

### **Top Down Depth-First Parsing**

Parsing a Grammar can be viewed as a graph or tree search problem. Generally, the parse is either leftmost or rightmost. Top down searches starting from the start start symbol. It then tries to apply each rule IN ORDER.

If the rule generates the wrong thing, then the search back-tracks.

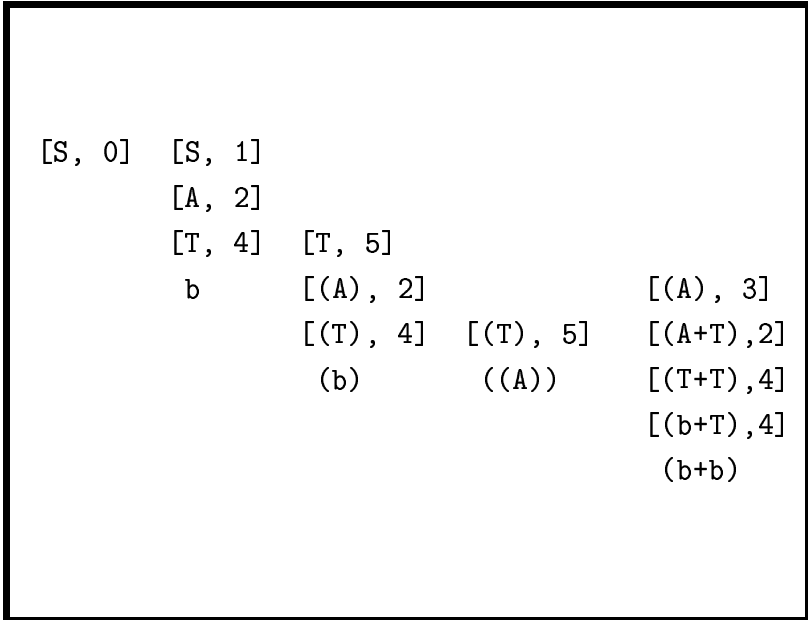
**Slide Lecture 2 -28**

TOP DOWN Depth First Search Using a Stack to generate (b+b)

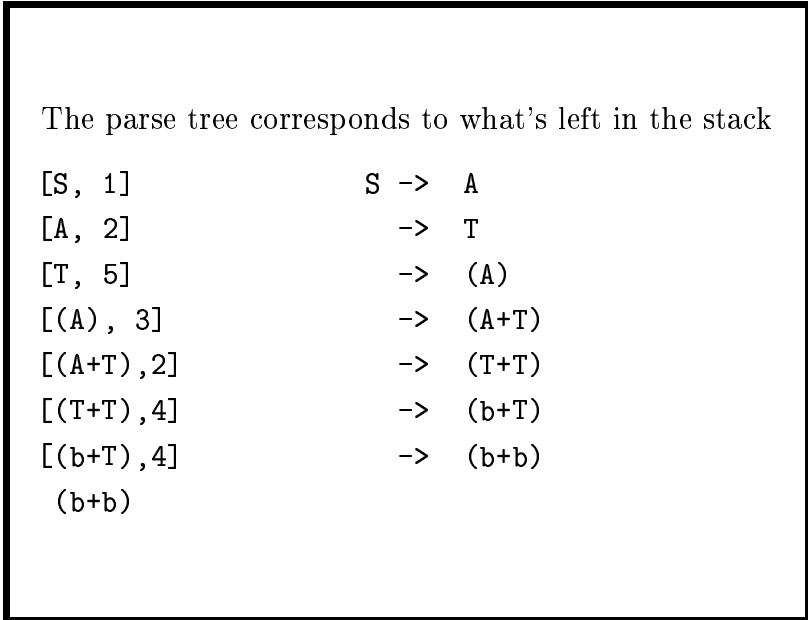
The Grammar:

1.  $S \rightarrow A$
2.  $A \rightarrow T$
3.  $A \rightarrow A+T$
4.  $T \rightarrow b$
5.  $T \rightarrow (A)$

**Slide Lecture 2 -29**



Slide Lecture 2 -30



Slide Lecture 2 -31

The Danger of Depth First Search  
DIFFERENT grammar with the following 4 rules.  
Generate: a+a

1.  $S \rightarrow A$  [S, 0] [S, 1]
2.  $A \rightarrow A+T$  [A, 2]
3.  $A \rightarrow a$  [A+T, 2]
4.  $T \rightarrow b$  [A+T+T, 2]  
[A+T+T+T, 2]  
... to infinity  
(and beyond :-)

Slide Lecture 2 -32

We could change the order of the rules. But now try  
to generate b+b which is not in the language.

1.  $S \rightarrow A$
2.  $A \rightarrow a$
3.  $A \rightarrow A+T$
4.  $T \rightarrow b$

We initialize the stack with  
[S, 0]

and immediately go to  
[S, 0] [S, 1]

Slide Lecture 2 -33

```

[S, 1]
[A, 2]  [A, 3]
a      [A+T, 2]  [A+T, 3]
       [a+T, 4]  [A+T+T, 2]  [A+T+T, 3]
                [a+T+T, 4]  [A+T+T+T, 2]
                        [a+T+T+T, 4]
                                a+T+T+T+T+...
                                    ... to infinity

```

Slide Lecture 2 -34

A standard problem with DEPTH FIRST SEARCH is that it can result in infinite searches if there are infinite branches in the search tree.

BREATH FIRST SEARCH will always find an answer and halt \*if\* there is an answer.

Slide Lecture 2 -35

Next we look at BOTTOM UP Parsing.  
Back to our original grammar.

1.  $S \rightarrow A$
2.  $A \rightarrow T$
3.  $A \rightarrow A+T$
4.  $T \rightarrow b$
5.  $T \rightarrow (A)$

Slide Lecture 2 -36

Parse (b)+b

|       |                     |
|-------|---------------------|
| (b)+b |                     |
| (T)+b | $T \rightarrow b$   |
| (A)+b | $A \rightarrow T$   |
| T+b   | $T \rightarrow (A)$ |
| A+b   | $A \rightarrow T$   |
| A+T   | $T \rightarrow b$   |
| A     | $A \rightarrow A+T$ |
| S     | $S \rightarrow A$   |

Slide Lecture 2 -37

Doing a leftmost bottom up matching process results  
in a right most parse

| RULE     | PARSE |
|----------|-------|
| S -> A   | A     |
| A -> A+T | A+T   |
| T -> b   | A+b   |
| A -> T   | T+b   |
| T -> (A) | (A)+b |
| A -> T   | (T)+b |
| T -> b   | (b)+b |

Slide Lecture 2 -38

SHIFTING to find the set of next possible rule applications. Let \* be the empty string. Initialize U = \* V is the current state of the parse

|       | U     | V     | Rule     | Reduction |
|-------|-------|-------|----------|-----------|
|       | *     | (A+T) |          |           |
| shift | (     | A+T)  |          |           |
| shift | (A    | +T)   | S -> A   | (S+T)     |
| shift | (A+   | T)    |          |           |
| shift | (A+T  | )     | A -> A+T | (A)       |
|       |       |       | A -> T   | (A+A)     |
| shift | (A+T) | *     |          |           |

Slide Lecture 2 -39

THESE are all possible reductions for this state. Can be used for Breadth first parsing.

**Slide Lecture 2 -40**

DEPTH-FIRST First Bottom Up using a STACK

Assume the start symbol is not recursive.

Let \* be the empty string

[X,Y,Z] and U,i,V represent: [scanned, rule,  
unscanned]

This represents the state of the parse at any point.  
POP cause the parse to either start or restart. Only  
reduce and pop update the stack.

**Slide Lecture 2 -41**

We try to match a rule to U.

0. Start
1. If possible, REDUCE, go to Start.  
>> A reduction moves [U,rule,V] onto stack
2. If you cannot reduce, SHIFT, and go to Start.  
>> Shifts tokens from V to U.
3. If you cannot shift, POP, and go to Start.  
>> Only Pop reset the "i" variable.
4. If you cannot pop, then FAIL.

Slide Lecture 2 -42

| OP     | STACK       | U  | i | V     |
|--------|-------------|----|---|-------|
|        | [*,0,(b+b)] |    |   |       |
| pop    |             | *  | 0 | (b+b) |
| shift  |             | (  | 0 | b+b)  |
| shift  |             | (b | 0 | +b)   |
| reduce | [(b,4,+b)]  | (T | 0 | +b)   |
|        | [(T,2,+b)]  |    |   |       |
| reduce | [(b,4,+b)]  | (A | 0 | +b)   |

Slide Lecture 2 -43

NOTE S-> A does not reduce V to \*.

shift (A+ 0 b)

shift (A+b 0 )

[(T,2,+b)]

reduce [(b,4,+b)] (A 0 +b)

Slide Lecture 2 -44