



```

S(); // Remaining input
    // (a,a)
  match("("); // a,a)
  L(); // a,a)
    S(); // a,a)
      match("a"); // ,a)
    L'(); // ,a)
      match(","); // a)
      S(); // a)
        match("a"); // )
      L'(); // )
    match(")"); //

```

---

8 a) Construct the canonical set of LR(0) items

Grammar	Augmented Grammar			
S → AS   b	S' → S	First(S') = a b	Follow(S') = \$	
A → SA   a	S → AS   b	First(S) = b a	Follow(S) = \$ a b	
	A → SA   a	First(A) = a b	Follow(A) = a b	

Note that whenever S or A is about to be parsed, we have the nonkernel LR(0) items:

$$\text{NK} = \begin{bmatrix} S \rightarrow *AS \\ S \rightarrow *b \\ A \rightarrow *SA \\ A \rightarrow *a \end{bmatrix}$$

State 0: [ S' → \*S ]  
NK items

State 1 = Goto(State 0, a) = [ A → a\* ]

State 2 = Goto(State 0, b) = [ S → b\* ]

State 3 = Goto(State 0, S) = [ S' → S\* ]  
                                   [ A → S\*A ]  
                                   [ S → \*AS ] \ NK  
                                   [ S → \*b ] \ Items  
                                   [ A → \*SA ] /  
                                   [ A → \*a ] /

State 4 = Goto(State 3, A) = [ A → SA\* ]  
                                   [ S → A\*S ]  
                                   [ S → \*AS ] \ NK  
                                   [ S → \*b ] \ Items  
                                   [ A → \*SA ] /  
                                   [ A → \*a ] /

State 5 = Goto(State 4, A) = [ S → A\*S ]  
                                   [ S → \*AS ] \ NK  
                                   [ S → \*b ] \ Items  
                                   [ A → \*SA ] /  
                                   [ A → \*a ] /



Note that whenever S or A is about to be parsed with lookahead {a,b}, we have the nonkernel LR(1) items:

$$\text{NK} = \begin{bmatrix} [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix}$$

$$\begin{aligned} \text{State 0} = \text{start state} &= \begin{bmatrix} [ S' \rightarrow *S , \$ ] \\ [ S \rightarrow *AS , \$ ] \\ [ S \rightarrow *b , \$ ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\text{State 1} = \text{Goto}(\text{State 0}, a) = [ A \rightarrow a* , \{a,b\} ]$$

$$\text{State 2} = \text{Goto}(\text{State 0}, b) = [ S \rightarrow b* , \{a,b,\$ \} ]$$

$$\begin{aligned} \text{State 3} = \text{Goto}(\text{State 0}, S) &= \begin{bmatrix} [ S' \rightarrow S* , \$ ] \\ [ A \rightarrow S*A , \{a,b\} ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\begin{aligned} \text{State 4} = \text{Goto}(\text{State 3}, A) &= \begin{bmatrix} [ A \rightarrow SA* , \{a,b\} ] \\ [ S \rightarrow A*S , \{a,b\} ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\begin{aligned} \text{State 5} = \text{Goto}(\text{State 4}, A) &= \begin{bmatrix} [ S \rightarrow A*S , \{a,b\} ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\begin{aligned} \text{State 6} = \text{Goto}(\text{State 4}, S) &= \begin{bmatrix} [ S \rightarrow AS* , \{a,b\} ] \\ [ A \rightarrow S*A , \{a,b\} ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\begin{aligned} \text{State 7} = \text{Goto}(\text{State 6}, S) &= \begin{bmatrix} [ A \rightarrow S*A , \{a,b\} ] \\ [ S \rightarrow *AS , \{a,b\} ] \\ [ S \rightarrow *b , \{a,b\} ] \\ [ A \rightarrow *SA , \{a,b\} ] \\ [ A \rightarrow *a , \{a,b\} ] \end{bmatrix} \begin{array}{l} \\ \\ \backslash \\ \backslash \\ / \\ / \end{array} \begin{array}{l} \\ \\ \text{NK} \\ \text{Items} \\ \\ \end{array} \end{aligned}$$

$$\text{State 8} = \text{Goto}(\text{State 3}, b) = [ S \rightarrow b* , \{a,b\} ]$$

$$\begin{aligned} \text{State 9} = \text{Goto}(\text{State 0}, A) &= \begin{bmatrix} [ S \rightarrow A*S , \{a,b,\$ \} ] \\ [ S \rightarrow *AS , \$ ] \end{bmatrix} \end{aligned}$$

```

[ S -> *b , $ ]
[ S -> *AS , {a,b} ] \
[ S -> *b , {a,b} ] \ NK
[ A -> *SA , {a,b} ] / Items
[ A -> *a , {a,b} ] /

```

```

State 10= Goto(State 9, S) = [ S -> AS* , {a,b,$} ]
[ A -> S*A , {a,b} ]
[ S -> *AS , {a,b} ] \
[ S -> *b , {a,b} ] \ NK
[ A -> *SA , {a,b} ] / Items
[ A -> *a , {a,b} ] /

```

[All state transitions]

```

Goto(State 0, a) = State 1
Goto(State 3, a) = State 1
Goto(State 4, a) = State 1
Goto(State 5, a) = State 1
Goto(State 6, a) = State 1
Goto(State 7, a) = State 1
Goto(State 9, a) = State 1
Goto(State 10,a) = State 1

```

```

Goto(State 0, b) = State 2
Goto(State 9, b) = State 2

```

```

Goto(State 0, S) = State 3

```

```

Goto(State 3, A) = State 4
Goto(State 6, A) = State 4
Goto(State 7, A) = State 4
Goto(State 10,A) = State 4

```

```

Goto(State 4, A) = State 5
Goto(State 5, A) = State 5

```

```

Goto(State 4, S) = State 6
Goto(State 5, S) = State 6

```

```

Goto(State 3, S) = State 7
Goto(State 6, S) = State 7
Goto(State 7, S) = State 7
Goto(State 10,S) = State 7

```

```

Goto(State 3, b) = State 8
Goto(State 4, b) = State 8
Goto(State 5, b) = State 8
Goto(State 6, b) = State 8
Goto(State 7, b) = State 8
Goto(State 10,b) = State 8

```

```

Goto(State 0, A) = State 9
Goto(State 9, A) = State 9

```

```

Goto(State 9, S) = State 10

```

(State 3 is a final state because it contains [ S' -> S\* , \$ ])

~~~~~  
 c) Construct the LR(1) action/goto tables

```

  Num    Production
  1      S' -> S
  2      S  -> AS
  3      S  -> b
  4      A  -> SA
  5      A  -> a
  
```

| State | Action   |          |     | Goto |   |
|-------|----------|----------|-----|------|---|
|       | a        | b        | \$  | S    | A |
| 0     | s1       | s2       |     | 3    | 9 |
| 1     | r5       | r5       |     |      |   |
| 2     | r3       | r3       | r3  |      |   |
| 3     | s1       | s8       | acc | 7    | 4 |
| 4     | s1<br>r4 | s8<br>r4 |     | 6    | 5 |
| 5     | s1       | s8       |     | 6    | 5 |
| 6     | s1<br>r2 | s8<br>r2 |     | 7    | 4 |
| 7     | s1       | s8       |     | 7    | 4 |
| 8     | r3       | r3       |     |      |   |
| 9     | s1       | s2       |     | 10   | 9 |
| 10    | s1<br>r2 | s8<br>r2 | r2  | 7    | 4 |

~~~~~  
 d) List and resolve conflicts

From the ACTION/GOTO tables, it's clear to see that there are shift/reduce conflicts in states 4, 6, and 10 when the lookahead is either a or b. The actual LR(1) items in conflict in the states are:

```

State    [ A -> SA* , {a,b} ]    reduce on a/b
  4      [ S -> *b , {a,b} ]    shift  on b
        [ A -> *a , {a,b} ]    shift  on a
  
```

```

States      [ S -> AS* , {a,b} ]   reduce on a/b
6 & 10     [ S -> *b  , {a,b} ]   shift  on b
           [ A -> *a  , {a,b} ]   shift  on a

```

If we always shift for a shift/reduce conflict, we would never reduce  $A \rightarrow SA$ , and we would only reduce  $S \rightarrow AS$  with  $\$$  as lookahead. This is similar to removing the production  $A \rightarrow SA$  from the grammar. It would cause us to accept only the strings "ab", "aab", "aaab", etc.

If we always reduce for a shift/reduce conflict, we never have more than two non-terminals on the stack, but should be able to generate all strings in the language.

~~~~~  
e) Example LR(1) parse for "abab"

Some rightmost derivations

```

S' -> S -> AS -> AAS -> AAb -> ASAb -> ASab -> Abab -> abab
S' -> S -> AS -> Ab -> SAB -> Sab -> ASab -> Abab -> abab

```

| Stack                 | Input      | Action/Goto                       |
|-----------------------|------------|-----------------------------------|
| \$ 0                  | a b a b \$ | action(0,a) = s1                  |
| \$ 0 a 1              | b a b \$   | action(1,b) = r5 (A -> a)         |
| \$ 0 A                | b a b \$   | goto(0,A) = 9                     |
| \$ 0 A 9              | b a b \$   | action(9,b) = s2                  |
| \$ 0 A 9 b 2          | a b \$     | action(2,a) = r3 (S -> b)         |
| \$ 0 A 9 S            | a b \$     | goto(9,S) = 10                    |
| A: \$ 0 A 9 S 10      | a b \$     | action(10,a) = s1 or r2 (S -> AS) |
|                       |            | (assume we do s1 from A:)         |
| \$ 0 A 9 S 10 a 1     | b \$       | action(1,b) = r5 (A -> a)         |
| \$ 0 A 9 S 10 A       | b \$       | goto(10,A) = 4                    |
| B: \$ 0 A 9 S 10 A 4  | b \$       | action(4,b) = s8 or r4 (A -> SA)  |
|                       |            | (assume we do s8 from B:)         |
| \$ 0 A 9 S 10 A 4 b 8 | \$         | action(8,\$) = error              |
|                       |            | (assume we do r4 from B:)         |
| \$ 0 A 9 A            | b \$       | goto(9,A) = 9                     |
| \$ 0 A 9 A 9          | b \$       | action(9,b) = s2                  |
| \$ 0 A 9 A 9 b 2      | \$         | action(2,\$) = r3 (S -> b)        |
| \$ 0 A 9 A 9 S        | \$         | goto(9,S) = 10                    |
| \$ 0 A 9 A 9 S 10     | \$         | action(10,\$) = r2 (S -> AS)      |
| \$ 0 A 9 S            | \$         | goto(9,S) = 10                    |
| \$ 0 A 9 S 10         | \$         | action(10,\$) = r2 (S -> AS)      |
| \$ 0 S                | \$         | goto(0,S) = 3                     |
| \$ 0 S 3              | \$         | action(3,\$) = accept!            |
|                       |            | (assume we do r2 from A:)         |
| \$ 0 S                | a b \$     | goto(0,S) = 3                     |
| \$ 0 S 3              | a b \$     | action(3,a) = s1                  |
| \$ 0 S 3 a 1          | b \$       | action(1,b) = r5 (A -> a)         |

|    |                  |      |                                  |
|----|------------------|------|----------------------------------|
|    | \$ 0 S 3 A       | b \$ | goto(3,A) = 4                    |
| C: | \$ 0 S 3 A 4     | b \$ | action(4,b) = s8 or r4 (A -> SA) |
|    |                  |      | (assume we do s8 from C:)        |
|    | \$ 0 S 3 A 4 b 8 | \$   | action(8,\$) = error             |
|    |                  |      | (assume we do r4 from C:)        |
|    | \$ 0 A           | b \$ | goto(0,A) = 9                    |
|    | \$ 0 A 9         | b \$ | action(9,b) = s2                 |
|    | \$ 0 A 9 b 2     | \$   | action(2,\$) = r3 (S -> b)       |
|    | \$ 0 A 9 S       | \$   | goto(9,S) = 10                   |
|    | \$ 0 A 9 S 10    | \$   | action(10,\$) = r2 (S -> AS)     |
|    | \$ 0 S           | \$   | goto(0,S) = 3                    |
|    | \$ 0 S 3         | \$   | action(3,\$) = accept!           |

9) S -> Aa | bAc | Bc | bBa  
A -> d  
B -> d

~~~~~  
LR(1) Parser

State 0: [ S -> \*Aa , \$ ]  
[ S -> \*bAc , \$ ]  
[ S -> \*Bc , \$ ]  
[ S -> \*bBa , \$ ]  
[ A -> \*d , a ]  
[ B -> \*d , c ]

State 1: Goto(State 0, d) = [ A -> d\* , a ]  
[ B -> d\* , c ]

State 2: Goto(State 0, b) = [ S -> b\*Ac , \$ ]  
[ S -> b\*Ba , \$ ]  
[ A -> \*d , c ]  
[ B -> \*d , a ]

State 3: Goto(State 2, d) = [ A -> d\* , c ]  
[ B -> d\* , a ]

~~~~~  
LALR(1) Parser

Merge lookaheads for State 1 and State 3 in LR(1) parser to create new state:

State = Merge (State 1, State 3) = [ A -> d\* , a ]  
[ A -> d\* , c ]  
[ B -> d\* , c ]  
[ B -> d\* , a ]

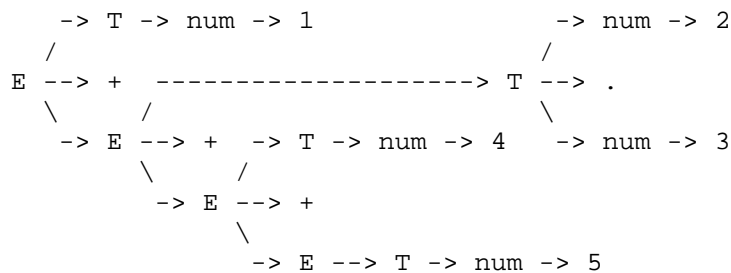
Reduce/reduce conflict for lookahead "a" and "c"  
(i.e., can't decide whether to reduce "d" to A or B)

---

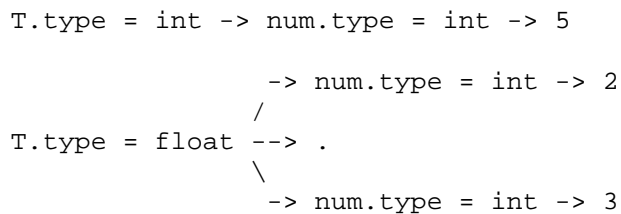
|       |              |                                                                                                  |
|-------|--------------|--------------------------------------------------------------------------------------------------|
| 10 a) | Grammar      | Attribute Rules                                                                                  |
|       | E -> E' + T  | if ((E'.type == integer) && (T.type == integer))<br>E.type = integer;<br>else<br>E.type = float; |
|       | E -> T       | E.type = T.type                                                                                  |
|       | T -> num.num | T.type = float                                                                                   |
|       | T -> num     | T.type = integer                                                                                 |

b) "5 + 4 + 3.2 + 1"

Parse Tree



Annotated Parse Tree




---

11 a) foo => array(0..99, record((a X integer) X (b X integer)))

b) bar => (integer X record((a X integer) X (b X integer))) ->  
pointer(record((a X integer) X (b X integer)))

---

12 a)

```

E1 := E2 + E3  { E1.type.lo = E2.type.lo + E3.type.lo;
                 E1.type.hi = E2.type.hi + E3.type.hi; }

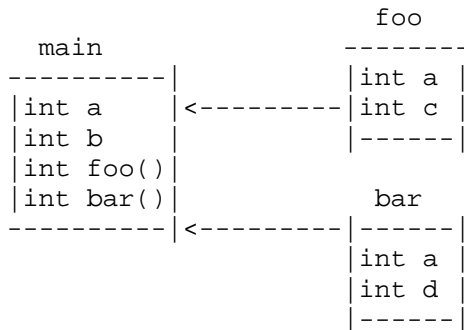
| E2 * E3     { /* assuming that all numbers are positive */
                 E1.type.lo = E2.type.lo * E3.type.lo;
                 E1.type.hi = E2.type.hi * E3.type.hi; }
  
```

---

13 Given   int a, b;  
          int foo() { int a, c; }

```
int bar() { int a, d; /* HERE */ }
```

- a) Show how symbol tables can represent the state of each scope at the point marked HERE

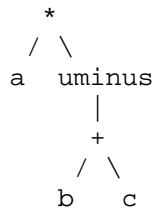


- b) what symbols are visible / not visible:

```
visible:      a (from bar), d, foo, bar, b
not visible:  a (from foo), c, a (from main)
```

14 Give the intermediate representations for  $a * - (b + c)$

- a) syntax tree



- b) postfix notation

```
a b c + uminus *      OR      b c + uminus a *
```

- c) 3-address code

|                                                         |                                                                                                 |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <pre>// abstract t1 = b + c t2 = - t1 t3 = a * t2</pre> | <pre>// load-store RTL load R1 b load R2 c add R3 R1 R2 neg R4 R1 load R5 a mult R6 R5 R4</pre> |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------|

- d) stack code

|                                                     |                                                                           |
|-----------------------------------------------------|---------------------------------------------------------------------------|
| <pre>// abstract push a push b push c add neg</pre> | <pre>// Java iload index(a) iload index(b) iload index(c) iadd ineg</pre> |
|-----------------------------------------------------|---------------------------------------------------------------------------|

mult

imul