Bottom-up Parsing (Objectives)

- Given a context-free grammar, the student will be able to determine if it is SLR by constructing its corresponding parse table.
- Given a parse table and an input string, the student will be able to apply shift-reduce parsing to the string using the parse table.

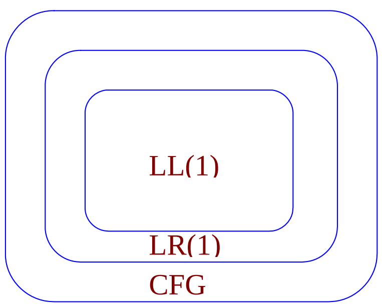
Context-Free Grammars

- A context-free grammar G is a quadruple, (N,T,P,S) where N is a set of nonterminals, T is a set of terminals, P is a set of productions and $S \in N$ is the start symbol.
- Example

$$
\begin{aligned}
E &\rightarrow T + E \\
&\mid T - E \\
&\mid T \\
T &\rightarrow F * T \\
&\mid F / T \\
&\mid F \\
F &\rightarrow \text{num} \\
&\mid \text{id}
\end{aligned}
$$

N = {E,T,F}
T = {+,-,*,/,num,id}
S = E

Types of Context-Free Grammars



LR Parsing

- LR(1) parsing (Left-to-right scan, Reverse of the rightmost derivation, 1 symbol of lookahead)
  - table-driven shift-reduce parsing
  - characteristic DFA plus a stack
  - SLR, Simple LR (ignore context for grammar productions)
  - Automatic parser generators (yacc, bison and lemon generate a subset of LR(1) known as LALR(1))

Parsing Terms

- For a grammar G with start symbol S, any string α, such that $S \Rightarrow^* \alpha$, is called a sentential form.
- If α contains only terminal symbols, then $\alpha \in L(G)$
- A left-sentential form occurs in a leftmost derivation.
- A right-sentential form occurs in a rightmost derivation

Bottom-Up Methodology

- Given a grammar G, construct a parse tree for a string from the leaves of the tree upward to the root.
  - Match a right sentential form against the tree's upper frontier.
  - Apply a reduction to advance the frontier.
  - Construct the derivation in reverse
  - Find a string α in the parse tree's upper frontier that matches some production $A \to \alpha$ and replace α with the lhs of the production.
  - α is a handle and this process is called handle pruning.

Shift-Reduce Parsers

- Bottom-up parser with 4 actions

1. *shift* - next input symbol is pushed onto the stack
2. *reduce* - pop a handle off of the stack and push the lhs of the corresponding production
3. *accept* - terminate parsing and signal success
4. *error* - call an error recovery routine

Shift-Reduce Algorithm

```
Stack.Push(S0)
repeat {
  s = Stack.Top();
  a = NextChar();
  if action[s,a] == 'shift sk' then
     Stack.Push(a); Stack.Push(sk);
  else
    if action[s,a] == 'reduce A → α'
    then {
           Stack.Pop(2*|α|);
           sk = Stack.Top();
           Stack.Push(A);
           Stack.Push(goto[sk,A]);
           UnPutChar();
           Emit(A → α);
         }
    else if action[s,a] = accept then {
       Emit("Success");
       return;
    else
       error()
}
```

Example Grammar

1. S' → E
2. E → T + E
3. | T
4. T → F * T
5. | F
6. F → id

Example Parse Table

	ACTION				GOTO		
	id	+	*	$	E	T	F
S_0	s4				1	2	3
S_1				!			
S_2		s5		r3			
S_3		r5	s6	r5			

	id	+	*	$	E	T	F
S_4		r6	r6	r6			
S_5	s4				7	2	3
S_6	s4					8	3
S_7				r2			
S_8		r4		r4			

Example

- Construct a parse for id + id * id

Common Types of Parsers

- SLR(1)
  - smallest class of grammars and tables
  - simple and fast
- LR(1)
  - large class of grammars
  - largest tables
  - slow, large constructions
- LALR(1)
  - intermediate class of grammars
  - same table size as SLR(1)
- LR(0), SLR(1) and LALR(1) use the same CFSM (Characteristic Finite State Machine)

Constructing SLR Parse Tables

- Add S' → S to P, S' is the start symbol with only one production
- LR(0) items which represent the possible states in a parse

  [A → α•β]

  - α is the sequence of symbols we have seen
  - β is the sequence of symbols we expect to see

Constructing LR(0) Items

```
C = {closure({S' → •S})}
repeat
    ∀ I∈C and ∀ x ∈ (N ∪ T)
      if goto(I,x) is a new set
         add goto(I,x) to C
until C does not change

goto(I,x)
    let J be the set of items
       [A → αx•β] such that
       [A → α•xβ] is in I
    return closure(J)

closure(I)
    repeat
     ∀ items [A → α•Bβ] in I
       ∀ production B →γ in G
          if [B → •γ] ∉I then
             add [B → •γ] to I
    until I does not change
    return I
```


Example Grammar

1. S' → E
2. E → T + E
3. | T
4. T → F * T
5. | F
6. F → id

Example

- Construct the LR(0) items for the example grammar.

SLR Table Construction

1. Construct the set of LR(0) items, C
2. Construct state s_i from $I_i \in$ C as follows
   1. If $A \rightarrow \alpha \cdot a\beta \in I_i$ and goto(I_i,a) == I_j then
      action[i,a] = “shift j”
   2. If $A \rightarrow \alpha \cdot \in I_i$ then
      $\forall a \in$ FOLLOW(A) | A ≠ S’, action[i,a] = “reduce $A \rightarrow \alpha$”
   3. If $S' \rightarrow S \cdot \in I_i$ then
      action[i,$] = “accept”

Example Parse Table

	ACTION				GOTO		
	id	+	*	$	E	T	F
S_0							
S_1							
S_2							
S_3							

	id	+	*	$	E	T	F
S_4							
S_5							
S_6							
S_7							
S_8							

FOLLOW Sets

- SLR parsing utilizes a lookahead string of length 1
  - Use FOLLOW sets – the set of terminals that may immediately follow a terminal or non-terminal in some derivation of a string
  - Used to determine when a reduction should be applied

- To construct FOLLOW sets:
  1. place $ in FOLLOW(S’)
  2. for $A \rightarrow \alpha B\beta$ add FIRST(β)-{ε} to FOLLOW(B)
  3. for $A \rightarrow \alpha B$ add FOLLOW(A) to FOLLOW(B)
  4. for $A \rightarrow \alpha B\beta$ if $\varepsilon \in$ FIRST(β) add FOLLOW(A) to FOLLOW(B)

Computing First Sets

- FIRST sets

  For some rhs of a production α, FIRST(α) is the set of tokens that appear as the first symbol in some string derived from α

  $$x \in \text{FIRST}(\alpha) \Leftrightarrow \exists x \in \Sigma \mid \alpha \Rightarrow^* x\gamma$$

  To compute FIRST(X)

  if X is a terminal then
  FIRST(X) = {X}
  else if $X \rightarrow \varepsilon$ then
  FIRST(X) $\cup$= {ε}
  else if $X \rightarrow Y_1 Y_2 \ldots Y_k$ then
  FIRST(X) $\cup$= FIRST(Y_1)
  $\forall i \mid \varepsilon \in \text{FIRST}(Y_j),\ 1 \le j < i,\ \text{FIRST}(X) \cup= \text{FIRST}(Y_i)$

Construct FIRST & FOLLOW Sets

Symbol	FIRST	FOLLOW
S'		
E		
T		
F		

Construct the SLR parse table for our example expression grammar

	ACTION				GOTO		
	id	+	*	$	E	T	F
S_0							
S_1							
S_2							
S_3							

	id	+	*	$	E	T	F
S_4							
S_5							
S_6							
S_7							
S_8							

Errors in Constructing the Parse Table

- shift/reduce conflict - the input grammar is ambiguous
  - rewrite the grammar to be unambiguous or use precedence

  S → if E then S
  | if E then S else S

  consider parsing if x then
  if y then print x;
  else print y;
- reduce/reduce conflict - grammar has two identical rhs with different lhs that appear in same context

Example

- Is the following grammar SLR(1)?

  S → Aa | bAc | Bc | bBa
  A → d
  B → d

Practice Example

Is the following grammar SLR?

$S \rightarrow Aa \mid bAc \mid dc \mid bda$
$A \rightarrow d$