LR(1) Parsing Tables Example

CS 4447 / CS 9545 -- Stephen Watt
University of Western Ontario

Example Generating LR(1) Tables

Grammar

Terminals = { **$** , **;** , **id** , **:=** , **+** }

Nonterminals = {S', S, A, E }

Start Symbol = S'

Productions = {

1. S' → S **$**
2. S → S **;** A
3. S → A
4. A → E
5. A → **id** **:=** E
6. E → E **+** **id**
7. E → **id**

}

- Note that
  - *None* of the symbols are nullable.
  - FIRST(t) = t for all terminals.
  - FIRST(nt) = "id" for all non-terminals.

The Start State – Computing the Closure

- Find the closure of items with start symbol S’ as LHS and $ as look-ahead.

```
I0 := closure( { [S’ → . S, $] } )
```

- For `[S’ → . S, $]` we have `B = S, β = ε so`

```
S → A
S → S ; A
FIRST($) = $
```

  and we have the items

```
[S  → . A, $],
[S  → . S ; A, $].
```

- For `[S  → . A, $]` we have `B = A, β = ε so`

```
A → E
A → id := E
first($) = $
```

  and we have the items

```
[A  → . E, $],
[A  → . id := E, $].
```

The Start State (contd)

```
[S  → . S ; A, $] B = S, β = ; A
S → A
S → S ; A
        [S → . A, ;]
        [S → . S ; A, ;]

[A  → . E, $] B=E, β = ε
E → E + id
E → id
       [E → . E + id, $]
       [E → . id, $]

[S → . A, ;] B = A, β = ε
A → E
A → id := E
       [ A → . E, ; ]
       [ A → . id := E, ; ]

[S → . S ; A, ;], B=S, β =  ; A
S → A
S → S ; A
       Nothing new to add
```

```
[ E → . E + id, $] B=E, β=+ id
E → E + id
E → id
      [ E → . E + id, +]
      [ E → . id, +]

[A → . E, ; ], B=E, β=ε
E → E + id
E → id
      [ E → . E + id, ;]
      [ E → . id, ;]

[ E → . E + id, +] B=E, β=+ id
E → E + id
E → id
       Nothing new to add

[ E → . E + id, ;] B=E, β=+ id
E → E + id
E → id
       Nothing new to add
```

The Start State (cont'd)

So

```
I0 = Closure([S' → . S, $]) = {
  [S' → . S, $],
  [S → . A, $],
  [S → . S ; A, $],
  [A → . E, $],
  [A → . id := E, $],
  [S → . A, ;],
  [S → . S ; A, ;],
  [E → . E + id, $],
  [E → . id, $],
  [A → . E, ; ],
  [A → . id := E, ; ],
  [E → . E + id, +],
  [E → . id, +],
  [E → . E + id, ;],
  [E → . id, ;]
 }
```

The First Transitions

- Form `I1-I4` by moving the dot past the first symbol in each rule.
- This means moving the dot past `S, A, E, id`

```
I1 := Closure { [S'→ S ., $], [S → S . ; A, $], [S → S . ; A, ;] }
I2 := Closure { [S → A ., $], [S → A ., ;] }
I3 := Closure { [A → E ., $], [A → E ., ;],
                [E → E . + id, $], E → E . + id, ;], E → E . + id, +] }
I4 := Closure { [A → id . := E, $], [A → id . := E, ;],
                [E → id ., $], [E→ id ., ;], [E → id ., +] }
```

- Because in each case the dot either precedes a terminal or is at the end there cannot be any rules such that A → α . B β where B → γ is a production in the grammar.   We have:

```
I1 := { [S'→ S ., $],  [S → S . ; A, $], [S → S . ; A, ;] }
I2 := { [S → A ., $], [S → A ., ;] }
I3 := { [A → E ., $], [A → E ., ;],
        [E → E . + id, $], E → E . + id, ;], E → E . + id, +] }
I4 := { [A → id . := E, $], [A → id . := E, ;],
        [E → id ., $], [E→ id ., ;], [E → id ., +] }
```

The First Transitions (contd)

- We have the transitions

```
I0 → [S]  I1
I0 → [A]  I2
I0 → [E]  I3
I0 → [id] I4
```

Next Transitions

- We now need to determine the sets given by moving the dot past the symbols in the RHS of the productions in each of the new sets `I1-I4.`
- In `I1` the only symbol the dot can move past is ";".
  Likewise the only symbol dot can move past in `I3` is "+" and in `I4` is ":=".

```
GoTo(I1,;)  = closure {[S → S ; . A, $], [S → S ; . A, ;]}  = I5 =
   { [S→S ; . A, ;], [S→S ; . A, $],
     [A→. id := E, $], [A→. id := E, ;],
     [A→. E, $], [A→. E, ;],
     [E→. E + id, $], [E→. E + id, ;], [E→. E + id, +],
     [E→. id, $], [E→. id, ;], [E→. id, +] }

GoTo(I3,+)  = closure {[E→E+.id], $], [E→E+.id, ;], [E→E+.id, +] }) = I6 =
   { [E → E + . id, $], [E → E + . id, ;], [E → E + . id, +] }

GoTo(I4,:=) = closure {[A → id := . E,$], [A → id := . E,;]}) = I7 =
   { [A→id := . E, $], [A→id := . E, ;],
     [E→. E + id, $], [E→. E + id, ;], [E→. E + id, +],
     [E→. id, $], [E→. id, ;], [E→. id, +] }
```

Next Transitions (contd)

- We have the transitions

```
I1 →[;]  I5
I3 →[+]  I6
I4 →[:=] I7
```

More Transitions

- We must compute GoTo sets for `I5, I6` and `I7`.

```
GoTo(I5,A)   = closure { [S → S ;  A ., $], [S → S ;  A ., ;]}
             = { [S → S ;  A ., $], [S → S ;  A ., ;] } = I8

GoTo(I5,E)   = closure {[A→ E ., $], [A→ E ., ;],
                   [E→ E . + id, $], [E→ E . + id, ;], [E→ E . + id, +]} = I3

GoTo(I5,id)  = closure {[A→ id . := E, $], [A→ id . := E, ;],
                   [E→ id ., $], [E→ id ., ;], [E→ id ., +]}  = I4

GoTo(I6,id)  = closure {[E→ E+id., $], [E→ E+id., ;], [E→ E+id., +]}
             = {[E→ E + id ., $], [E→ E + id ., ;], [E→ E + id ., +]} = I9

GoTo(I7,E)   = closure {[E→ E.+id, $], [E→ E.+id, ;], [E→ E.+id, +],
                          [A→ id:=E., $], [A→ id:=E., ;]}
              = {[E→ E.+id, $], [E→ E.+id, ;], [E→ E.+id, +],
                  [A→ id:=E., $], [A→ id:=E., ;]} = I10

GoTo(I7,id)  = closure {[E→ id ., $], [E→ id ., ;], [E→ id ., +]}
             = {[E→ id ., $], [E→ id ., ;], [E→ id ., +]} = I11
```

More Transitions (contd)

- These are the transitions:

```
I5 →[A]  I8
I5 →[E]  I3
I5 →[id] I4
I6 →[id] I9
I7 →[E]  I10
I7 →[id] I11
```

- From this we see we need to compute new GoTo set:

```
GoTo(I10,+) = closure {[E→ E+.id, $], [E→ E+.id, ;], [E→ E+.id, +] }) = I6
```

- The last transition is therefore:

```
I10 →[+] I6
```

Parsing Automaton

- The parsing automaton has the following states:

```
I0 = { [S' → . S, $],
       [S → . A, $],       [S → . A, ;],          [S → . S ; A, $],   [S → . S ; A, ;],
       [A → . id := E, $], [A → . id := E, ; ],  [A → . E, $],       [A → . E, ; ],
       [E → . E + id, $],  [E → . E + id, +],     [E → . E + id, ;],
       [E → . id, $],      [E → . id, ;],         [E → . id, +] }

I1 = { [S'→ S ., $], [S → S . ; A, $], [S → S . ; A, ;] }
I2 = { [S → A ., $], [S → A ., ;] }
I3 = { [A → E ., $], [A → E ., ;], [E → E . + id, $], E → E . + id, ;], E → E . + id, +] }
I4 = { [A → id . := E, $], [A → id . := E, ;], [E → id ., $], [E→ id ., ;], [E → id ., +] }

I5 = { [S→S ; . A, ;], [S→S ; . A, $],
        [A→. id := E, $], [A→. id := E, ;], [A→. E, $], [A→. E, ;],
        [E→. E + id, $], [E→. E + id, ;], [E→. E + id, +], [E→. id, $], [E→. id, ;], [E→. id, +] }
I6 = { [E → E + . id, $], [E → E + . id, ;], [E → E + . id, +] }
I7 = { [A→id := . E, $], [A→id := . E, ;],
        [E→. E + id, $], [E→. E + id, ;], [E→. E + id, +],
        [E→. id, $], [E→. id, ;], [E→. id, +] }
I8 = { [S → S ;  A ., $], [S → S ;  A ., ;] }
I9 = { [E→ E + id ., $], [E→ E + id ., ;], [E→ E + id ., +]} )
I10= { [E→ E.+id, $], [E→ E.+id, ;], [E→ E.+id, +], [A→ id:=E., $], [A→ id:=E., ;]}
I11= { [E→ id ., $],  [E→ id ., ;],  [E→ id ., +]}
```

Parsing Automaton

- And the automaton has the following transitions:

```
I0 →[S] I1
I0 →[A] I2
I0 →[E] I3
I0 →[id] I4

I1 →[;] I5

I3 →[+] I6

I4 →[:=] I7

I5 →[A]  I8
I5 →[E]  I3
I5 →[id] I4

I6 →[id] I9

I7 →[E]  I10
I7 →[id] I11

I10 →[+] I6
```

Table Entries

- The states imply the following table entries

```
I0: none.

I1:
   [S'→ S ., $]   Action[I1,$] = accept
I2:
   [S → A ., $]   Action[I2,$] = reduce 3
   [S → A ., ;]   Action[I2,;] = reduce 3
I3:
   [A → E ., $]   Action[I3,$] = reduce 4
   [A → E ., ;]   Action[I3,;] = reduce 4
I4:
   [E → id ., $]  Action[I4,$] = reduce 7
   [E → id ., ;]  Action[I4,;] = reduce 7
   [E → id ., +]  Action[I4,+] = reduce 7

I5: none

I6: none

I7: none

I8:
   [S → S ;  A ., $]  Action[I8,$] = reduce 2
   [S → S ;  A ., ;]  Action[I8,;] = reduce 2

I9:
   [E→ E + id ., $]  Action[I9,$] = reduce 6
   [E→ E + id ., ;]  Action[I9,;] = reduce 6
   [E→ E + id ., +]  Action[I9,+] = reduce 6

I10:
   [A→ id:=E., $]    Action[I10,$] = reduce 5
   [A→ id:=E., ;]    Action[I10,;] = reduce 5

I11:
   [E→ id ., $]     Action[I11,$] = reduce 7
   [E→ id ., ;]     Action[I11,;] = reduce 7
   [E→ id ., +]     Action[I11,+] = reduce 7
```

Table Entries

- The transitions imply the following table entries:

```
I0 →[S] I1              GoTo[I0,S] = I1
I0 →[A] I2              GoTo[I0,A] = I2
I0 →[E] I3              GoTo[I0,E] = I3
I0 →[id] I4             Action[I0,id] = shift I4

I1 →[;] I5              Action[I1,;]  = shift I5

I3 →[+] I6              Action[I3,+]  = shift I6

I4 →[:=] I7             Action[I4,:=] = shift I7
I5 →[A]  I8             GoTo[I5,A] = I8
I5 →[E]  I3             GoTo[I5,E] = I3
I5 →[id] I4             Action[I5,id] =  shift I4

I6 →[id]  I9            Action[I6,id] =  shift I9

I7 →[E]  I10            GoTo[I7,E] = I10
I7 →[id] I11            Action[I7,id] = shift I11


I10→[+]  I6             Action[I10,+] = shift I6
```

Filling in the Tables

1. S' → S **$**
2. S → S **;** A
3. S → A
4. A → E
5. A → **id** **:=** E
6. E → E **+** **id**
7. E → **id**

W e see that adding one lookahead token removes all shift-reduce conflicts.

State	Action					GoTo			
	Id	**;**	**+**	**:=**	**$**	S'	S	A	E
0	S I4						I1	I2	I3
1		S I5			acc				
2		R 3			R 3				
3		R 4	S I6		R 4				
4		R 7	R 7	S I7	R 7				
5	S I4							I8	I3
6	S I9								
7	S I11								I10
8		R 2			R 2				
9		R 6	R 6		R 6				
10		R 5	S I6		R 5				
11		R 7	R 7		R 7				