Syntax Analysis
(Part 2)

CSE 415: Compiler Construction

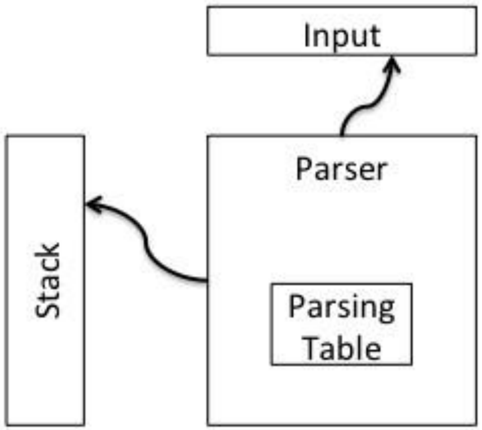
Parsing

- 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

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 $M[X,a] == X \rightarrow Y_1 Y_2 \ldots Y_k$ then {
      pop X from Stack;
      push $Y_k, Y_{k-1}, \ldots, Y_1$ onto Stack;
        (Y_1 on top)
    }
} until Stack has emptied;

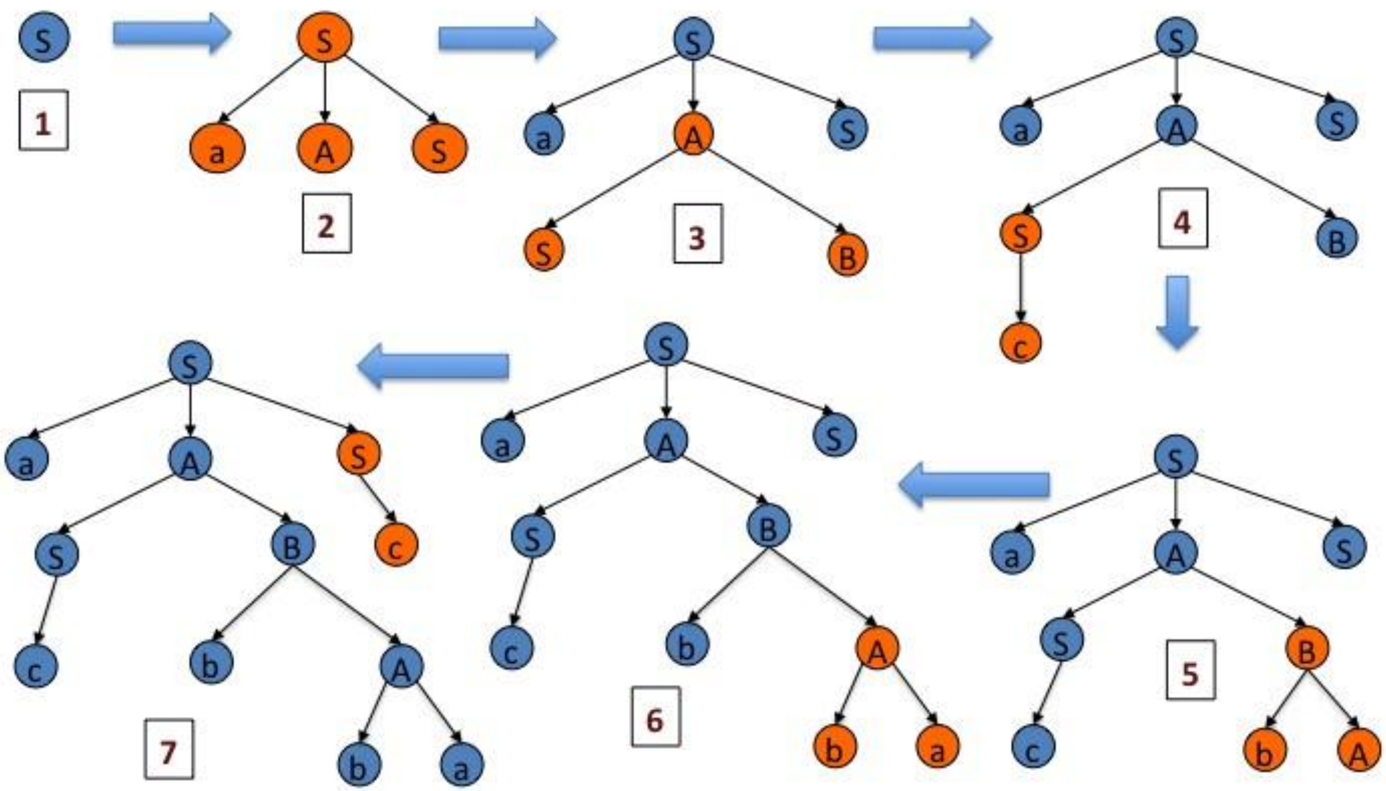
Top-down Parsing

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 Example

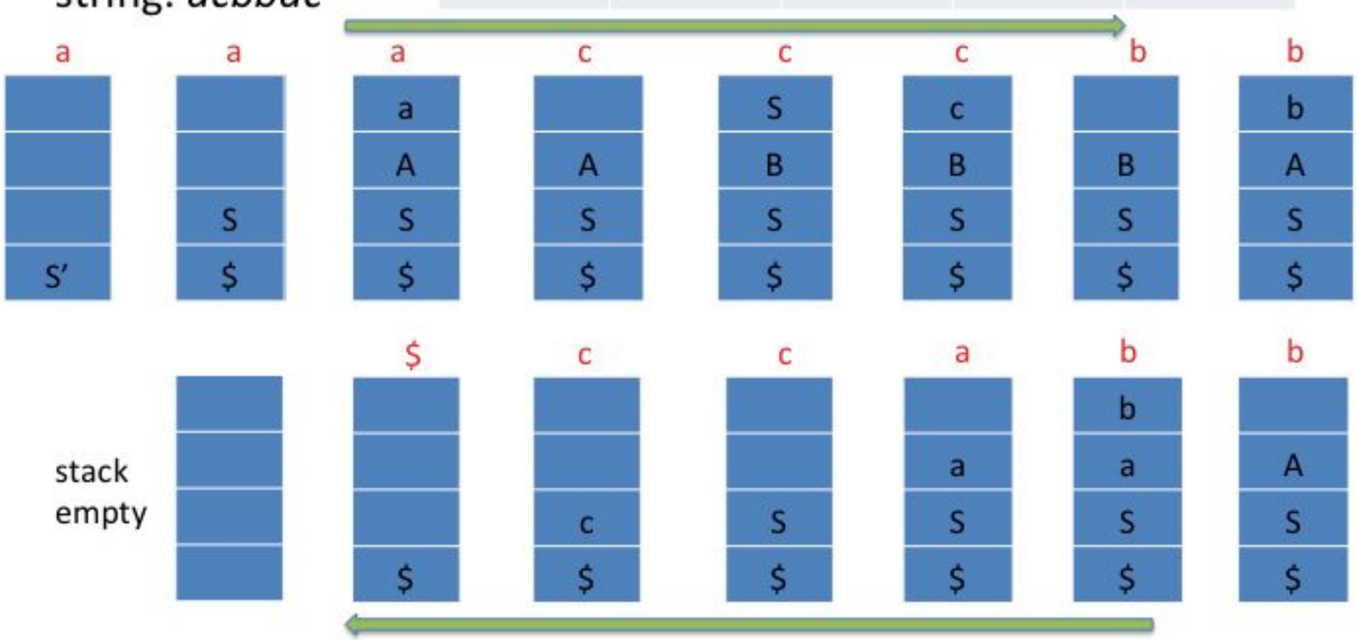
Grammar

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	



Testable conditions for LL(1) Grammars

- We call strong LL(1) as LL(1) from now on and we will not consider lookaheads longer than 1
- The classical condition for LL(1) property uses *FIRST* and *FOLLOW* sets
- If α is any string of grammar symbols ($\alpha \in (N \cup T)^*$), then
  $FIRST(\alpha) = \{a \mid a \in T, \ and\ \alpha \Rightarrow^* ax, \ x \in T^*\}$
  $FIRST(\epsilon) = \{\epsilon\}$
- If A is any nonterminal, then
  $FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, \ \alpha, \beta \in (N \cup T)^*, \ a \in T \cup \{\$\}\}$
- $FIRST(\alpha)$ is determined by α alone, but $FOLLOW(A)$ is determined by the "context" of A, i.e., the derivations in which A occurs

FIRST and FOLLOW Computation Example

- Consider the following grammar

  $S' \rightarrow S\$,\ \ S \rightarrow aAS \mid c,\ \ A \rightarrow ba \mid SB,\ \ B \rightarrow bA \mid S$
- $FIRST\,(S') = FIRST\,(S) = \{a, c\}$ because

  $S' \Rightarrow S\$ \Rightarrow \underline{c}\$$, and $S' \Rightarrow S\$ \Rightarrow \underline{a}AS\$ \Rightarrow \underline{a}baS\$ \Rightarrow \underline{a}bac\$$
- $FIRST\,(A) = \{a, b, c\}$ because

  $A \Rightarrow \underline{b}a$, and $A \Rightarrow SB$, and therefore all symbols in $FIRST\,(S)$ are in $FIRST\,(A)$
- $FOLLOW\,(S) = \{a, b, c, \$\}$ because

  $S' \Rightarrow \underline{S\$}$,

  $S' \Rightarrow^* aAS\$ \Rightarrow a\underline{S}BS\$ \Rightarrow aS\underline{b}AS\$$,

  $S' \Rightarrow^* a\underline{S}BS\$ \Rightarrow a\underline{S}SS\$ \Rightarrow aS\underline{a}ASS\$$,

  $S' \Rightarrow^* a\underline{S}SS\$ \Rightarrow aS\underline{c}S\$$
- $FOLLOW\,(A) = \{a, c\}$ because

  $S' \Rightarrow^* a\underline{A}S\$ \Rightarrow aA\underline{a}AS\$$,

  $S' \Rightarrow^* a\underline{A}S\$ \Rightarrow aA\underline{c}$

FIRST Computation: Algorithm Trace - 1

- Consider the following grammar
  $S' \rightarrow S\$$, $S \rightarrow aAS \mid \epsilon$, $A \rightarrow ba \mid SB$, $B \rightarrow cA \mid S$
- Initially, FIRST(S) = FIRST(A) = FIRST(B) = $\varnothing$
  - Iteration 1
  - FIRST(S) = $\{a, \epsilon\}$ from the productions $S \rightarrow aAS \mid \epsilon$
  - FIRST(A) = $\{b\} \cup$ FIRST(S) - $\{\epsilon\} \cup$ FIRST(B) - $\{\epsilon\} = \{b, a\}$

  from the productions $A \rightarrow ba \mid SB$
  (since $\epsilon \in$ FIRST(S), FIRST(B) is also included; since FIRST(B)=φ, ϵ is not included)
  FIRST(B) = $\{c\} \cup$ FIRST(S) - $\{\epsilon\} \cup \{\epsilon\} = \{c, a, \epsilon\}$
  from the productions $B \rightarrow cA \mid S$
  (ϵ is included because $\epsilon \in$ FIRST(S))

FIRST Computation: Algorithm Trace - 2

- The grammar is
  $S' \rightarrow S\$$, $S \rightarrow aAS \mid \epsilon$, $A \rightarrow ba \mid SB$, $B \rightarrow cA \mid S$
- From the first iteration,
  FIRST(S) = $\{a, \epsilon\}$, FIRST(A) = $\{b, a\}$, FIRST(B) = $\{c, a, \epsilon\}$
- Iteration 2
  (values stabilize and do not change in iteration 3)
  FIRST(S) = $\{a, \epsilon\}$ (no change from iteration 1)
  FIRST(A) = $\{b\} \cup$ FIRST(S) - $\{\epsilon\} \cup$ FIRST(B) - $\{\epsilon\} \cup \{\epsilon\}$
  = $\{b, a, c, \epsilon\}$ (changed!)
  FIRST(B) = $\{c, a, \epsilon\}$ (no change from iteration 1)

Follow Computation: Algorithm Trace - 1

- Consider the following grammar
  $S' \rightarrow S\$$, $S \rightarrow aAS \mid \epsilon$, $A \rightarrow ba \mid SB$, $B \rightarrow cA \mid S$
- Initially, follow (S) = {\$}; follow (A) = follow (B) = $\varnothing$
  first (S) = {a, ϵ}; first (A) = {a, b, c, ϵ}; first (B) = {a, c, ϵ};
- Iteration 1 /* In the following, x $\cup$ = y means x = x $\cup$ y */ $S \rightarrow aAS$:
    follow (S) $\cup$ = {\$}; rest = follow (S) = {\$} follow (A) $\cup$ = (first (S) $-$ {ϵ}) $\cup$
    rest = {a, \$}
  $A \rightarrow SB$: follow (B) $\cup$ = follow (A) = {a, \$}
  rest = follow (A) = {a,\$}
  follow (S) $\cup$ = (first (B) $-$ {ϵ}) $\cup$ rest = {a, c, \$}
  $B \rightarrow cA$: follow (A) $\cup$ = follow (B) = {a,\$}
  $B \rightarrow S$: follow (S) $\cup$ = follow (B) = {a, c,\$}
  At the end of iteration 1
  follow (S) = {a, c,\$}; follow (A) = follow (B) = {a, \$}

FIRST Computation: Algorithm Trace - 2

- first (S) = $\{a, \epsilon\}$; first (A) = $\{a, b, c, \epsilon\}$; first (B) = $\{a, c, \epsilon\}$;
  - At the end of iteration 1

  follow (S) = $\{a, c, \$\}$; follow (A) = follow (B) = $\{a, \$\}$
- Iteration 2

  $S \longrightarrow aAS$: follow (S) $\cup$ = $\{a, c, \$\}$; rest = follow (S) = $\{a, c, \$\}$

  follow (A) $\cup$ = (first (S) $-$ $\{\epsilon\}$) $\cup$ rest = $\{a, c, \$\}$ (changed!)

  $A \longrightarrow SB$: follow (B) $\cup$ = follow (A) = $\{a, c, \$\}$ (changed!)

  rest = follow (A) = $\{a, c, \$\}$

  follow (S) $\cup$ = (first (B) $-$ $\{\epsilon\}$) $\cup$ rest = $\{a, c, \$\}$ (no change)

  At the end of iteration 2

  follow (S) = follow (A) = follow (B) = $\{a, c, \$\}$;

  The follow sets do not change any further