SLR Example (ASU Ch 4.7 pp221-230)

- **Constructing SLR Parsing Tables**
  - **LR(0) item is a grammar rule with a dot at some position of the RHS**
  - **LR(0) items from A ➔ X Y Z** *("item" for short)*

    A ➔ ¤ X Y Z *(¤ = dot for readability)*

    A ➔ X ¤ Y Z *(represents a viable prefix X)*

    A ➔ X Y ¤ Z *(represents a viable prefix XY)*

    A ➔ X Y Z ¤ *(represents the handle XYZ)*
  - **moving the dot represents how much of a production has been processed i.e. a viable prefix (until the handle has been found)**
  - **in the parse table for T = terminal & NT = Non-Terminal**
    - **¤ T ➔ T ¤** will appear as a shift to a new state in the action table
    - **¤ NT ➔ NT ¤** will appear as a state transition in the goto table

SLR Example (ASU Ch 4.7 pp221-230)

- The main steps in constructing an SLR Parse Table are
  - construct the LR(0) items *(using the dot notation)*
  - apply the closure operation (ASU pp222-224)
    - augment the grammar S’ → S
  - apply the goto operation (ASU p224)
  - derive the canonical LR(0) collection (ASU Fig 4.35)
- to understand how this works, look at the example in ASU
- grammar

E’ → E	*rule 0*
E → E + T ; E → T	*rules 1 & 2*
T → T * F ; T → F	*rules 3 & 4*
F → (E) ; F → id	*rules 5 & 6*

LR Parsers: example (ASU Ch 4.7, Ex 4.33, Fig 4.31)

E → (1) E+T | (2) T ; T → (3) T*F | (4) F ; F →(5) (E) | (6) id

VP/H	state	action id	+	*	(	)	$	goto E	T	F
	0	s5			s4			1	2	3
E	1		s6				acc			
T	2		r2	s7		r2	r2			
F	3		r4	r4		r4	r4			
(	4	s5			s4			8	2	3
id	5		r6	r6		r6	r6			
E+	6	s5			s4				9	3
T*	7	s5			s4					10
(E	8		s6			s11				
E+T	9		r1	s7		r1	r1			
T*F	10		r3	r3		r3	r3			
(E)	11		r5	r5		r5	r5			

Transitions (Ts =>shifts, NTs => gotos, I_x is item x)

I_0:	E' ➔ ¤ E	➔ ¤ T, ¤ F, ¤ (, ¤ id by the closure operation	
I_1:	goto(I_0, E)	{ [E' ➔ E ¤], [E ➔ E ¤ + T] }	
I_2:	goto(I_0, T)	{ [E ➔ T ¤], [T ➔ T ¤ * F] }	+ *goto(I_4, T)*
I_3:	goto(I_0, F)	{ [T ➔ F ¤] }	+ *goto(I_4, F) & goto(I_6, F)*
I_4:	goto(I_0, ( )	{ [F ➔ (¤ E)]	+ *all for ¤ E (see LR(0))* }
I_5:	goto(I_0, id)	{ [F ➔ id ¤] }	
I_6:	goto(I_1, +)	{ [E ➔ E + ¤ T]	+ *all for ¤ T (see LR(0))* }
I_7:	goto(I_2, *)	{ [T ➔ T * ¤ F]	+ *all for ¤ F (see LR(0))* }
I_8:	goto(I_4, E)	{ [F ➔ (E ¤)], [E ➔ E ¤ + T] }	
I_9:	goto(I_6, T)	{ [E ➔ E + T ¤], [T ➔ T ¤ * F] }	
I_{10}:	goto(I_7, F)	{ [T ➔ T * F ¤] }	
I_{11}:	goto(I_8,))	{ [F ➔ (E) ¤] }	

Transitions (Ts =>shifts, NTs => gotos)

State 0 (I_0 item 0)

I_0: E' → ¤ E { [E' → ¤ E], [E → ¤ E + T], [E → ¤ T],
[T → ¤ T * F], [T → ¤ F],
[F → ¤ (E)], [F → ¤ id] }

Transitions from state 0

I_1: goto(I_0, E)	{ [E' → E ¤], [E → E ¤ + T] }	→ goto 1
I_2: goto(I_0, T)	{ [E → T ¤], [T → T ¤ * F] }	→ goto 2
I_3: goto(I_0, F)	{ [T → F ¤] }	→ goto 3
I_4: goto(I_0, ( )	{ [F → (¤ E)] + *all for ¤ E (see LR(0))* }	→ shift 4
I_5: goto(I_0, id)	{ [F → id ¤] }	→ shift 5

Transitions (Ts =>shifts, NTs => gotos)

State 1 (I$_1$ item 1) – transition from state 0 on E

I$_1$: goto(I$_0$, E) **{ [E' ➔ E ¤], [E ➔ E ¤ + T]**

(i) E (start symbol) found ➔ accept (next T is $)

(ii) E found and next symbol may be + (rule 1)

Transitions from state 1

I$_6$: goto(I$_1$, +) **{ [E ➔ E ¤ + T] }**

➔ shift 6

Transitions (Ts =>shifts, NTs => gotos)

State 2 (I_2 item 2) – transition from state 0 on T

I_2: goto(I_0, T) { [E ➔ T ¤] } ➔ reduce 2

State 2 (I_2 item 2) – transition from state 4 on T

I_2: goto(I_4, T) { [E ➔ T ¤], [T ➔ T ¤ * F] } ➔ reduce 2

Transitions from state 2

I_7: goto(I_2, *) { [T ➔ T ¤ * F] } ➔ shift 7

Transitions (Ts =>shifts, NTs => gotos)

State 3 (I_3 item 3) – transition from state 0 on F

I_3: goto(I_0, F) { [T → F ¤] }

→ reduce 4

State 3 (I_3 item 3) – transition from state 4 on F

I_3: goto(I_4, F) { [T → F ¤] }

→ reduce 4

State 3 (I_3 item 3) – transition from state 6 on F

I_3: goto(I_6, F) { [T → F ¤] }

→ reduce 4

Transitions from state 3

none

Transitions (Ts =>shifts, NTs => gotos)

State 4 (I$_4$ item 4) – transition from state 0 on '('

I$_4$: goto(I$_0$, ( ) { [F → (¤ E)] + *all for ¤ E (see LR(0))* }

→ {[F → (¤ E)], [E → ¤ E + T], [E → ¤ T], [T → ¤ T * F], [T → ¤ F], [F → ¤ (E)], [F → ¤ id] }

State 4 (I$_4$ item 4) – transition from state 4 on '('

State 4 (I$_4$ item 4) – transition from state 6 on '('

State 4 (I$_4$ item 4) – transition from state 7 on '('

Transitions (Ts =>shifts, NTs => gotos)

Transitions from state 4

I_8: goto(I_4, E)	{ [F ➔ (E ¤)], [E ➔ E ¤ + T] }	➔ goto 8
I_2: goto(I_4, T)	{ [E ➔ T ¤], [T ➔ T ¤ * F] }	➔ goto 2
I_3: goto(I_4, F)	{ [T ➔ F ¤] }	➔ goto 3
I_4: goto(I_4, ( )	{ [F ➔ (¤ E)] + *all for ¤ E (see LR(0))* }	➔ shift 4
I_5: goto(I_4, id)	{ [F ➔ id ¤] }	➔ shift 5

State 5 (I_5 item 5) – transition from state 0 on 'id'

I_5: goto(I_0, id) { [F ➔ id ¤] } ➔ reduce 6

Transitions from state 5

none

Transitions (Ts =>shifts, NTs => gotos)

State 6 (I_6 item 6) – transition from state 1 on '+'

I_6: goto(I_1, +) { [E → E + ¤ T] + *all for ¤ T (see LR(0))* }

=> {[E → E + ¤ T], [T → ¤ T * F], [T → ¤ F],

[F → ¤ (E)], [F → ¤ id] }

Transitions from state 6

I_9: goto(I_6, T)	{ [<u>E → E + T ¤</u>], [T → T ¤ * F] }	→ goto 9
I_3: goto(I_6, F)	{ [<u>T → F ¤</u>] }	→ goto 3
I_4: goto(I_6, ( )	{ [F → (¤ E)] + *all for ¤ E (see LR(0))* }	→ shift 4
I_5: goto(I_6, id)	{ [<u>F → id ¤</u>] }	→ shift 5

Transitions (Ts =>shifts, NTs => gotos)

State 7 (I7 item 7) – transition from state 2 on '*'

I_7: goto(I_2, *) { [T → T * ¤ F] + *all for ¤ F (see LR(0))* }

=> {[T → T * ¤ F], [T → ¤ F], [F → ¤ (E)], [F → ¤ id] }

Transitions from state 7

I_{10}: goto(I_7, F)	{ [T → T * F ¤] }	→ goto10
I_4: goto(I_7, ( )	{ [F → (¤ E)] + *all for ¤ E (see LR(0))* }	→ shift 4
I_5: goto(I_7, id)	{ [F → id ¤] }	→ shift 5

Transitions (Ts =>shifts, NTs => gotos)

State 8 (I_8 item 8) – transition from state 4 on E

I_8: goto(I_4, E) { [F ➔ (E ¤)], [E ➔ E ¤ + T] }

Transitions from state 8

I_{11}: goto(I_8,)) { [F ➔ (E) ¤] } ➔ shift11

I_6: goto(I_8, +) { [E ➔ E + ¤ T] + *all for ¤ T (see LR(0))* } ➔ shift 6

State 9 (I_9 item 9) – transition from state 6 on T

I_9: goto(I_6, T) { [E ➔ E + T ¤], [T ➔ T ¤ * F] } ➔ reduce1

Transitions from state 9

I_7: goto(I_9, *) { [T ➔ T * ¤ F] + *all for ¤ F (see LR(0))* } ➔ shift 7

State 10 (I_{10} item 10) – transition from state 7 on F

I_{10}: goto(I_7, F) { [T ➔ T * F ¤] } ➔ reduce3

State 11 (I_{11} item 11) – transition from state 8 on ‘)’

I_{11}: goto(I_8,)) { [F ➔ (E) ¤] } ➔ reduce5

Canonical LR(0) Collection (ASU Fig 4.35)

I_0: E' → ¤ E
E → ¤ E+T
E → ¤ T
T → ¤ T*F
T → ¤ F
F → ¤ (E)
F → ¤ id

I_1: E' → E ¤
E → E ¤ +T

I_2: E → T ¤
T → T ¤ *F

I_3: E → F ¤

I_4: F → (¤ E)
E → ¤ E+T
E → ¤ T
T → ¤ T*F
T → ¤ F
F → ¤ (E)
F → ¤ id

I_5: F → id ¤

I_6: E → E+ ¤ T
T → ¤ T*F
T → ¤ F
F → ¤ (E)
F → ¤ id

I_7: T → T* ¤ F
F → ¤ (E)
F → ¤ id

I_8: F → (E ¤)
E → E ¤ +T

I_9: E → E + T ¤
T → T ¤ * F

I_{10}: T → T * F ¤

I_{11}: F → (E) ¤

SLR Algorithm (ASU pp227-228, Alg 4.8)

- Input: an augmented grammar G’
- Output: the SLR parsing table functions action and goto for G’
- Construct C = { I_0, I_1, …, I_n } - the collection of sets of LR(0) items

- State i is constructed from I_i - the parsing actions are determined
  - if [A ➔ α ¤ aβ] is in I_i and goto(I_i, a) = I_j then set action[i, a] to shift j

    a must be a terminal
  - if [A ➔ α ¤] is in I_i then set action[i, a] to reduce A => α

    for all a in follow(A) *(A may not be S’)*
  - if [S’ ➔ S ¤] is in I_i then set action[i, $] to “accept”
- follow(E) = { $, +,) }, follow(T) ={ $, +,), * }, follow(F) = { $, +,), * }

SLR Algorithm (ASU pp227-228, Alg 4.8)

- The goto transitions for state i are constructed for all non-terminals A using the rule:

  if goto(I_i, A) = I_j then goto[i, A] = j

- all other entries not defined by the above are made error
- the initial state (S_0) is the one constructed from the set of items containing [S' → ¤ S]