Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

https://courses.engr.illinois.edu/cs421/sp2023

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

LR Parsing

- Read tokens left to right (L)
- Create a rightmost derivation (R)
- How is this possible?
- Start at the bottom (left) and work your way up
- Last step has only one non-terminal to be replaced so is right-most
- Working backwards, replace mixed strings by non-terminals
- Always proceed so that there are no non-terminals to the right of the string to be replaced

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

<Sum> =>

= ● (0 + 1) + 0 shift

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

<Sum> =>

= (● 0 + 1) + 0 shift

= ● (0 + 1) + 0 shift

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

<Sum> =>

=> (0 ● + 1) + 0 reduce

= (● 0 + 1) + 0 shift

= ● (0 + 1) + 0 shift

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

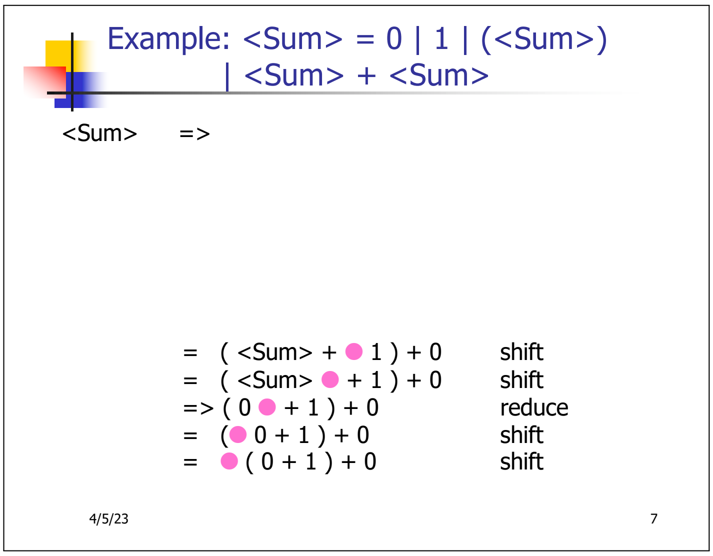
<Sum> =>

= (<Sum> ● + 1) + 0 shift

=> (0 ● + 1) + 0 reduce

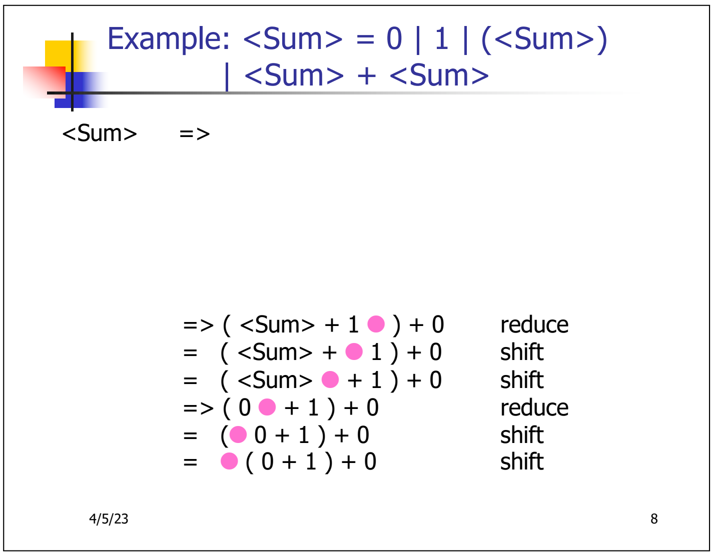
= (● 0 + 1) + 0 shift

= ● (0 + 1) + 0 shift



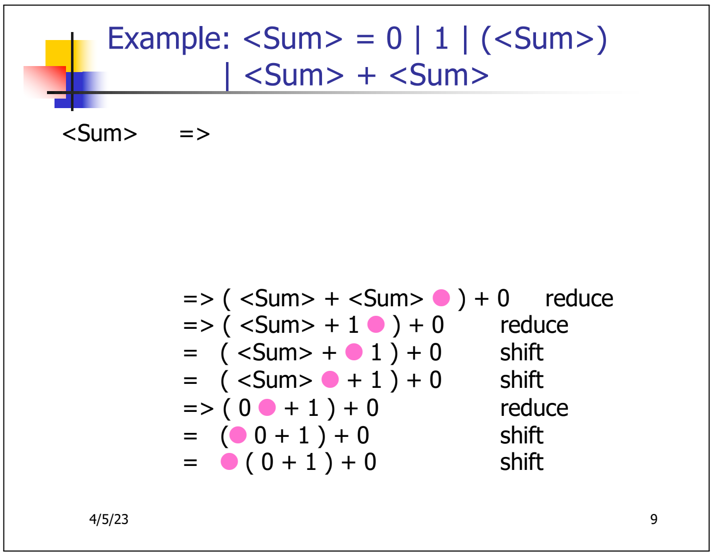
Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```



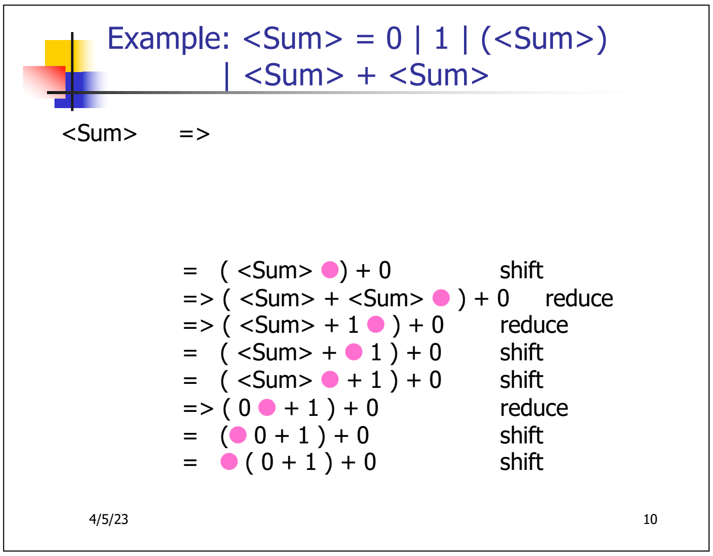
Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         => ( <Sum> + 1 ● ) + 0         reduce
         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```



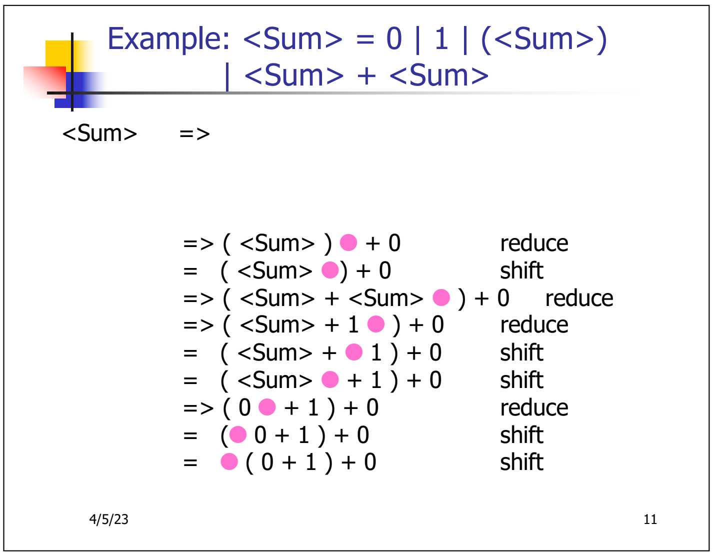
Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         => ( <Sum> + <Sum> ● ) + 0     reduce
         => ( <Sum> + 1 ● ) + 0         reduce
         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```



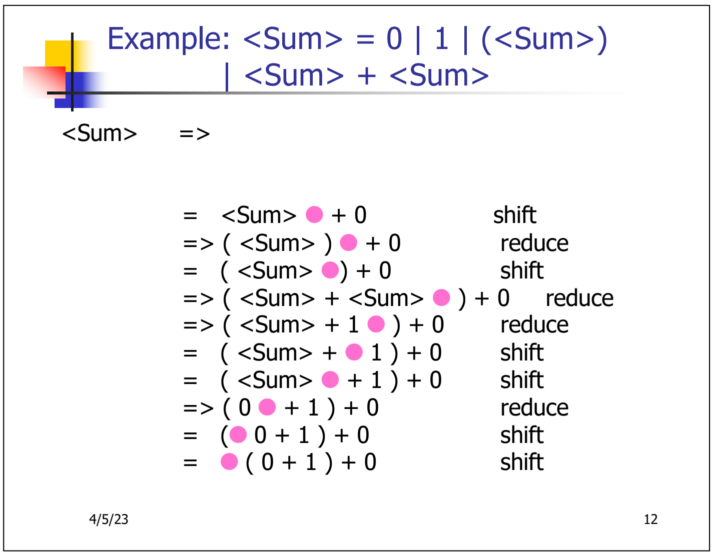
Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         =  ( <Sum> ●) + 0              shift
         => ( <Sum> + <Sum> ● ) + 0     reduce
         => ( <Sum> + 1 ● ) + 0         reduce
         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```



Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         => ( <Sum> ) ● + 0             reduce
         =  ( <Sum> ●) + 0              shift
         => ( <Sum> + <Sum> ● ) + 0     reduce
         => ( <Sum> + 1 ● ) + 0         reduce
         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```



Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>    =>

         =  <Sum> ● + 0                 shift
         => ( <Sum> ) ● + 0             reduce
         =  ( <Sum> ●) + 0              shift
         => ( <Sum> + <Sum> ● ) + 0     reduce
         => ( <Sum> + 1 ● ) + 0         reduce
         =  ( <Sum> + ● 1 ) + 0         shift
         =  ( <Sum> ● + 1 ) + 0         shift
         => ( 0 ● + 1 ) + 0             reduce
         =  (● 0 + 1 ) + 0              shift
         =  ● ( 0 + 1 ) + 0             shift
```

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>     =>

           =   <Sum> + ● 0                 shift
           =   <Sum> ● + 0                 shift
           => ( <Sum> ) ● + 0              reduce
           =   ( <Sum> ●) + 0              shift
           => ( <Sum> + <Sum> ● ) + 0      reduce
           => ( <Sum> + 1 ● ) + 0          reduce
           =   ( <Sum> + ● 1 ) + 0         shift
           =   ( <Sum> ● + 1 ) + 0         shift
           => ( 0 ● + 1 ) + 0              reduce
           =   (● 0 + 1 ) + 0              shift
           =   ● ( 0 + 1 ) + 0             shift
```

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>     =>
           => <Sum> + 0 ●                  reduce
           =   <Sum> + ● 0                 shift
           =   <Sum> ● + 0                 shift
           => ( <Sum> ) ● + 0              reduce
           =   ( <Sum> ●) + 0              shift
           => ( <Sum> + <Sum> ● ) + 0      reduce
           => ( <Sum> + 1 ● ) + 0          reduce
           =   ( <Sum> + ● 1 ) + 0         shift
           =   ( <Sum> ● + 1 ) + 0         shift
           => ( 0 ● + 1 ) + 0              reduce
           =   (● 0 + 1 ) + 0              shift
           =   ● ( 0 + 1 ) + 0             shift
```

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

```
<Sum>     => <Sum> + <Sum > ●             reduce
           => <Sum> + 0 ●                  reduce
           =   <Sum> + ● 0                 shift
           =   <Sum> ● + 0                 shift
           => ( <Sum> ) ● + 0              reduce
           =   ( <Sum> ●) + 0              shift
           => ( <Sum> + <Sum> ● ) + 0      reduce
           => ( <Sum> + 1 ● ) + 0          reduce
           =   ( <Sum> + ● 1 ) + 0         shift
           =   ( <Sum> ● + 1 ) + 0         shift
           => ( 0 ● + 1 ) + 0              reduce
           =   (● 0 + 1 ) + 0              shift
           =   ● ( 0 + 1 ) + 0             shift
```

Example: <Sum> = 0 | 1 | (<Sum>) | <Sum> + <Sum>

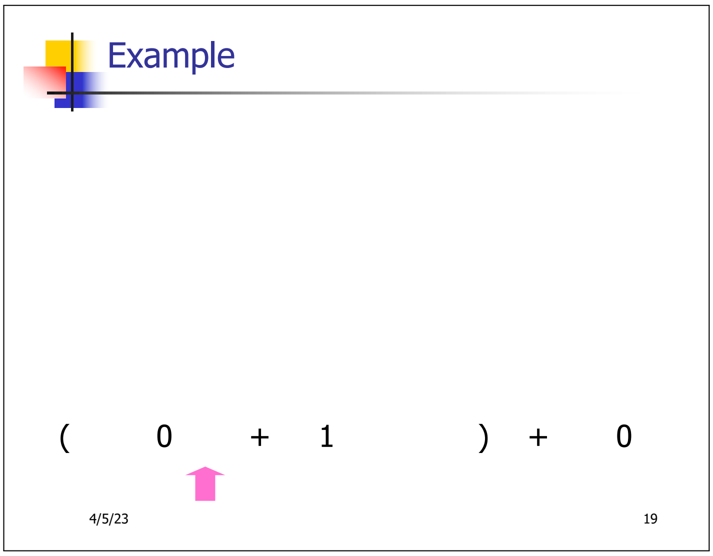
```
<Sum> ●   => <Sum> + <Sum > ●             reduce
           => <Sum> + 0 ●                  reduce
           =   <Sum> + ● 0                 shift
           =   <Sum> ● + 0                 shift
           => ( <Sum> ) ● + 0              reduce
           =   ( <Sum> ●) + 0              shift
           => ( <Sum> + <Sum> ● ) + 0      reduce
           => ( <Sum> + 1 ● ) + 0          reduce
           =   ( <Sum> + ● 1 ) + 0         shift
           =   ( <Sum> ● + 1 ) + 0         shift
           => ( 0 ● + 1 ) + 0              reduce
           =   (● 0 + 1 ) + 0              shift
           =   ● ( 0 + 1 ) + 0             shift
```

Example

```
 (      0     +    1          )   +     0
⬆
```

Example

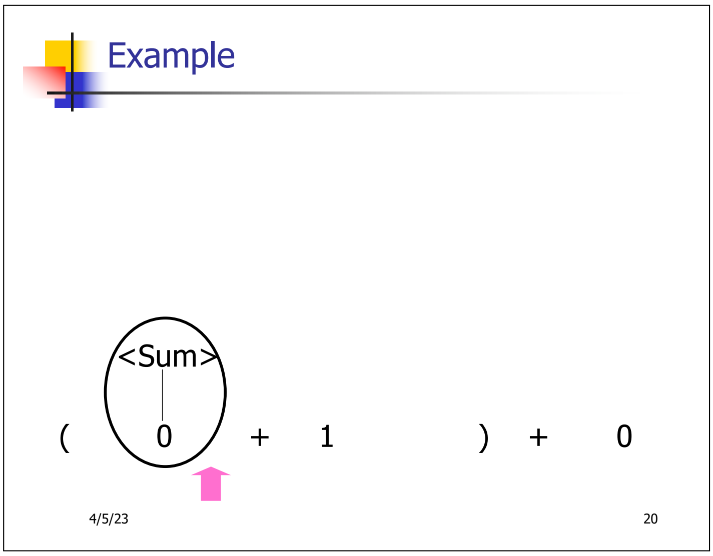
```
(      0     +    1          )   +     0
   ⬆
```



Example

(0 + 1) + 0

4/5/23 19

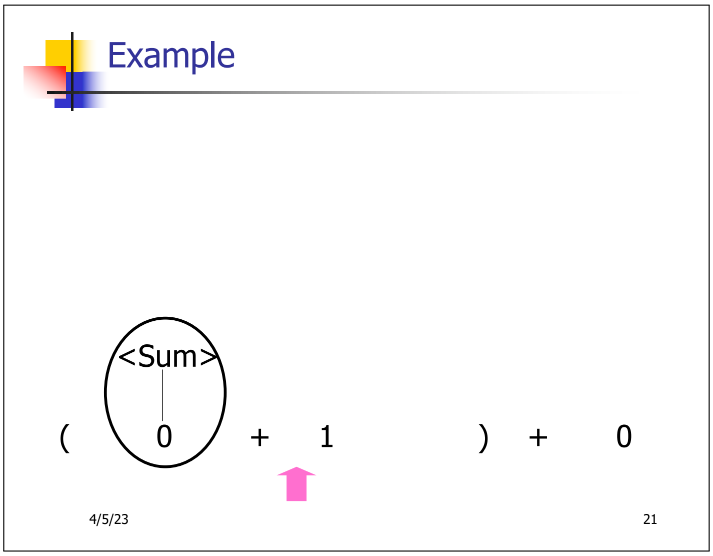


Example

<Sum>

(0 + 1) + 0

4/5/23 20

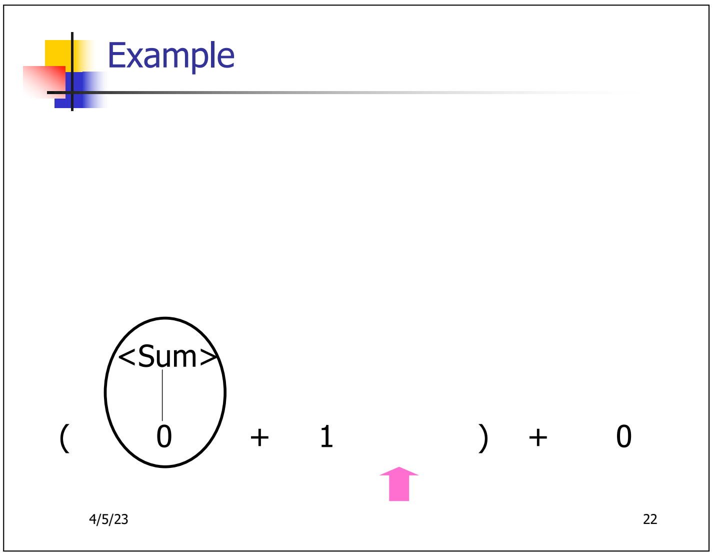


Example

<Sum>

(0 + 1) + 0

4/5/23 21

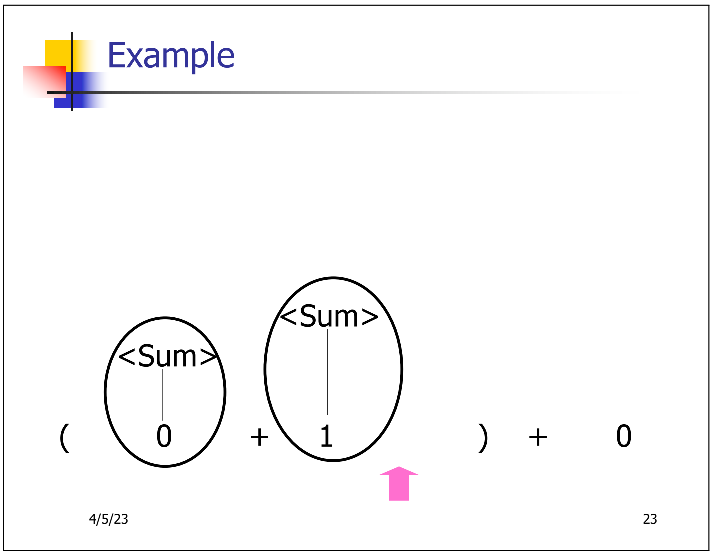


Example

<Sum>

(0 + 1) + 0

4/5/23 22

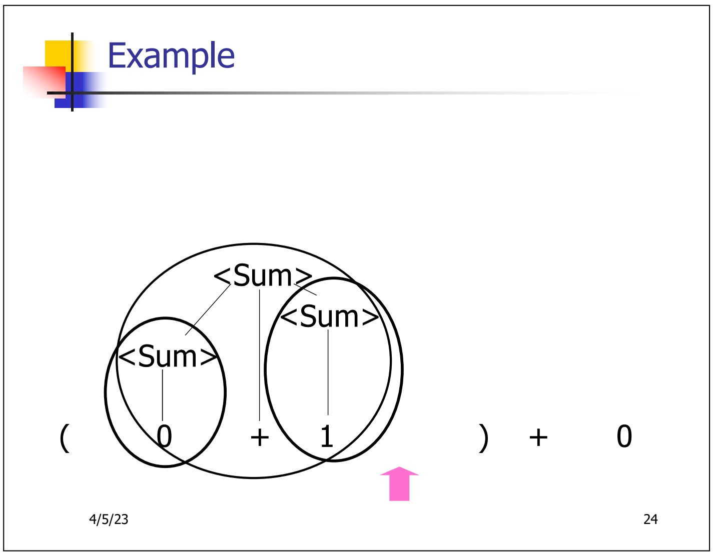


Example

<Sum> <Sum>

(0 + 1) + 0

4/5/23 23



Example

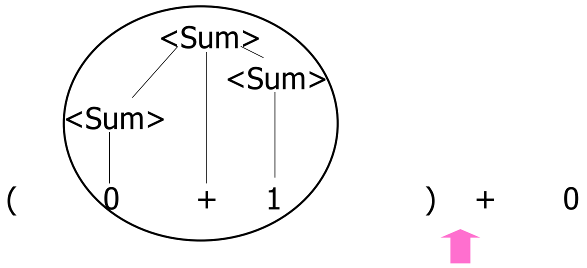
<Sum>

<Sum> <Sum>

(0 + 1) + 0

4/5/23 24

Example



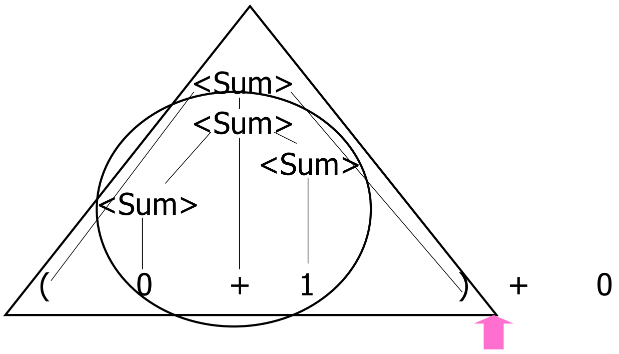
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



<Sum>

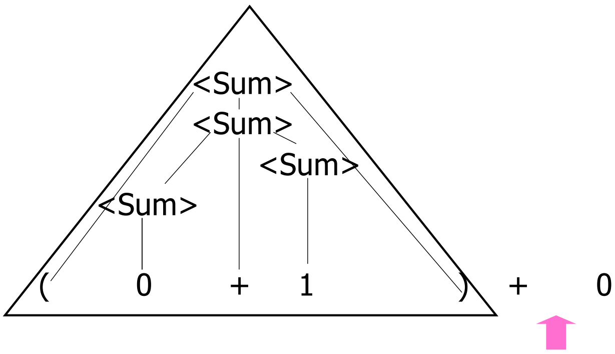
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



<Sum>

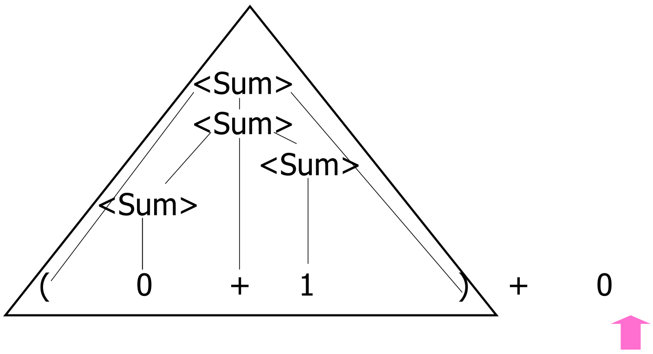
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



<Sum>

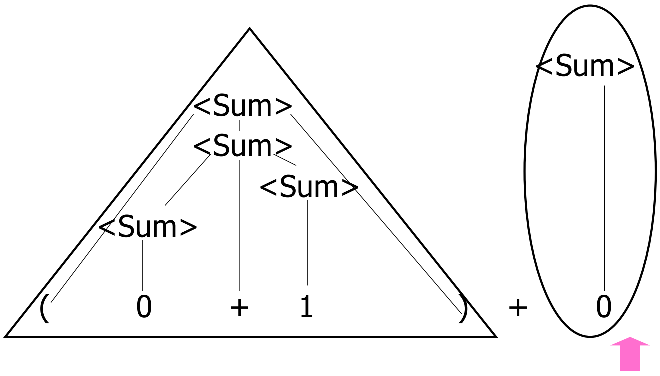
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



<Sum>

<Sum>

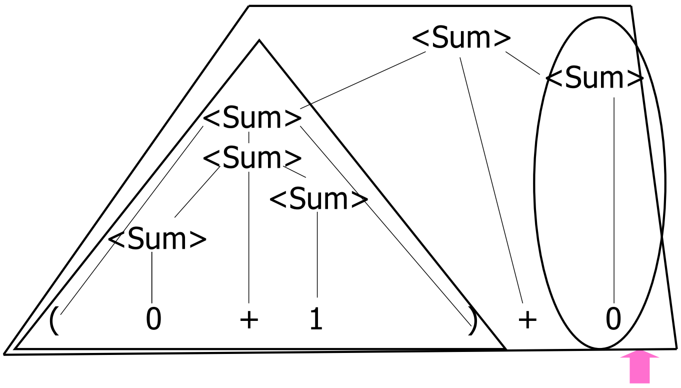
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



<Sum>

<Sum>

<Sum>

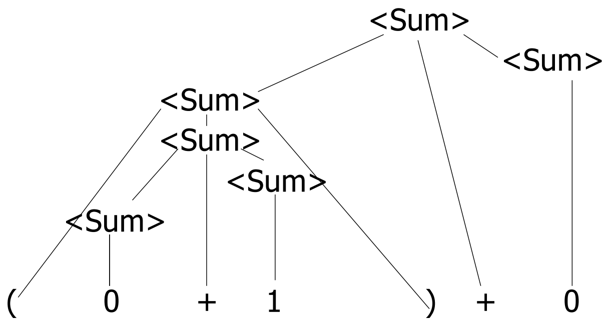
<Sum>

<Sum>

<Sum>

(0 + 1) + 0

Example



LR Parsing Tables

- Build a pair of tables, Action and Goto, from the grammar
  - This is the hardest part, we omit here
  - Rows labeled by states
  - For Action, columns labeled by terminals and "end-of-tokens" marker
    - (more generally strings of terminals of fixed length)
  - For Goto, columns labeled by non-terminals

Action and Goto Tables

- Given a state and the next input, Action table says either
  - **shift** and go to state *n*, or
  - **reduce** by production *k* (explained in a bit)
  - **accept** or **error**
- Given a state and a non-terminal, Goto table says
  - go to state *m*

LR(i) Parsing Algorithm

- Based on push-down automata
- Uses states and transitions (as recorded in Action and Goto tables)
- Uses a stack containing states, terminals and non-terminals

LR(i) Parsing Algorithm

0. Insure token stream ends in special "end-of-tokens" symbol
1. Start in state 1 with an empty stack
2. Push **state**(1) onto stack

→3. Look at next *i* tokens from token stream (*toks*) (don' t remove yet)

4. If top symbol on stack is **state**(*n*), look up action in Action table at (*n*, *toks*)

LR(i) Parsing Algorithm

5. If action = **shift** *m*,
   a) Remove the top token from token stream and push it onto the stack
   b) Push **state**(*m*) onto stack
   c) Go to step 3

LR(i) Parsing Algorithm

6. If action = **reduce** *k* where production *k* is E ::= u
   a) Remove 2 * length(u) symbols from stack (u and all the interleaved states)
   b) If new top symbol on stack is **state**(*m*), look up new state *p* in Goto(*m*,E)
   c) Push E onto the stack, then push **state**(*p*) onto the stack
   d) Go to step 3

LR(i) Parsing Algorithm

7. If action = **accept**
   - Stop parsing, return success
8. If action = **error**,
   - Stop parsing, return failure

Adding Synthesized Attributes

- Add to each **reduce** a rule for calculating the new synthesized attribute from the component attributes
- Add to each non-terminal pushed onto the stack, the attribute calculated for it
- When performing a **reduce**,
  - gather the recorded attributes from each non-terminal popped from stack
  - Compute new attribute for non-terminal pushed onto stack

Shift-Reduce Conflicts

- **Problem**: can't decide whether the action for a state and input character should be **shift** or **reduce**
- Usually caused by lack of associativity or precedence information in grammar
  - Can be that the grammar needs the parser to look at more than the next token

1425 minutes

Example: <Sum> = 0 | 1 | (<Sum>)
| <Sum> + <Sum>

● 0 + 1 + 0	shift
-> 0 ● + 1 + 0	reduce
-> <Sum> ● + 1 + 0	shift
-> <Sum> + ● 1 + 0	shift
-> <Sum> + 1 ● + 0	reduce
-> <Sum> + <Sum> ● + 0	

Example - cont

- **Problem:** shift or reduce?
- You can shift-shift-reduce-reduce or reduce-shift-shift-reduce
- Shift first - right associative
- Reduce first- left associative

Reduce - Reduce Conflicts

- **Problem:** can’t decide between two different rules to reduce by
- **Symptom:** RHS of one production suffix of another
- Requires examining grammar and rewriting it
- Harder to solve than shift-reduce errors

Example

- S ::= A | aB   A ::= abc   B ::= bc

● abc	shift
a ● bc	shift
ab ● c	shift
abc ●	

- Problem: reduce by B ::= bc then by S ::= aB, or by A::= abc then S::A?