

The first example of constructing a bottom-up parser

- Given the following grammar that defines a program that consists a list of simple assignment statements:

1. $\langle \text{program} \rangle \rightarrow L$
2. $L \rightarrow \langle \text{stmt} \rangle$
3. $\quad \rightarrow \langle \text{stmt} \rangle L$
4. $\langle \text{stmt} \rangle \rightarrow \text{id} = \langle \text{expr} \rangle ;$
5. $\langle \text{expr} \rangle \rightarrow \text{int}$

NOTE: We will later expand the language by introducing more complex expressions and by allowing the left-hand side variable to be an array reference, etc.

The parsing table we constructed in class

States	Action					Goto		
	id	=	int	;	\$	L	<stmt>	<expr>
1	s2					g8	g7	
2		s3						
3			s4					g5
4	r5	r5	r5	r5	r5			
5				s6				
6	r4	r4	r4	r4	r4			
7	s2				r2	g9	g7	
8					acc			
9	r3	r3	r3	r3	r3			

NOTE: r5 means using rule 5 to reduce; acc means accept the input;
From this table, one can draw the state diagram that has circles and edges,
which we omit here.

Details in each state

- We constructed our parsing table based on “items” we found in each state and deduce all possible transitions.
- If a transition leads to a set of items (including the closure) that hasn’t appeared before, we create a new state.
- All transitions from the same state and labeled by the same symbol must have the same state as the target!
- State numbering is arbitrary. In the future, we examine each state, for which we haven’t examined the transitions yet, one by one, creating new states as needed. We keep a work list to store the new states. Removing a state from the work list at a time.
- For this example, we introduce new states in “depth-first” manner instead, in order to explain the state transitions in terms of an input example.
- The state tables built “depth-first” and “breadth-first” will be equivalent, differing only in the state numbering.

Our example

- S1: $\langle \text{program} \rangle \rightarrow . L$
 $L \rightarrow . \langle \text{stmt} \rangle$
 $L \rightarrow . \langle \text{stmt} \rangle L$
 $\langle \text{stmt} \rangle \rightarrow . \text{Id} = \langle \text{expr} \rangle ;$
- S2: $\langle \text{stmt} \rangle \rightarrow \text{id} . = \langle \text{expr} \rangle ;$
- S3: $\langle \text{stmt} \rangle \rightarrow \text{id} = . \langle \text{expr} \rangle ;$
 $\langle \text{expr} \rangle \rightarrow . \text{Int}$
- S4: $\langle \text{expr} \rangle \rightarrow \text{int} .$ (r5)
- S5: $\langle \text{stmt} \rangle \text{id} = \langle \text{expr} \rangle . ;$
- S6: $\text{id} = \langle \text{expr} \rangle ; .$ (r4)
- S7: $L \rightarrow \langle \text{stmt} \rangle .$ (r2 under \$, explained later)
 $L \rightarrow \langle \text{stmt} \rangle . L$
 $L \rightarrow . \langle \text{stmt} \rangle$
 $L \rightarrow . \langle \text{stmt} \rangle L$
 $\langle \text{stmt} \rangle \rightarrow . \text{Id} = \langle \text{expr} \rangle ;$
- S8: $\langle \text{program} \rangle \rightarrow L .$ (acc under \$)
- S9: $L \rightarrow \langle \text{stmt} \rangle L .$ (r3)

Applying the parsing table to an input example

- The parser maintains a parsing stack to store the history of shifts , reduces and gotos.
- Given input: "a = 3; b = 0;"

– Stack		-- next token position
– 1		-- ^ a = 1; b = 0;
– 1 2		-- a ^ = 1; b = 0;
– 1 2 3		-- a = ^ 1; b = 0;
– 1 2 3 4	(r5)	-- a = 1 ^ ; b = 0;
– 1 2 3	(g5)	
– 1 2 3 5		
– 1 2 3 5 6	(r4)	-- a = 1 ; ^ b = 0;
– 1	(g7)	
– 1 7		
– 1 7 2		-- a = 1 ; b ^ = 0;
– 1 7 2 3		-- a = 1; b = ^ 0;
– 1 7 2 3 4	(r5)	-- a = 1; b = 0 ^ ;
– 1 7 2 3	(g5)	
– 1 7 2 3 5		
– 1 7 2 3 5 6	(r4)	-- a = 1; b = 0 ; ^
– 1 7	(g7)	
– 1 7 7	(r2)	
– 1 7	(g9)	
– 1 7 9	(r3)	
– 1	(g8)	
– 1 8	(acc)	