Top-Down Parsing

$S \to aAbB$
$A \to aA \mid \epsilon$
$B \to bB \mid a$

$S \to aAbB$
1 $\to aaAbB$
2 $\to aacbB$
3 $\to aacbbB$
4 $\to aacbba$

Top down approach
LMD

1
aacbba 4
2 3

A recursive distance parser - top down approach
~~pt~~ push-down automata

aAbB

Bottom up → from leaves to root
RMD

$S \to aAbB$ →
$A \to aA \mid c$
$B \to bB \mid a$

aAbB
aAbbB
aAbba
aaAbba
aacbba
RMD

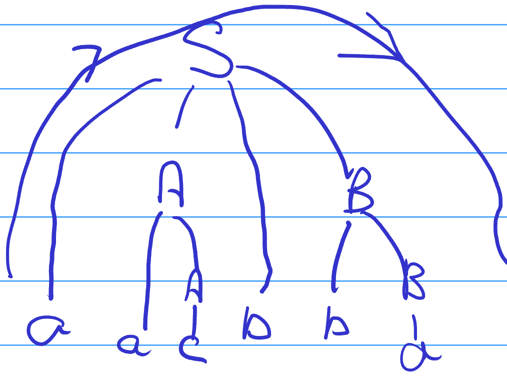
$S \to aAbB$
$B \to bB$
$B \to a$
$A \to aA$
$A \to c$

} You can derive topdown yeah

aacbba

(Bottom-up can be more complex)

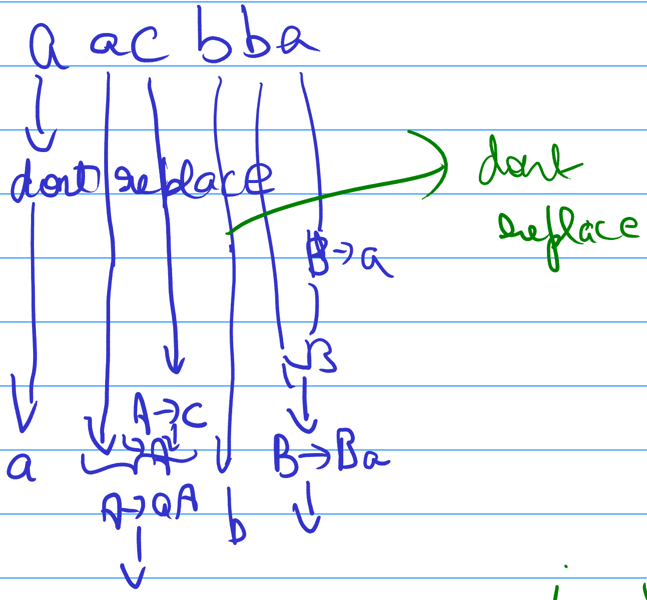
Exactly the reverse of RMD ←



consume 'a' (?)
~~a~~ go to next symbol

There are algos to resolve this

[we dunno what exactly to replace]



a a c b b a

i cant replace this with B or replace this at all, we cant easily know

Only those substrings when chars/substrings are replaced with give the previous sentencial form in RMD those are handles.

Common family : LR parsers

L → Left to right

R → RMD (in reverse order, since Left to right)

(Usually complexity of general parsing is $O(n^3)$

But less general ones used in compilers can do it in $O(n)$

Recursive descent

```
lexeme ()
S () {
    if (nextToken = 'a')
    {
        lexeme()
        A()
        lexeme()
        if (nextToken = 'b')
        { lexeme()
          B()
        }
    }
}
```

returns ~~to~~ the next lexeme (token actually) that return value is stored in nextToken

returns

aac bba

```
A {
    if(next token = 'a')
    {
        lexeme()
        A() }
    else if (next Token = 'c');
}

B()
{ if (next token = 'b')
    lexeme()
    B()
  else if(nexttoken='a');
}
```

READ IB