Top Down Parsers

1

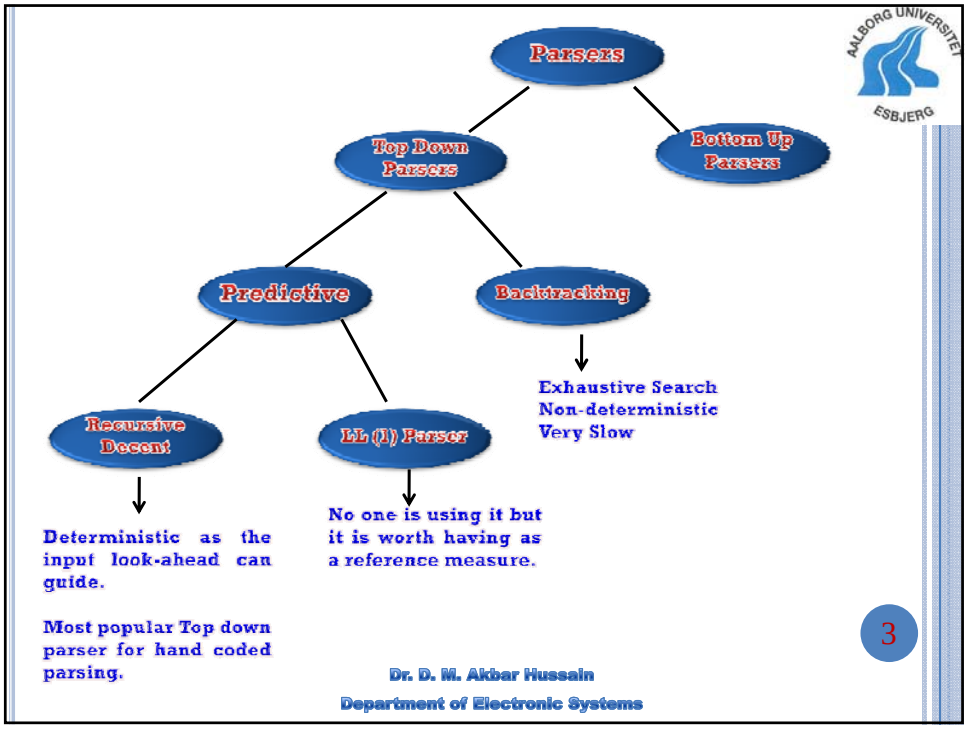
Dr. D. M. Akbar Hussain
Department of Electronic Systems

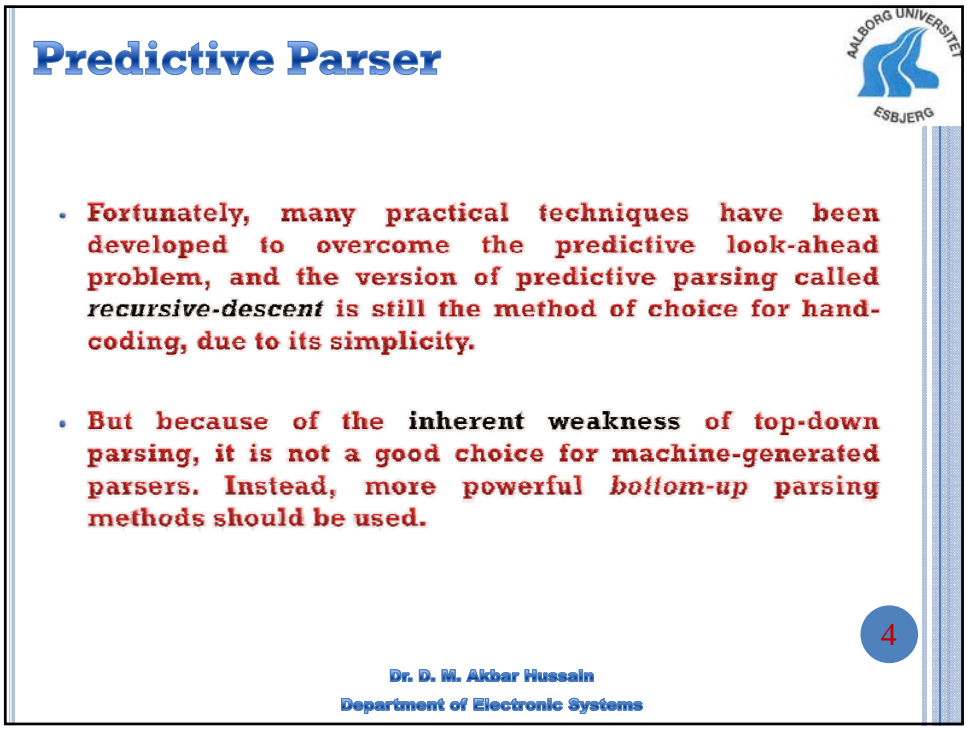
Top Down Parsers

- ***Predictive parsers:*** That try to make decisions about the structure of the tree below a node based on a few look-ahead tokens (usually one!). This is a weakness, since little program structure has been seen before predictive decisions must be made.
- ***Backtracking parsers:*** That solve the look-ahead problem by backtracking if one decision turns out to be wrong and making a different choice. But such parsers are slow (exponential time in general).

2

Dr. D. M. Akbar Hussain
Department of Electronic Systems


Parsers
Top Down Parsers
Bottom Up Parsers
Predictive
Backtracking
Recursive Decent
LL (1) Parser
Exhaustive Search
Non-deterministic
Very Slow
Deterministic as the input look-ahead can guide.
Most popular Top down parser for hand coded parsing.
No one is using it but it is worth having as a reference measure.
3
Dr. D. M. Akbar Hussain
Department of Electronic Systems


Predictive Parser
Fortunately, many practical techniques have been developed to overcome the predictive look-ahead problem, and the version of predictive parsing called recursive-descent is still the method of choice for hand-coding, due to its simplicity.
But because of the inherent weakness of top-down parsing, it is not a good choice for machine-generated parsers. Instead, more powerful bottom-up parsing methods should be used.
4
Dr. D. M. Akbar Hussain
Department of Electronic Systems

Recursive Decent Parser

Simple Idea:

- For each non-terminal a Function/Procedure is used.
- Detail (structure) of the procedure is determined by the right hand side of the rule.
- Choice can be interpreted as control "if" or "case" statement, choice between terminals and non-terminals corresponds to matching of input or calls to other procedures.

5

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Recursive Decent Parser

$S \rightarrow bA \mid c$
$A \rightarrow dS \mid e$

So we have two NT, we write two Procedure/Functions (Pseudo-ops):

```
Boolean S_function ()
        {
        if Token_Is == 'b')
                Try/Apply  S ---> bA
                Call A_function ();
        else if Token_Is == 'c'
                Token_Is = 'c';
        else
                fail ();
                return FALSE;
        }
```

```
Boolean A_function ()
        {
        if Token_Is == 'd')
                Try/Apply A ---> dS
                Call S_function ();
        else if Token_Is == 'e'
                Token_Is = 'e';
        else
                fail ();
                return FALSE;
        }
```

6

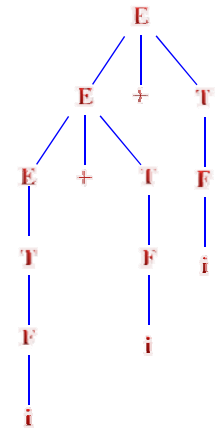
Dr. D. M. Akbar Hussain
Department of Electronic Systems

Left Recursion

Top down parser cannot handle left recursion.
Example:

$E \rightarrow E + T \mid T$
$E \rightarrow T * F \mid F$
$F \rightarrow i \mid (E)$

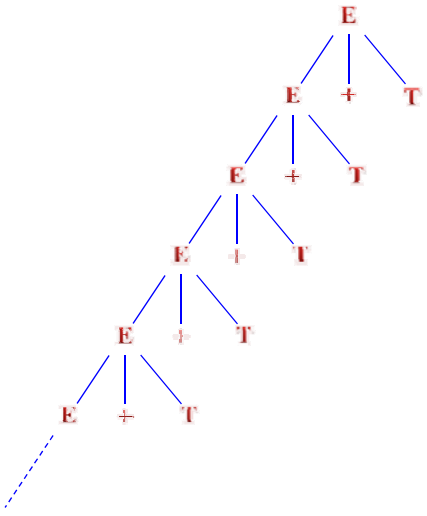
String: i + i + i



7

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Left Recursion Continue

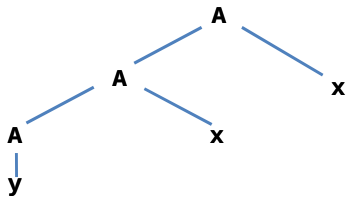


8

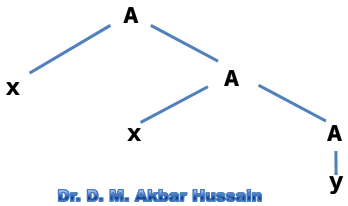
Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: Left & Right Recursion

- yxx: from Left recursion: A → A x | y



- xxy: Right recursion: A → x A | y

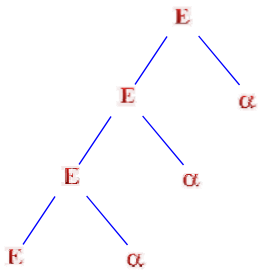


9

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Immediate Left Recursion

Example: E → E α



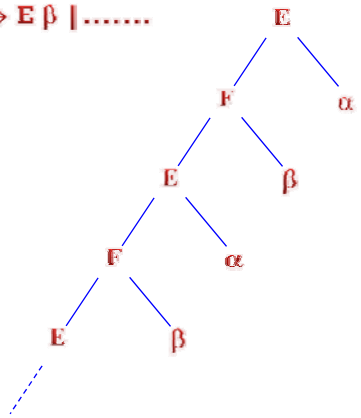
10

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Indirect Left Recursion

Example: $E \rightarrow F\,\alpha \mid \ldots\ldots$

$F \rightarrow E\,\beta \mid \ldots\ldots$



11

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Removing Left Recursion

Rules to Remove Left Recursion:

1. Separate the left recursive production from non left recursive ones;

   $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots\ldots$ (left recursive)

   $A \rightarrow \gamma_1 \mid \gamma_2 \mid \gamma_3 \mid \ldots\ldots$ (non left recursive)

2. Introduce a new non-terminals A'
3. Change non left recursive productions as

   $A \rightarrow \gamma_1 A' \mid \gamma_2 A' \mid \gamma_3 A' \mid \ldots\ldots$

4. Change left recursive productions as

   $A' \rightarrow \varepsilon \mid \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots\ldots$

12

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: S → S a | b

Grammar: S ⇒ Sa ⇒ Saa ⇒ Saaa ⇒ ba*

Apply Rules to Remove Left Recursion:

1. S → Sa (left recursive)
   S → b (non left recursive)
2. S'
3. S → bS'
4. S' → aS' | ε

Grammar: S ⇒ bS' ⇒ baS' ⇒ baaS' ⇒ ba*

13

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: A → Ac | Ad | e | f

1. A → Ac | Ad (left recursive)
   A → e | f (non left recursive)
2. A'
3. A → eA' | f A'
4. A' → cA' | dA' | ε

14

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example:

$S \rightarrow A|B|Se|dS$
$A \rightarrow Bd|cA|f$
$B \rightarrow Se|Ad|g$

Left Recursive

As we see First rule is left recursive so we remove that first by introducing a new NT symbol X.

- $S \rightarrow AX|BX|dSX$ and $X \rightarrow eX|\varepsilon$

Now we should replace S in the existing productions because it has been changed, however there will be no change in A production rules as there is no S.

- $B \rightarrow AXe|BXe|dSXe|Ad|g$

15

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Continue Example:

- $B \rightarrow AXe|BXe|dSXe|Ad|g$

We replace A in B production rules as well:

- $B \rightarrow BdXe|cAXe|fXe|BXe|dSXe|Bdd|cAd|fd|g$

The problem is now B became left recursive so let us remove it, introduce a new symbol (NT) Y:

- $B \rightarrow cAXeY|fXeY|dSXeY|cAdY|fdY|gY$
- $Y \rightarrow dXeY|XeY|ddY|\varepsilon$

16

Dr. D. M. Akbar Hussain
Department of Electronic Systems

General Rules for Multiple Rules

1. List all non-terminals
2. Go through each non-terminals in sequence and for each non-terminal:

a) If right hand side begins with a non-terminal e.g.,

$A \rightarrow A\beta$ and if there are

$A \rightarrow \gamma_1 \mid \gamma_2 \mid \gamma_3 \mid \ldots\ldots$ Then substitute

$A \rightarrow \gamma_1\beta \mid \gamma_2\beta \mid \gamma_3\beta \mid \ldots\ldots$

b) If right hand side begins with any thing else do nothing.

17

Dr. D. M. Akbar Hussain
Department of Electronic Systems

General Rules for Multiple Rules

Example: $S \rightarrow aA \mid b \mid cS$

$A \rightarrow Sd \mid e$

1: Non-terminals: S, A

2: For S → productions nothing is required, for A → (a) is applicable

Final: $A \rightarrow aAd \mid bd \mid cSd \mid e$

18

Dr. D. M. Akbar Hussain
Department of Electronic Systems

First Set and Follow Set

FIRST SET

If α begins with a terminal x, then FIRST (α) = x.
If α =>* ε then FIRST (α) includes ε.
FIRST (ε) = {ε}.
If α begins with a non-terminal A, then FIRST (α) includes FIRST (A) – {ε}.

FOLLOW SET (Tokens that can "follow" a symbol)

If A is the starting symbol, then put the end marker $ into FOLLOW (A).
Look through the grammar rules for all the appearances of non-terminal A on the right hand side of productions, consider for example T → xAb.

1. If b begins with a terminal t, then t is in FOLLOW (A).
2. Otherwise, FOLLOW (A) includes FIRST (b) – {ε}.
3. If b=ε, meaning A is at the end, or b is null able, then include FOLLOW (T) in FOLLOW (A).

19

Dr. D. M. Akbar Hussain
Department of Electronic Systems

First Set and Follow Set

$$E \rightarrow T\ Q$$
$$Q \rightarrow +\ T\ Q \mid -\ T\ Q \mid \varepsilon$$
$$T \rightarrow F\ R$$
$$R \rightarrow *\ F\ R \mid /\ F\ R \mid \varepsilon$$
$$F \rightarrow (E) \mid i$$

20

Dr. D. M. Akbar Hussain
Department of Electronic Systems

LL(1) Parsing Method

S → (S) S | ε

$ Indicate end (bottom of stack) and input is augmented with $.

Parsing Stack	Input	Action
$ S	( ) $	*s* → (s)s
$ S) S (	( ) $	”(” match
$ S) S	) $	*s* → ε
$ S)	) $	”)” match
$ S	$	*s* → ε
$	$	accept

21

Dr. D. M. Akbar Hussain
Department of Electronic Systems

LL(1) Parsing Method

S → aABb A → c | ε B → d | ε

Example: acdb

Parsing Stack	Input	Action
$ S	acdb $	*s* → aABb
$ bBAa	acdb $	Pop a
$ bBA	cdb $	*A* → c
$ bBc	cdb $	Pop c
$ bB	db $	*B* → d
$ bd	d b $	Pop d
$ b	b $	Pop b
$	$	Success

22

Dr. D. M. Akbar Hussain
Department of Electronic Systems

LL (1) Parsing Table

SCAN THROUGH ALL PRODUCTIONS

Suppose $X \rightarrow \alpha$

1. For all terminals (a) in FIRST (α) except ε, there is table entry:

   Table [X, a] = α

2. If $\alpha = \varepsilon$ or if FIRST (α) has ε then all terminals (a) in FOLLOW (X), there is table entry:

   Table [X, a] = ε

23

LL(1) Parsing Method

$S \rightarrow (S)\ S \mid \varepsilon$

First (S) = First ((S)S) U First (ε) = { (, ε }

Follow(S) = First ( )S) = First( )) = {), $ }

Note: Follow S contains $ as it is the start symbol

N \ T	(	)	$
S	$S \rightarrow (S)S$	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$

24

Table Driven LL(1) Parser

$S \rightarrow aABb \quad A \rightarrow c \mid \varepsilon \quad B \rightarrow d \mid \varepsilon$ **Example: acdb**

First (S) = First (aABb) = {a}

First (A) = First (c) U First (ε) = {c, ε }

First (B) = First (d) U First (ε) = {d, ε }

Follow (A) = First (Bb) = First (B) – First (ε) U First (b) = {d, b}

Follow (B) = First (b) = {b}

N \ T	a	b	c	d	$
S	$S \rightarrow aABb$				
A		$A \rightarrow \varepsilon$	$A \rightarrow c$	$A \rightarrow \varepsilon$	
B		$B \rightarrow \varepsilon$		$B \rightarrow d$	

25

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Error Recovery in Parsers

- A parser should try to determine that an error has occurred *as soon as possible*. Waiting too long before declaring error means the location of the actual error may have been lost.
- After an error has occurred, the parser must pick a likely place to resume the parse. A parser should always try to parse as much of the code as possible, in order to find as many real errors as possible during a single translation.
- A parser should try to avoid the error cascade problem, in which one error generates a lengthy sequence of spurious error messages.
- A parser must avoid infinite loops on errors, in which an unending cascade of error messages is generated without consuming any input.

26

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Panic Mode in Recursive Decent

- Extra parameter consisting of a set of *synchronizing tokens.*
- As parsing proceeds, tokens that may function as synchronizing tokens are added to the synchronizing set as each call occurs.
- If an error is encountered, the parser scans ahead, throwing away tokens until one of the synchronizing set of tokens is seen in the input, whence parsing is resumed.

27

Dr. D. M. Akbar Hussain
Department of Electronic Systems