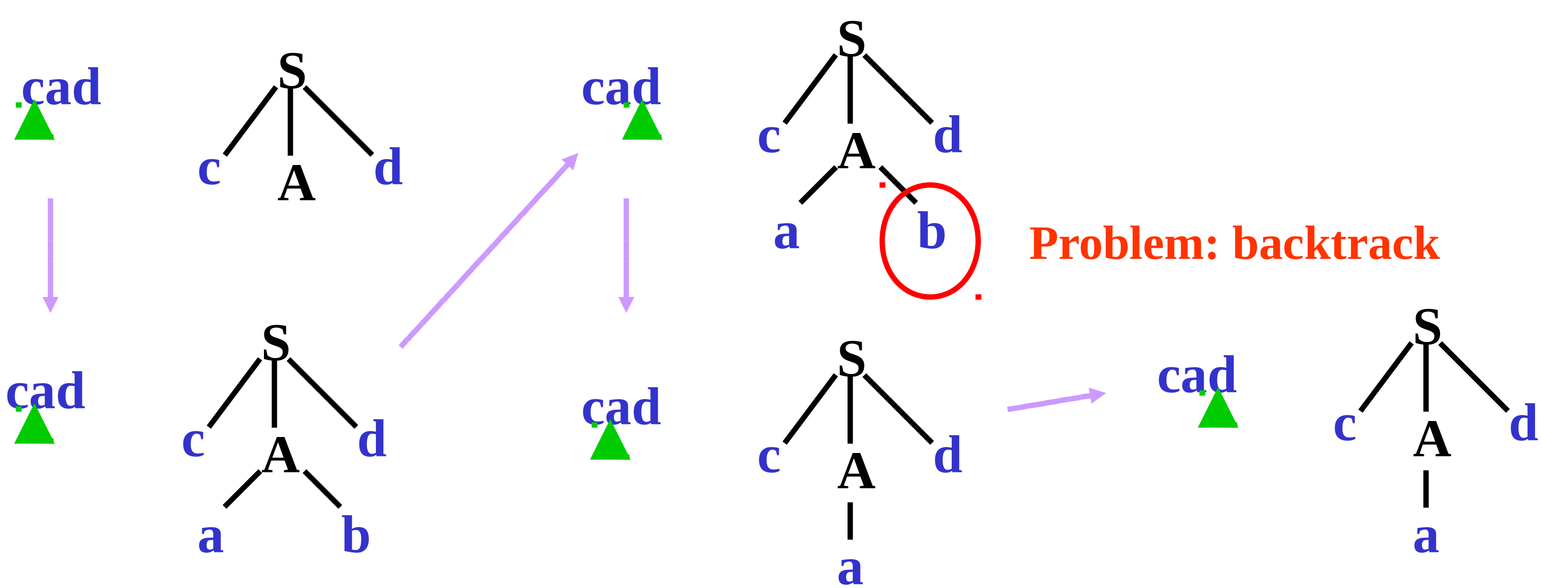


Top-Down Parsing

- Choose production rule based on input symbol
- May require backtracking to correct a wrong choice.
- Example:

$$\left.\begin{array}{l} S \rightarrow c\,A\,d \\ A \rightarrow ab \mid a \end{array}\right\}$$

input: cad



Parsing – Top-Down & Predictive

- Top-Down Parsing $\Rightarrow$ Parse tree / derivation of a token string occurs in a top down fashion.
- For Example, Consider:

Start symbol

$$
\begin{aligned}
type &\rightarrow simple \\
&\mid \uparrow \textbf{id} \\
&\mid \textbf{array} \; [\; simple \;] \; \textbf{of} \; type \\
simple &\rightarrow \textbf{integer} \\
&\mid \textbf{char} \\
&\mid \textbf{num dotdot num}
\end{aligned}
$$

Suppose input is :

array [**num dotdot num**] **of integer**

Parsing would begin with

type $\rightarrow$ **???**

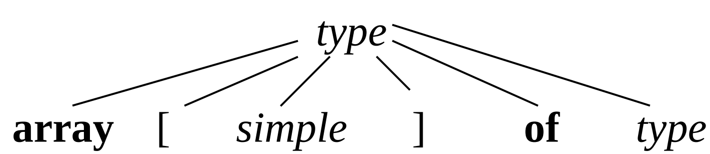
Top-Down Parse (type = start symbol)

Lookahead symbol

Input : **array** [**num dotdot num**] **of integer**

type

?



$type \rightarrow simple$ **Start symbol**

| ↑ **id**

| **array** [*simple*] **of** *type*

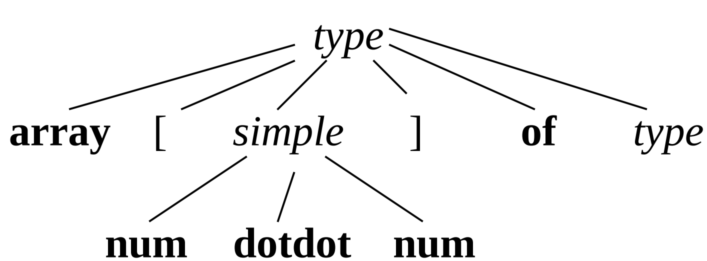
$simple \rightarrow$ **integer**

| **char**

| **num dotdot num**

Lookahead symbol

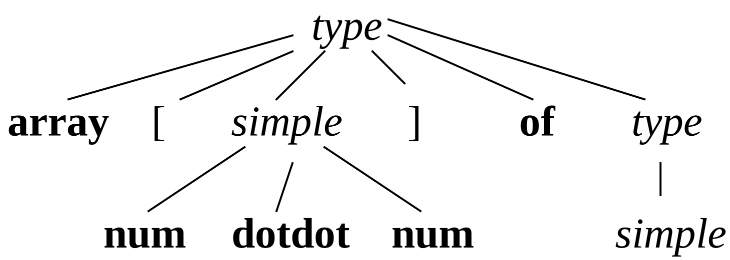
Input : **array** [**num dotdot num**] **of integer**

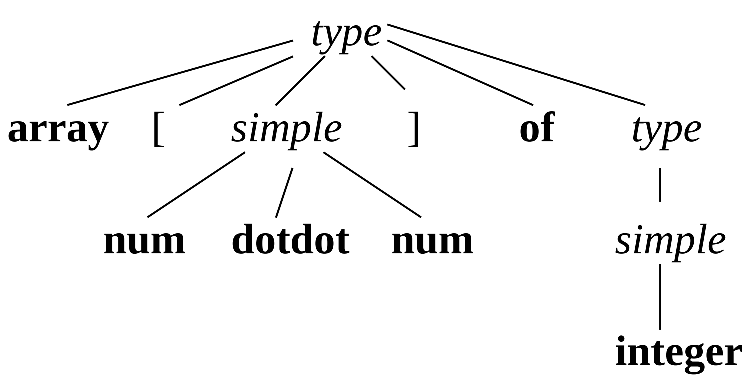


Top-Down Parse (*type* = start symbol)

Lookahead symbol

Input : **array** [**num dotdot num**] **of** **integer**





type → *simple* **Start symbol**

| ↑ **id**

| **array** [*simple*] **of** *type*

simple → **integer**

| **char**

| **num dotdot num**

Top-Down Parsing
Recursive Descent

- Parser Operates by Attempting to Match Tokens in the Input Stream

array [**num dotdot num**] **of** **integer**

type → *simple*
  | ↑ **id**
  | **array** [*simple*] **of** *type*
simple → **integer**
  | **char**
  | **num dotdot num**

procedure *match* (*t* : token) ;
begin
  if *lookahead* = *t* **then**
    lookahead := *nexttoken*
  else *error*
end ;

Recursive Descent (continued)

```
procedure simple ;
begin
    if lookahead = integer then match ( integer );
    else if lookahead = char then match ( char );
        else if lookahead = num then begin
            match (num); match (dotdot); match (num)
          end
          else error
end ;
```

type → *simple*
  | ↑ **id**
  | **array** [*simple*] **of** *type*

simple → **integer**
  | **char**
  | **num dotdot num**

Recursive Descent (continued)

procedure *type* ;
begin
    if *lookahead* is in { **integer, char, num** } **then** *simple*
    else if *lookahead* = '↑' **then** **begin** *match* ('↑') ; *match*(**id**) **end**
      else if *lookahead* = **array** **then** **begin**
        match(**array**); *match*('['); simple; *match*(']'); *match*(**of**); *type*
        end
        else *error*
end ;

type → *simple*
  | ↑ **id**
  | **array** [*simple*] **of** *type*

simple → **integer**
  | **char**
  | **num dotdot num**

How to write tests for selecting the appropriate production rule ?

Basic Tools:

First: **Let α be a string of grammar symbols. First(α) is the set that includes every terminal that appears leftmost in α or in any string originating from α.**
NOTE: If $\alpha \stackrel{*}{\Rightarrow} \in$, then $\in$ is First(α).

Follow: **Let A be a non-terminal. Follow(A) is the set of terminals *a* that can appear directly to the right of A in some sentential form. ($S \stackrel{*}{\Rightarrow} \alpha A a \beta$, for some α and β).**
NOTE: If $S \stackrel{*}{\Rightarrow} \alpha A$, then \$ is Follow(A).

Computing First(X) : All Grammar Symbols

1. **If X is a terminal, First(X) = {X}**
2. **If $X \to \in$ is a production rule, add $\in$ to First(X)**
3. **If X is a non-terminal, and $X \to Y_1 Y_2 \ldots Y_k$ is a production rule**

Place First(Y_1) - $\in$ in First(X)

if $Y_1 \overset{*}{\Rightarrow} \in$, Place First($Y_2$) - $\in$ in First(X)

if $Y_2 \overset{*}{\Rightarrow} \in$, Place First($Y_3$) - $\in$ in First(X)

…

if $Y_{k-1} \overset{*}{\Rightarrow} \in$, Place First($Y_k$) in First(X)

NOTE: As soon as $Y_i \overset{*}{\not\Rightarrow} \in$, Stop.

Repeat above steps until no more elements are added to any First( ) set.

Checking "$Y_j \overset{*}{\Rightarrow} \in$?" essentially amounts to checking whether $\in$ belongs to First(Y_j)

Computing First(X) : All Grammar Symbols - continued

Informally, suppose we want to compute

$First(X_1\ X_2 \ldots X_n) = First\ (X_1) - \in$ "+"

$First(X_2)$ if $\in$ is in $First(X_1) - \in$ "+"

$First(X_3)$ if $\in$ is in $First(X_2) - \in$ "+"

…

$First(X_n)$ if $\in$ is in $First(X_{n-1})$

Note 1: Only add $\in$ to $First(X_1\ X_2 \ldots X_n)$ if $\in$ is in $First(X_i)$ for all i

Note 2: For $First(X_1)$, if $X_1 \rightarrow Z_1\ Z_2 \ldots Z_m$, then we need to compute $First(Z_1\ Z_2 \ldots Z_m)$!

Example 1

Given the production rules:

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

Example 1

Given the production rules:

$S \rightarrow$ i E t SS' | a

S' $\rightarrow$ eS | $\in$

$E \rightarrow$ b

Verify that

First(S) = { i, a }

First(S') = { e, $\in$ }

First(E) = { b }

Example 2

Computing First for:

$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T' \rightarrow * FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

Example 2

Computing First for:

$E \to TE'$

$E' \to +\ TE' \mid \in$

$T \to FT'$

$T' \to *\ FT' \mid \in$

$F \to (\ E\ ) \mid id$

First(E) → **First(TE')** → **First(T) "+" First(E')**

Not First(E') since $T \stackrel{*}{\not\Rightarrow} \in$

First(TE') → **First(T)**

First(T) → **First(F) "+" First(T')** → **First(F)**

Not First(T') since $F \stackrel{*}{\not\Rightarrow} \in$

First(F) → **First((E)) "+" First(id)** → **"(" and "id"**

Overall:	First(E) = { (, id } = First(F)
	First(E') = { + , ∈ } First(T') = { * , ∈ }
	First(T) → First(F) = { (, id }

Computing Follow(A) : All Non-Terminals

1. **Place \$ in Follow(A), where A is the start symbol and \$ signals end of input**

2. **If there is a production $B \rightarrow \alpha A \beta$, then everything in First(β) is in Follow(A) except for $\in$.**

3. **If $B \rightarrow \alpha A$ is a production, or $B \rightarrow \alpha A \beta$ and $\beta \overset{*}{\Rightarrow} \in$ (First(β) contains $\in$), then everything in Follow(B) is in Follow(A)**

   (Whatever followed B must follow A, since nothing follows A from the production rule)

We'll calculate Follow for two grammars.

The Algorithm for Follow – pseudocode

1. Initialize Follow(X) for all non-terminals X to empty set. Place $ in Follow(S), where S is the start NT.
2. Repeat the following step until no modifications are made to any Follow-set

   For any production $\mathbf{X} \rightarrow \mathbf{X_1\ X_2 \ldots X_m}$

   For j=1 to m,

   if $\mathbf{X_j}$ is a non-terminal then:

Follow($\mathbf{X_j}$)=Follow($\mathbf{X_j}$)$\cup$(**First($\mathbf{X_{j+1},\ldots,X_m}$)-{ε});**
If **First($\mathbf{X_{j+1},\ldots,X_m}$) contains ε or $\mathbf{X_{j+1},\ldots,X_m = \varepsilon}$**
then Follow($\mathbf{X_j}$)=Follow($\mathbf{X_j}$)$\cup$ Follow(**X**);

Computing Follow : 1st Example

Recall:

$S \rightarrow$ i E t SS' | a	First(S) = { i, a }
$S' \rightarrow$ eS | $\in$	First(S') = { e, $\in$ }
$E \rightarrow$ b	First(E) = { b }

Computing Follow : 1st Example

Recall:

$S \rightarrow$ i E t SS' | a   **First(S) = { i, a }**

$S' \rightarrow$ eS | $\in$ **First(S') = { e, $\in$ }**

$E \rightarrow$ b   **First(E) = { b }**

Follow(S) – Contains \$, since S is start symbol

Since $S \rightarrow$ i E t SS' , put in First(S') – not $\in$

Since $S' \overset{*}{\Rightarrow} \in$, Put in Follow(S)

Since $S' \rightarrow$ eS, put in Follow(S')   **So…. Follow(S) = { e, \$ }**

Follow(S') = Follow(S) HOW?

Follow(E) = { t }

Example 2

Compute Follow for:

E → TE’

T → FT’

E’ → + TE’ | ∈

F → (E) | id

T’ → * FT’ | ∈

Example 2

Compute Follow for:

E → TE'

T → FT'

E' → + TE' | ∈

F → (E) | id

T' → * FT' | ∈

	First
E	(id
E'	∈ +
T	(id
T'	∈ *
F	(id

	Follow
E	$)
E'	$)
T	+ $)
T'	+ $)
F	+ * $)