SLR(1) Parsing

Recall

- Fairly straightforward to make LR(0) parser
- But languages that LR(0) parser can recognize are very limited
  - Usually, we have SR or RR conflicts
- Paradoxically, we can make the parser more powerful by restricting what it can do

SLR(1)

- Compute DFA as for LR(0)
- Create table
  - Shift-entries: Just as in LR(0)
  - Reduce entries: Will be conditioned on *follow* of left hand side
- Note: Consider follow[S'] to be {$}

Code

- Compute table: The only difference from LR(0) is in the reductions:

```
for s in states:
    row = {}
    for sym in s.transitions:
        if sym is a terminal:
            row[sym] = ("S",s.transitions[sym].index)
        else:
            row[sym] = ("T",s.transitions[sym].index)
    for item in s.items:
        lhs, rhs, dpos = item
        if dpos at end of rhs:
            for w in follow[lhs]:
                row[w] = ("R", len(rhs), lhs )
    table.append(row)
```

Example

- Grammar:

if $\rightarrow$ \bif\b
= $\rightarrow$ =
x $\rightarrow$ \d+
id $\rightarrow$ \w+
; $\rightarrow$;

($\rightarrow$ \(
) $\rightarrow$ \)

S $\rightarrow$ S stmt | λ
stmt $\rightarrow$ id (x) ; | id = stmt ; | x ;

Table

	State	$	(	)	;	=	S	id	stmt	x
0	S → • S → • S stmt S' → • S	R,0,S					T,1	R,0,S		R,0,S
1	stmt → • id (x) ; stmt → • id = stmt ; stmt → • x ; S → S • stmt S' → S •	R,1,S'						S,2	T,4	S,3
2	stmt → id • (x) ; stmt → id • = stmt ;		S,6			S,7				
3	stmt → x • ;				S,5					
4	S → S stmt •	R,2,S						R,2,S		R,2,S
5	stmt → x ; •	R,2,stmt			R,2,stmt			R,2,stmt		R,2,stmt
6	stmt → id (• x) ;									S,10
7	stmt → • id (x) ; stmt → • id = stmt ; stmt → • x ; stmt → id = • stmt ;							S,2	T,8	S,3
8	stmt → id = stmt • ;				S,9					
9	stmt → id = stmt ; •	R,4,stmt			R,4,stmt			R,4,stmt		R,4,stmt
10	stmt → id (x •) ;			S,11						
11	stmt → id (x) • ;				S,12					
12	stmt → id (x) ; •	R,5,stmt			R,5,stmt			R,5,stmt		R,5,stmt

Using Table

- How do we use the table?
- We maintain a stack
  - Stack contains automaton states
  - Top stack state $\rightarrow$ current state
  - Table actions tell us what to do with the stack and input

Shift

- Take token from input
- Push new FA state to stack

Reduce

- Let n = number of things in RHS of production we are reducing with
- Let L = lhs of production we're reducing with
- Pop n things from stack
- Consider the state that's on top of the stack.
- Transition out of that state using symbol L
- Push resulting state to stack

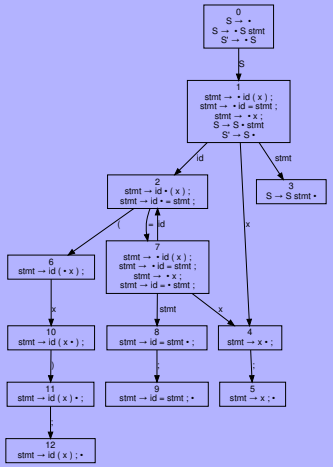
Example

if → \bif\b
= → =
x → \d+
id → \w+
; → ;

(→ \(
) → \)

S → S stmt | λ
stmt → id (x) ; | id = stmt ; | x ;

DFA


0
S → •
S → • S stmt
S' → • S
S
1
stmt → • id (x) ;
stmt → • id = stmt ;
stmt → • x ;
S → S • stmt
S' → S •
id
stmt
x
2
stmt → id • (x) ;
stmt → id • = stmt ;
3
S → S stmt •
(
= id
7
stmt → • id (x) ;
stmt → • id = stmt ;
stmt → • x ;
stmt → id = • stmt ;
6
stmt → id (• x) ;
x
stmt
x
10
stmt → id (x •) ;
8
stmt → id = stmt • ;
4
stmt → x • ;
)
;
;
11
stmt → id (x) • ;
9
stmt → id = stmt ; •
5
stmt → x ; •
;
12
stmt → id (x) ; •

Table

	State	$	(	)	;	=	S	id	stmt	x
0	S → • S → • S stmt S' → • S	R,0,S					T,1	R,0,S		R,0,S
1	stmt → • id (x) ; stmt → • id = stmt ; stmt → • x ; S → S • stmt S' → S •	R,1,S'						S,2	T,4	S,3
2	stmt → id • (x) ; stmt → id • = stmt ;		S,6			S,7				
3	stmt → x • ;				S,5					
4	S → S stmt •	R,2,S						R,2,S		R,2,S
5	stmt → x ; •	R,2,stmt			R,2,stmt			R,2,stmt		R,2,stmt
6	stmt → id (• x) ;									S,10
7	stmt → • id (x) ; stmt → • id = stmt ; stmt → • x ; stmt → id = • stmt ;							S,2	T,8	S,3
8	stmt → id = stmt • ;				S,9					
9	stmt → id = stmt ; •	R,4,stmt			R,4,stmt			R,4,stmt		R,4,stmt
10	stmt → id (x •) ;			S,11						
11	stmt → id (x) • ;				S,12					
12	stmt → id (x) ; •	R,5,stmt			R,5,stmt			R,5,stmt		R,5,stmt

Parse

- Parse:
  x = 42 ;

Code

```
stk=[0] #stack with initial state
ti = 0  #current token index
while 1:
    s = stk.top()  #index of a state
    t = tokens[ti].sym
    if table[s][t] does not exist:
        syntax error
    action = table[s][t]
    if action[0] == "S":
        #shift
        stk.push( action[1] )
        ti+=1       #consume token
    elif action[0] == "R":
        _,numpop,lhs = action
        if lhs == "S'":
            if ti == len(tokens): Accept
            else: Reject
        pop numpop things from stk
        si=stk.top()
        action = table[si][lhs]
        stk.push( action[1] )
```

Parsing Algorithm

- What we've just seen: A transducer
- But if we want parse tree, we need to do a little more work
- We have two stacks: One for state numbers, one for tree nodes
- Every time we shift: We create a tree node and push
  - Token field of node = shifted token
- Every time we reduce
  - Create new node P
  - Nodes popped become children of P

Code

```
stateStack=[]; nodeStack=[]; ti=0
while 1:
    s = stateStack.top()
    t = tokens[ti].sym
    if table[s][t] does not exist: error
    action = table[s][t]
    if action[0] == "S":
        stateStack.push( action[1] )
        nodeStack.push( TreeNode( t, tokens[ti] ) )
    else:
        numpop = action[1]; reduceTo = action[2]
        n = TreeNode( reduceTo, None )
        do numpop times:
            stateStack.pop();
            n.children.prepend(nodeStack.pop())
        if reduceTo == "S'":
            if t == "$": accept
            else: reject
        s = stateStack.top()
        stateStack.push( table[s][reduceTo][1] )
        nodeStack.push(n)
```

Example

- Example parse of: a = foo(42);
  - Do in class
- Why does this stop with an error?

Error

- Grammar requires two semicolons!

Example

- Parse of a = foo(42);;

Assignment

- Write a program which works with the test harness.

Sources

- K. Louden. *Compiler Construction: Principles and Practice*
- A. Aho, M. Lam, R. Sethi, J. Ullman. Compilers: Principles, Techniques, & Tools. (2nd ed.) Addison-Wesley.

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org