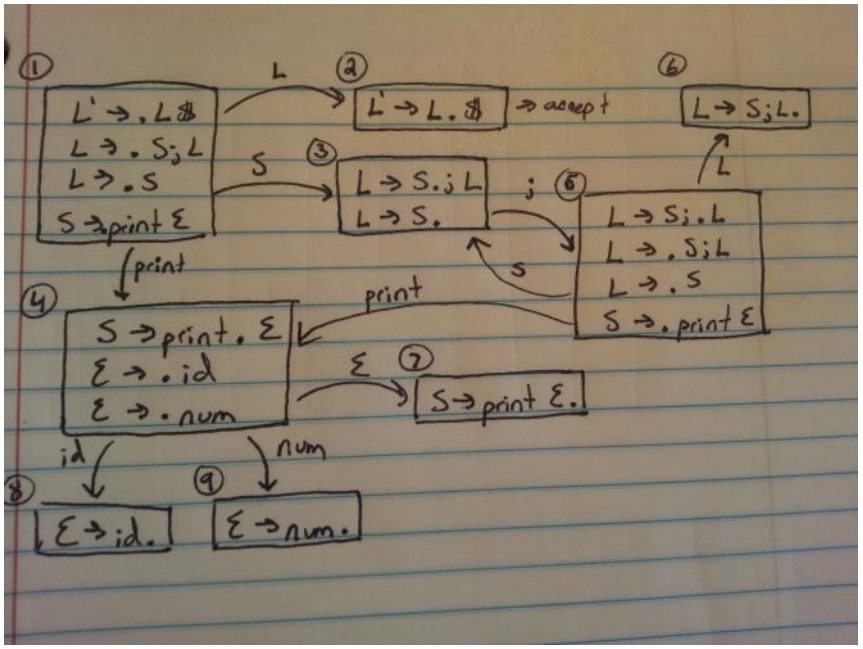
Shannon Gray
CMSC 305, HW 1
Due: Wednesday, October 2nd, 2013



State 1: Start of with Rule 0, the starting point. Since that has a Non-terminable after the (.) you have to add off of the rules that start with L ->, therefore getting the next two L rules. Since one of the L rules has a (.S), you have to add the S rule. Since the S rule has a terminable after the (.) we can now stop since we have reached closure.

State 2: An L-shift would bring you to this state, and then you move the (.) over, getting L.$, which leads to an accept state.

State 3: An S-Shift move to this state. You move the (.) like before, and you get one shift state and one reduce state, which causes a shift-reduce conflict.

State 4: Shifts on print. Moving the (.) after print puts it on E, which means you have to add the E rules to achieve closure again.

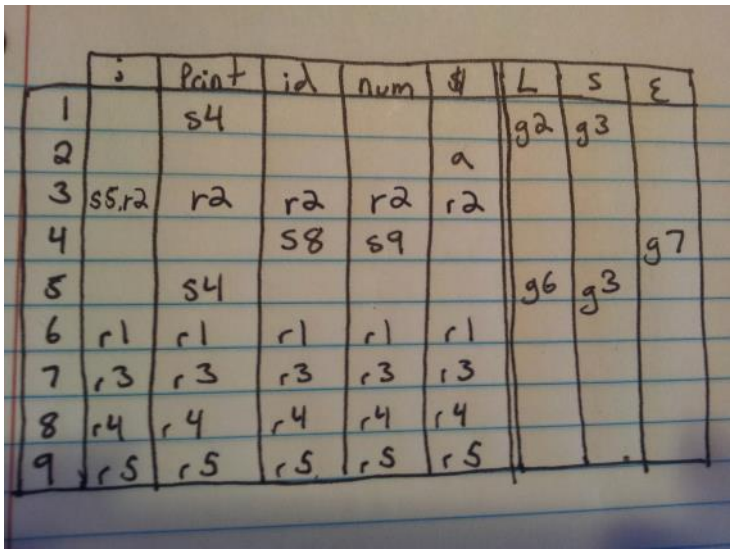
State 5: Shift on (;) so you put that rule, with the (.) moved. The new placement of the (.) now requires the L rules to be added, which then require the S rule as well, for closure.

State 6: Shift on L, which then makes this a reduce state.

State 7: E-Shift, also creates a reduce state.

State 8: id-Shift, creates a reduce state.

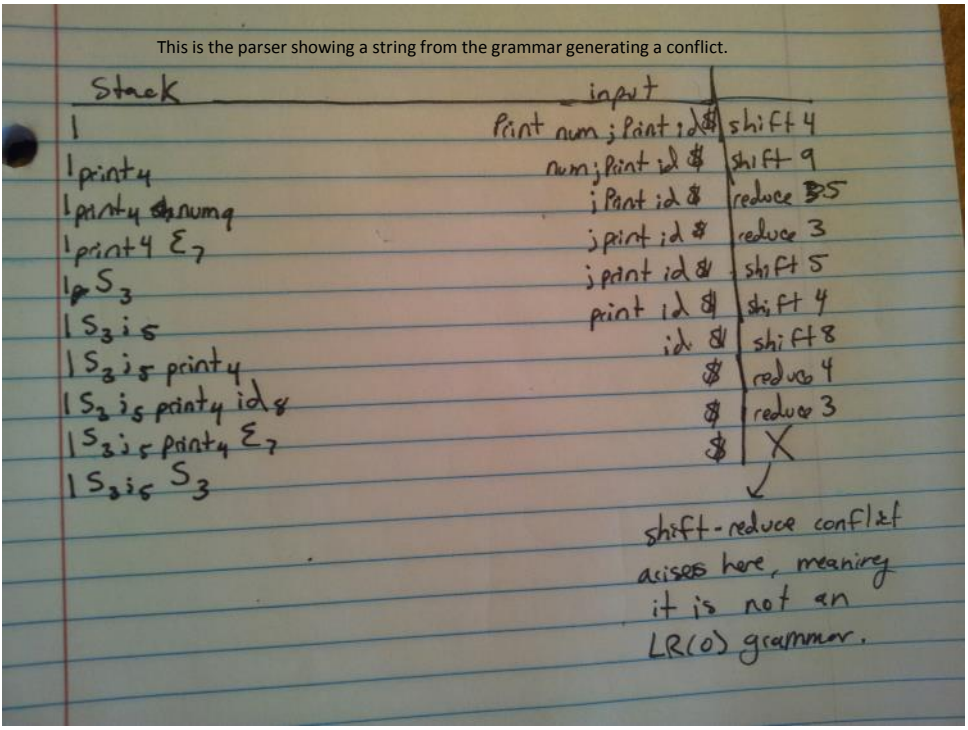
State 9: num-Shift, creates a reduce state.



	;	Print	id	num	$	L	S	E
1		s4				g2	g3	
2					a			
3	s5,r2	r2	r2	r2	r2			
4			s8	s9				g7
5		s4				g6	g3	
6	r1	r1	r1	r1	r1			
7	r3	r3	r3	r3	r3			
8	r4	r4	r4	r4	r4			
9	r5	r5	r5	r5	r5			

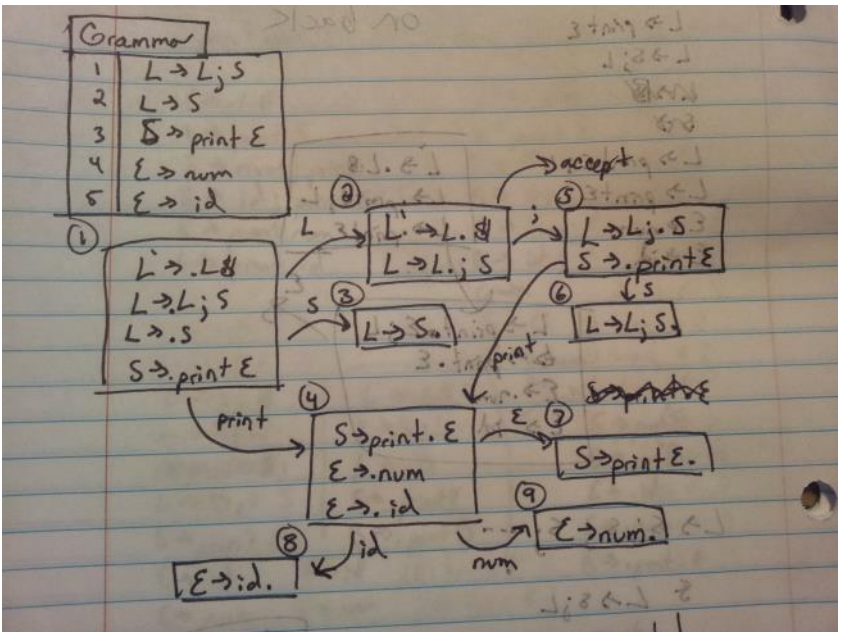
This is not an LR(0) grammar because it has a shift-reduce conflict in State 3. In order to be an LR(0) grammar, the DFA cannot have a shift and reduce in the same state.

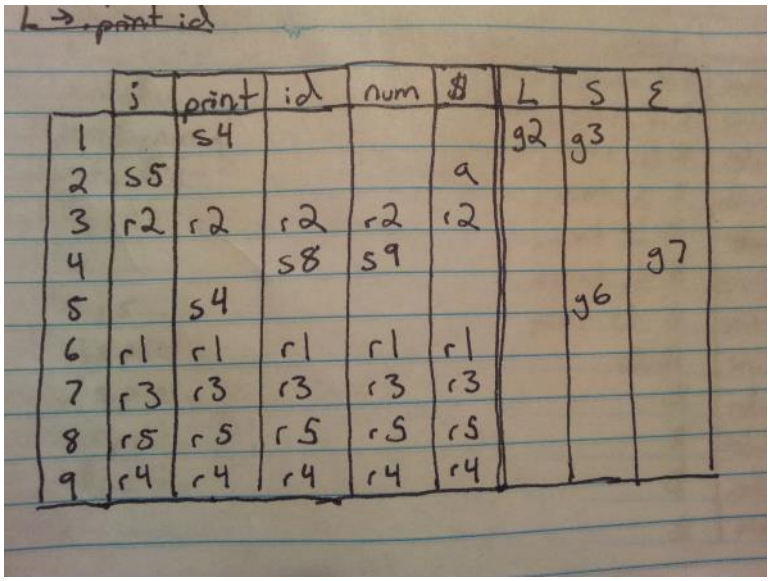
This is the parser showing a string from the grammar generating a conflict.

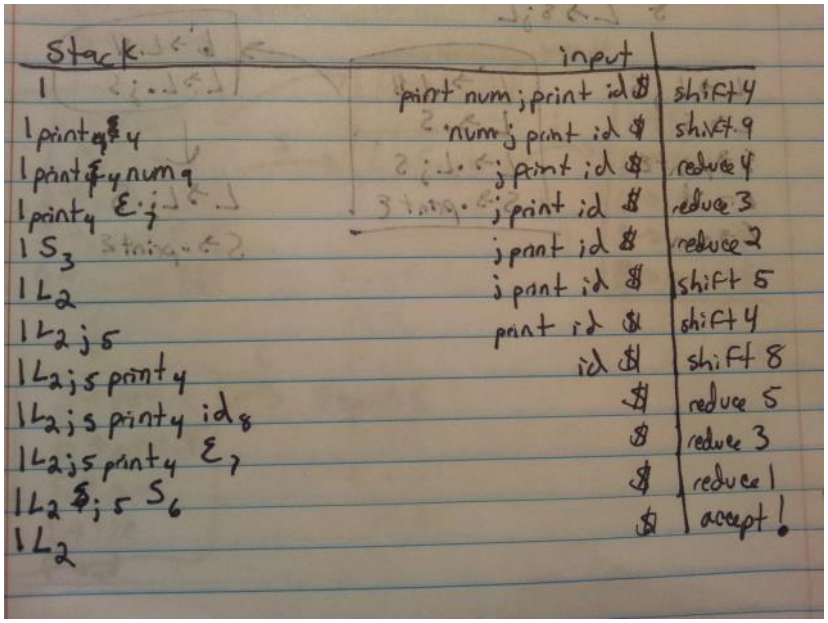


I think that introducing a rule that always shifts if able would correct the conflict. It would be like adding a "look ahead" component, which LR(0) grammars do not have, but LR(1) grammars do. It would allow the parser to know whether to shift or reduce in state 5, and correctly parse strings accepted by the grammar.

New LR(0) Grammar







Stack	input	
1	print num ; print id $	shift 4
1 print 4	num ; print id $	shift 9
1 print 4 num 9	; print id $	reduce 4
1 print 4 ε ;	; print id $	reduce 3
1 S_3	; print id $	reduce 2
1 L_2	; print id $	shift 5
1 L_2 ; 5	print id $	shift 4
1 L_2 ; 5 print 4	id $	shift 8
1 L_2 ; 5 print 4 id_8	$	reduce 5
1 L_2 ; 5 print 4 ε_7	$	reduce 3
1 L_2 S_6 ; 5 S_6	$	reduce 1
1 L_2	$	accept!