A grammar that is not SLR but is LR(1)

- The following example is the same as the non-SLR grammar example in the textbook and the previously posted bottomup.pdf file, except for the “;” symbol added to the right-hand side of rules for S.

  1. S → L = R ;
  2. S → R ;
  3. L → ID
  4. L → *R
  5. R → L

  NOTE: Here L implies the “left value” and R implies the “right value”, as used in programming language definitions.

- To show the given grammar to be not SLR, we only need to examine the following two LR(0) states.
- S1:
  - S → . L = R
  - S → . R
  - L → . ID
  - L → . * R
  - R → . L
- S2:
  - S → L . = R
  - R → L .

S1 goes to S2 on L.

From S→ L=R, we know Follow(L) includes "=".

From L → * R, we know that Follow(L) is a subset of Follow(R) .

Because the FOLLOW set of R contains "=", we have a shift/reduce conflict in S2 due to input "="

We now compute LR(1) states

- S1:
  - S → . L = R ; {$}
  - S → . R ; {$}
  - L → . ID {=}
  - L → . * R {=}
  - R → . L {;}
  - L → . ID {;}
  - L → . * R {;}
- S2:
  - S → L . = R ; {$}
  - R → L . {;}

We use colors to highlight how the look-ahead symbols are determined, e.g. "=" is the look-ahead for the first appearance of L → . ID, because L is from S → . L = R ;

In S2, we copy the look-ahead {$} for S → L . = R ; from S → . L = R in S1, and we copy the look-ahead {;} for R → L . from R → . L in S1.

We perform reduce R → L . only if the next token is ";"

We no longer have the shift/reduce conflict.