Parsing

- Parsing is the process of constructing a parse tree for a sentence generated by a given grammar
- If there are no restrictions on the language and the form of grammar used, parsers for context-free languages require $O(n^3)$ time (n being the length of the string parsed)
  - Cocke-Younger-Kasami's algorithm
  - Earley's algorithm
- Subsets of context-free languages typically require $O(n)$ time
  - Predictive parsing using $LL(1)$ grammars (top-down parsing method)
  - Shift-Reduce parsing using $LR(1)$ grammars (bottom-up parsing method)

Top-Down Parsing using LL Grammars

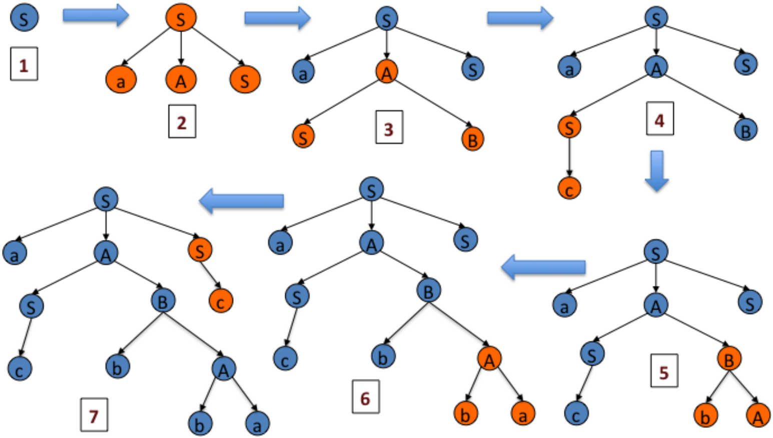
- Top-down parsing using predictive parsing, traces the left-most derivation of the string while constructing the parse tree
- Starts from the start symbol of the grammar, and “predicts” the next production used in the derivation
- Such “prediction” is aided by parsing tables (constructed off-line)
- The next production to be used in the derivation is determined using the next input symbol to lookup the parsing table (look-ahead symbol)
- Placing restrictions on the grammar ensures that no slot in the parsing table contains more than one production
- At the time of parsing table constrcution, if two productions become eligible to be placed in the same slot of the parsing table, the grammar is declared unfit for predictive parsing

Top-Down LL-Parsing Example

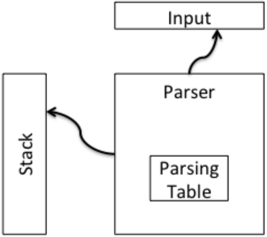
S → aAS | c
A → ba | SB
B → bA | S

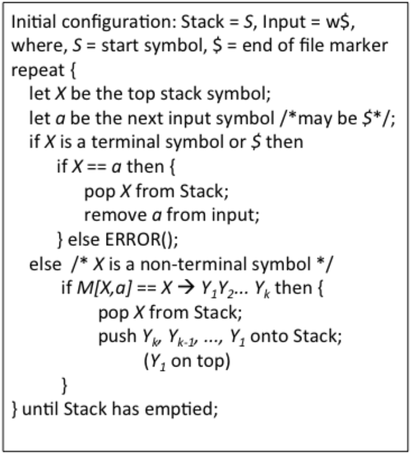
Leftmost derivation of the string *acbbac*

S => aAS => aSBS => acBS => acbAS => acbbaS => acbbac
1 2 3 4 5 6 7



LL(1) Parsing Algorithm





```
Initial configuration: Stack = $S$, Input = w$,
where, $S$ = start symbol, $ = end of file marker
repeat {
   let $X$ be the top stack symbol;
   let $a$ be the next input symbol /*may be $*/;
   if $X$ is a terminal symbol or $ then
        if $X$ == $a$ then {
             pop $X$ from Stack;
             remove $a$ from input;
        } else ERROR();
   else  /* $X$ is a non-terminal symbol */
         if $M[X,a]$ == $X \rightarrow Y_1Y_2... Y_k$ then {
              pop $X$ from Stack;
              push $Y_k, Y_{k-1}, ..., Y_1$ onto Stack;
                      ($Y_1$ on top)
         }
} until Stack has emptied;
```

LL(1) Parsing Algorithm Example

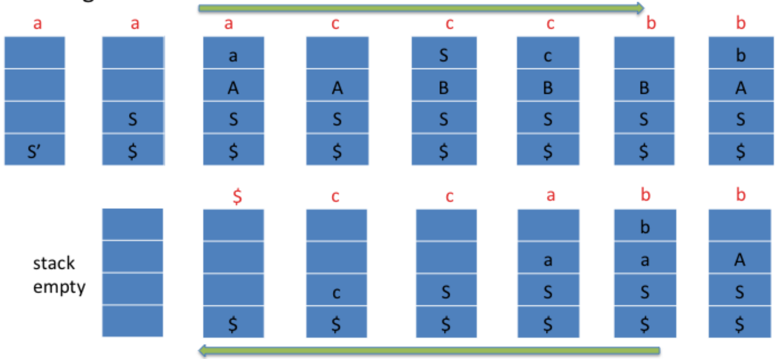
Grammar

S’ → S$
S → aAS | c
A → ba | SB
B → bA | S

string: *acbbac*

LL(1) Parsing Table

	a	b	c	$
S’	S’ → S$		S’ → S$	
S	S → aAS		S → c	
A	A → SB	A → ba	A → SB	
B	B → S	B → bA	B → S	


a
S’
a
S
$
a
a
A
S
$
c
A
S
$
c
S
B
S
$
c
c
B
S
$
b
B
S
$
b
b
A
S
$
stack
empty
$
$
c
c
$
c
S
$
a
a
S
$
b
b
a
S
$
b
A
S
$