Handout 5: LR(0), SLR Parser Example

Consider this grammar from Handout 4 (Predictive Parsers).

Grammar 2: Unambiguous grammar for +, *

Nonterminals: {S,E,T,F} (start, expressions, terms, factors)
Terminals: { id, num, *, +, (,), $ }
Productions:

(1) S → E $

(2) E → E + T
(3) E → T
(4) T → T * F
(5) T → F
(6) F → id
(7) F → num
(8) F → (E)

A rightmost derivation for the sentence:

x + 3 * (y + 4)

is shown below. Note how a sequence of terminal symbols is developed at the right end of the sequence.

```
S
E$
E  $
E+T  $
E+T*F  $
E+T*(E)  $
E+T*(E  )$
E+T*(E+T  )$
E+T*(E+F  )$
E+T*(E+4  )$
E+T*(E+  4)$
E+T*(E  +4)$
E+T*(T  +4)$
E+T*(F  +4)$
E+T*(y  +4)$
E+T*(  y+4)$
E+T*  (y+4)$
E+T  *(y+4)$
E+F  *(y+4)$
E+3  *(y+4)$
E+  3*(y+4)$
E  +3*(y+4)$
T  +3*(y+4)$
F  +3*(y+4)$
x  +3*(y+4)$
  x+3*(y+4)$
```

We turn this upside down to get a "parse" sequence. On the right is the terminal sentence being parsed. On the left is a "stack" of terminals and nonterminals.

```
  x+3*(y+4)$
x  +3*(y+4)$
F  +3*(y+4)$
T  +3*(y+4)$
E  +3*(y+4)$
E+  3*(y+4)$
E+3  *(y+4)$
E+F  *(y+4)$
E+T  *(y+4)$
E+T*  (y+4)$
E+T*(  y+4)$
E+T*(y  +4)$
E+T*(F  +4)$
E+T*(T  +4)$
E+T*(E  +4)$
E+T*(E+  4)$
E+T*(E+4  )$
E+T*(E+F  )$
E+T*(E+T  )$
E+T*(E  )$
E+T*(E)  $
E+T*F  $
E+T  $
E  $
E$
S
```

We can label this parse sequence with the actions performed at each step. How do we decide which transitions to perform?

```
    x+3*(y+4)$   - empty stack, full input
x    +3*(y+4)$   - shift
F    +3*(y+4)$   - reduce (Rule) 6
T    +3*(y+4)$   - reduce 5
E    +3*(y+4)$   - reduce 3
E+    3*(y+4)$   - shift
E+3    *(y+4)$   - shift
E+F    *(y+4)$   - reduce 7
E+T    *(y+4)$   - reduce 5
E+T*    (y+4)$   - shift  (could have reduced E+T by 2, why not?)
E+T*(    y+4)$   - shift
E+T*(y    +4)$   - shift
E+T*(F    +4)$   - reduce 6
E+T*(T    +4)$   - reduce 5
E+T*(E    +4)$   - reduce 3
E+T*(E+    4)$   - shift
E+T*(E+4    )$   - reduce 7
E+T*(E+F    )$   - reduce 5
E+T*(E+T    )$   - reduce 2
E+T*(E      )$   - shift
```

```
E+T*(E)      $   - reduce 8
E+T*F        $   - reduce 4
E+T          $   - reduce 2
E            $   - shift
E$               - reduce 1
S                - final accepting state, all input consumed
```

Parsing with states and transitions

Now we will elaborate the parsing process, associating with each stack position a "state" that summarizes the possible situation in terms of a set of "marked" productions, where a period is placed at the point in the production rhs representing our current position in the parse. The transition relation on these states guides the parsing actions.

Here n* indicates first appearance of a new state, and trans(s,x) indicates a transition from state s to a new state through the digestion of a given symbol x, which may be either a terminal or nonterminal.

1. Initial (state 1*)

```
    S -> .E$
    E -> .E+T
    E -> .T
    T -> .T*F
    T -> .F
    F -> .id
    F -> .num
    F -> .(E)

[1]    x+3*(y+4)$
```

2. shifting x(id): (state 2* from 1 by id)

```
    F -> id.

[1]x[2]     +3*(y+4)$
```

3. reducing Rule 6: (state 3* from 1 by F)

```
    T -> F.

[1]F[3]     +3*(y+4)$
```

4. reducing Rule 5: (state 4* from 1 by T)

```
    E -> T.
    T -> T.*F
```

```
[1]T[4]     +3*(y+4)$


5. reducing Rule 3: (state 5* from 1 by E)

   S -> E.$
   E -> E.+T

[1]E[5]     +3*(y+4)$


6. shifting +: (state 6* from 5 by +)

   E -> E+.T
   T -> .T*F
   T -> .F
   F -> .id
   F -> .num
   F -> .(E)

[1]E[5]+[6]     3*(y+4)$


7. shifting 3(num): state 7* = trans(6,num)

   F -> num.

[1]E[5]+[6]3[7]    *(y+4)$


8. reducing 7: state 3 = trans(6,F)

   T -> F.

[1]E[5]+[6]F[3]    *(y+4)$


9. reducing 5: state 8* = trans(6,T) [shift/reduce!]

   E -> E+T.
   T -> T.*F

[1]E[5]+[6]T[8]    *(y+4)$


10. shifting *: state 9* = trans(8,*)

   T -> T*.F
   F -> .id
   F -> .num
   F -> .(E)

[1]E[5]+[6]T[8]*[9]    (y+4)$
```

```
11. shifting "(": state 10* = trans(9,"(")

   F -> (.E)
   E -> .E+T
   E -> .T
   T -> .T*F
   T -> .F
   F -> .id
   F -> .num
   F -> .(E)

[1]E[5]+[6]T[8]*[9]([10]    y+4)$


12. shifting y: state 2 = trans(11,id)

   F -> id.

[1]E[5]+[6]T[8]*[9]([10]y[2]    +4)$


13. reducing Rule 6: state 3 = trans(10,F)

    T -> F.

[1]E[5]+[6]T[8]*[9]([10]F[3]    +4)$


14. reducing Rule 5: state 4 = trans(11,T)

    E -> T.
    T -> T.*F

[1]E[5]+[6]T[8]*[9]([10]T[4]    +4)$


15. reducing Rule 3: state 11* = trans(11,E)

   F -> (E.)
   E -> E.+T

[1]E[5]+[6]T[8]*[9]([10]E[11]    +4)$


16. shifting +: state 6 = trans(11,+)

   E -> E+.T
   T -> .T*F
   T -> .F
   F -> .id
   F -> .num
   F -> .(E)
```

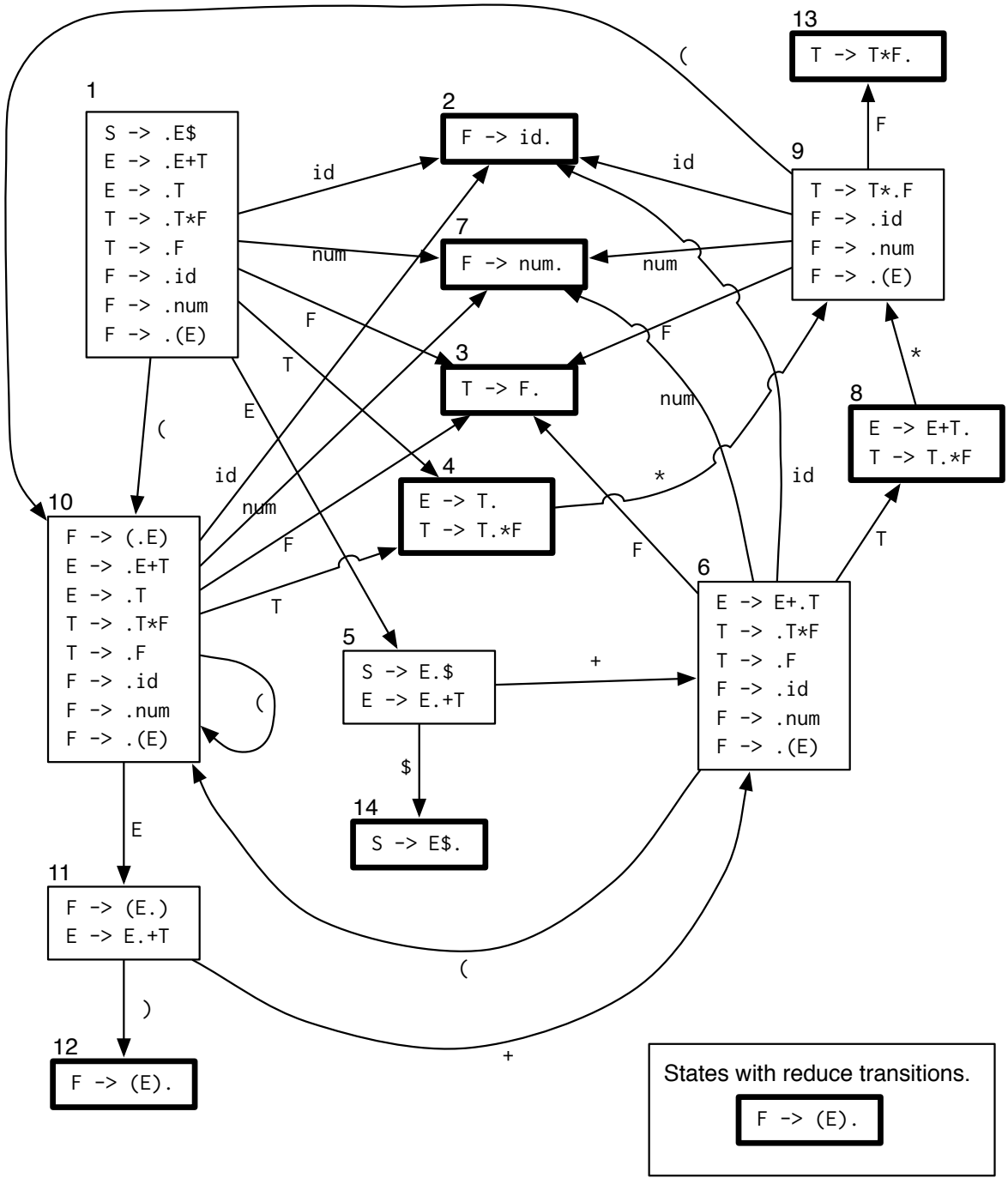
```
[1]E[5]+[6]T[8]*[9]([10]E[11]+[6]    4)$


17. shifting 4(num): state 7 = trans(6,num)

   F -> num.

[1]E[5]+[6]T[8]*[9]([10]E[11]+[6]4[7]    )$


18. reducing Rule 7: state 8 = trans(6,F)

   T -> F.

[1]E[5]+[6]T[8]*[0]([10]E[11]+[6]F[8]    )$


19. reducing Rule 5: state 8 = trans(6,T)

   E -> E+T.
   T -> T.*F

[1]E[5]+[6]T[8]*[9]([10]E[11]+[6]T[9]    )$


20. reducing Rule 2: state 12 = trans(11,E)

   F -> (E.)
   E -> E.+T

[1]E[5]+[6]T[8]*[9]([10]E[11]    )$


21. shifting ")": state 12* = trans(11,")")

   F -> (E).

[1]E[5]+[6]T[8]*[9]([10]E[11])[12]   $


22. reducing Rule 8: state 13* = trans(10,F)

   T -> T*F.

[1]E[5]+[6]T[8]*[9]F[13]   $


23. reducing Rule 4: state 8 = trans(6,T)

   E -> E+T.
   T -> T.*F

[1]E[5]+[6]T[8]   $
```

```
24. reducing Rule 2: state 5 = trans(1,E)

   S -> E.$
   E -> E.+T

[1]E[5]   $


25. shifting $:  state 14* = trans(5,$)

   S -> E$.

[1]E[5]$[14]


26. reducing Rule 1: state 15* = trans(1,S?)

   S.

[1]S

State 15 is accepting.
-----------------------------------------------------
```

So we have 15 states generated during the parse of this particular sentence (i.e. string of terminals).

What other states could be generated? Fortuitously, it turns out that these are all the possible states for an LR(0) grammar for this parser. The state machine is shown in the diagram below. States that allow a reduction (replacement of a rhs on the top of the stack by the lhs nonterminal) are shown with bold borders.

LR(0) Automaton for Grammer 2



Note that states 4 and 8 manifest shift reduce conflicts. For state 8, if the next input symbol is a *, we can either shift it and transition to state 9, or we can reduce by rule (2). But in this situation, if we choose the reduction, then our parse will fail, because * is not in FOLLOW(E), so we will not get to a state where we can

shift the *. A similar argument applies to state 4, which also exhibits a shift reduce conflict. So using the FOLLOW set for E, we can resolve the shift/reduce conflicts by refusing the reduction when the next terminal is *. This gives us an SLR parser.

Here is the transition table for this SLR parser:

	id	num	*	+	(	)	$	|	S	E	T	F
1	s2	s7			s10					g5	g4	g3
2	r6	r6	r6	r6	r6	r6	r6					
3	r5	r5	r5	r5	r5	r5	r5					
4	r3	r3	s9	r3	r3	r3	r3					
5				s6			s14					
6	s2	s7			s10						g8	g3
7	r7	r7	r7	r7	r7	r7	r7					
8	r2	r2	s9	r2	r2	r2	r2					
9	s2	s7			s10							g13
10	s2	s7			s10						g4	g3
11				s6		s12						
12	r8	r8	r8	r8	r8	r8	r8					
13	r4	r4	r4	r4	r4	r4	r4					
14	r1	r1	r1	r1	r1	r1	r1					
15	accept											