03/24/2010

Recitation #9 – Worksheet

SLR Parser Construction

Given the grammar $G = (\{SheepNoise\}, \{\text{baa}\}, SheepNoise, P)$, where P is:

$$SheepNoise \rightarrow SheepNoise \text{ baa}$$
$$SheepNoise \rightarrow \text{baa}$$

For brevity you can use the short-hand notation:

$$SN \rightarrow SN \text{ baa}$$
$$SN \rightarrow \text{baa}$$

1. Write down the **Augmented Grammar** G'

   $G' = (\{S, SN\}, \{\text{baa}\}, S, P)$, where P

   $$S \rightarrow SN$$
   $$SN \rightarrow SN \text{ baa}$$
   $$SN \rightarrow \text{baa}$$

2. What are the **LR(0) items** for production $SN \rightarrow SN \text{ baa}$?

   $\{SN \rightarrow \bullet SN \text{ baa}, SN \rightarrow SN \bullet \text{baa}, SN \rightarrow SN \text{ baa} \bullet\}$

3. Calculate **CLOSURE**(I) for $I = \{S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}\}$. What are the **Kernel Items**?
   - CLOSURE({ $S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}$ }) = { $S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}$ }
   - Because $S \rightarrow \bullet SN$ is in CLOSURE, add $SN \rightarrow \bullet SN \text{ baa}$ and $SN \rightarrow \bullet \text{baa}$
   - Nothing to be done for $SN \rightarrow SN \bullet \text{baa}$ and $SN \rightarrow \bullet \text{baa}$, because terminal follows after dot.
   - Nothing to be done for $SN \rightarrow \bullet SN \text{ baa}$, because rules deriving from SN were already added.
   - CLOSURE({ $S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}$ }) =

     $S \rightarrow \bullet SN$, $SN \rightarrow SN \bullet \text{baa}$ — *Kernel Items*

     $SN \rightarrow \bullet SN \text{ baa}$, $SN \rightarrow \bullet \text{baa}$ — *Nonkernel Items*

4. Calculate **GOTO**(I , X) where $I = \{S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}\}$ and $X = \text{baa}$.
   - GOTO({ $S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}$ }, baa)
   - "baa" is immediately to the right of the dot for $SN \rightarrow SN \bullet \text{baa}$
   - Calculate CLOSURE({ $SN \rightarrow SN\ \text{baa}\bullet$ } = GOTO({ $S \rightarrow \bullet SN, SN \rightarrow SN \bullet \text{baa}$ }, baa) = { $SN \rightarrow SN\ \text{baa}\bullet$ }

5. Construct the **Canonical LR(0) Collection** using the following algorithm:

```
void items(G') {
    C = CLOSURE({[S' → ·S]});
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO(I, X) is not empty and not in C )
                    add GOTO(I, X) to C;
    until no new sets of items are added to C on a round;
}
```

   - C = CLOSURE({ $S \rightarrow \bullet SN$ })
     → Add I_0 = { $S \rightarrow \bullet SN,\ SN \rightarrow \bullet SN\ \text{baa},\ SN \rightarrow \bullet \text{baa}$ }} to C
   - *Round 1*: For set I_0
     - Grammar symbol S :
       GOTO(I_0 , S) = $\varnothing$, because S is not to the right of any dot
     - Grammar symbol SN :
       GOTO(I_0 , SN) =
       CLOSURE({ $S \rightarrow SN\bullet,\ SN \rightarrow SN\bullet\ \text{baa}$ }) = { $S \rightarrow SN\bullet,\ SN \rightarrow SN\bullet\ \text{baa}$ }
       → Add I_1 = { $S \rightarrow SN\bullet,\ SN \rightarrow SN\bullet\ \text{baa}$ } to C
     - Grammar symbol "baa" :
       GOTO(I_0 , baa) =
       CLOSURE({ $SN \rightarrow \text{baa}\bullet$ }) = { $SN \rightarrow \text{baa}\bullet$ }
       → Add I_2 = { $SN \rightarrow \text{baa}\bullet$ } to C
   - *Round 2* : For set I_1
     - Grammar symbol S :
       GOTO(I_1 , S) = $\varnothing$, because S is not to the right of any dot
     - Grammar symbol SN :

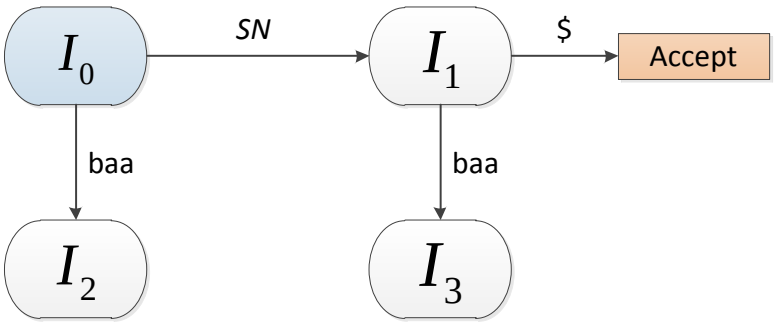
GOTO(I_1 , SN) = $\varnothing$, because SN is not to the right of any dot

- Grammar symbol "baa" :
  GOTO(I_1 , baa) =
  CLOSURE({ $SN \rightarrow SN$ baa• }) = { $SN \rightarrow SN$ baa• }
  → Add I_3 = { $SN \rightarrow SN$ baa• } to C

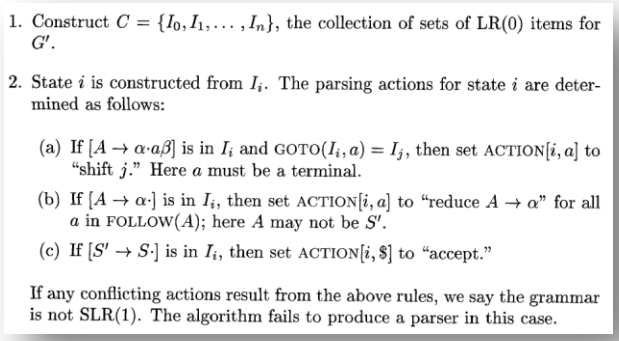
- *Round 2* : For set I_2
  - GOTO(I_2 , S) = $\varnothing$
  - GOTO(I_2 , SN) = $\varnothing$
  - GOTO(I_2 , baa) = $\varnothing$
- *Round 3* : For set I_3
  - GOTO(I_3 , S) = $\varnothing$
  - GOTO(I_3 , SN) = $\varnothing$
  - GOTO(I_3 , baa) = $\varnothing$

- C = { I_0, I_1, I_2, I_3 } = { { $S \rightarrow$•SN, $SN \rightarrow$•SN baa, $SN \rightarrow$•baa },
  { $S \rightarrow SN$•, $SN \rightarrow SN$• baa }, { $SN \rightarrow$ baa• }, { $SN \rightarrow SN$ baa• } }

6. Draw the resulting **LR(0) Automaton**.
   - Because $S \rightarrow SN$• is in I_1, add arc GOTO(I_1, $) = ACCEPT



7. Construct the **SLR-Parsing Table** using the following algorithm



1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.
2. State i is constructed from I_i. The parsing actions for state i are determined as follows:
   (a) If $[A \to \alpha \cdot a\beta]$ is in I_i and GOTO$(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.
   (b) If $[A \to \alpha \cdot]$ is in I_i, then set ACTION$[i, a]$ to "reduce $A \to \alpha$" for all a in FOLLOW(A); here A may not be S'.
   (c) If $[S' \to S \cdot]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.
3. The goto transitions for state i are constructed for all nonterminals A using the rule: If GOTO$(I_i, A) = I_j$, then GOTO$[i, A] = j$.
4. All entries not defined by rules (2) and (3) are made "error."
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

STATE	ACTION		GOTO	
	baa	$	S	SN
0	Shift 2			1
1	Shift 3	Accept		
2	Reduce $SN \to$ baa	Reduce $SN \to$ baa		
3	Reduce $SN \to SN$ baa	Reduce $SN \to SN$ baa		

- Calculate FOLLOW sets:
  - FOLLOW(S) = $\{\$\}$
  - FOLLOW(SN) = $\{\$, \text{baa}\}$
- GOTO columns can be taken from Canonical LR(0) collection or automaton
- Calculation of ACTION cells:
  - For state 0:
    - Rule (a) applies because $SN \to \bullet$baa is part of I_0 → GOTO(I_0, baa) = I_2 → ACTION(0, baa) = "Shift 2"
  - For state 1:
    - Rule (a) applies because $SN \to SN \bullet$ baa is part of I_1 → GOTO(I_1, baa) = I_3 → ACTION(1, baa) = "Shift 3"
    - Rule (c) applies because $S \to SN \bullet$ is part of I_1 → ACTION(1,$) = ACCEPT
  - For state 2:
    - Rule (b) applies because $SN \to$ baa$\bullet$ is part of I_2 → FOLLOW(SN) = $\{\$, \text{baa}\}$ → ACTION(2,$) = ACTION(2,baa) = "Reduce $SN \to$ baa"
  - For state 3:
    - Rule (b) applies because $SN \to SN$ baa$\bullet$ is part of I_3 → FOLLOW(SN) = $\{\$, \text{baa}\}$ → ACTION(3,$) = ACTION(3,baa) = "Reduce $SN \to SN$ baa"

8. Is this grammar SLR?
   Yes, because there are no shift/reduce or reduce/reduce conflicts in the parsing table.

9. Show me the actions of the resulting parser for the input string "baa baa baa baa" (bbbb)

	Stack	Input	Action	Output
1	\$0	bbbb\$	Shift 2	
2	\$0 b 2	bbb\$	Reduce $SN \rightarrow$ baa	$SN \rightarrow$ baa
3	\$0 *SN* 1	bbb\$	Shift 3	
4	\$0 *SN* 1 b 3	bb\$	Reduce $SN \rightarrow SN$ baa	$SN \rightarrow SN$ baa
5	\$0 *SN* 1	bb\$	Shift 3	
6	\$0 *SN* 1 b 3	b\$	Reduce $SN \rightarrow SN$ baa	$SN \rightarrow SN$ baa
7	\$0 *SN* 1	b\$	Shift 3	
8	\$0 *SN* 1 b 3	\$	Reduce $SN \rightarrow SN$ baa	$SN \rightarrow SN$ baa
9	\$0 *SN* 1	\$	Accept	

- At line (1) the SLR parser is in state 0, the initial state with no grammar symbols, and with "baa" the first input symbol
- ACTION(0,baa) = "Shift 2" meaning shift by pushing "baa" followed by state 2 onto the stack
- "baa" becomes the new input symbol in line (2).
- ACTION(2,baa) = "Reduce $SN \rightarrow$ baa". The state 2 and "baa" are then popped off the stack.
- SN is pushed onto the stack.
- Since state 0 is exposed and GOTO(0, SN) =1, push state 1 onto the stack
- …