SYNTAX ANALYSIS
BOTTOM-UP PARSING
Last Time

• Parsing
  – Top-down or bottom-up

• Top-down parsing
  – Recursive descent
  – LL(k) grammars
  – LL(k) parsing with pushdown automata

• LL(k) parsers
  – Cannot deal with common prefixes and left recursion
  – Left-recursion removal might result in complicated grammar
Parser Classes – Reminder

- Top-down (predictive)

- Bottom-up (shift-reduce)
Parser Classes – Reminder

- **Top-down (predictive)**

- **Bottom-up (shift-reduce)**

![Diagram illustrating parser classes]

- Sentential form representing x
- Already read...
- To be read
LR(k) Grammars

• A grammar is in the class LR(k) when it can be derived via:
  ‣ **Bottom-up** analysis
  ‣ Scanning the input from **left to right** (L)
  ‣ Producing the **rightmost derivation** (R)
  ‣ With **lookahead** of k tokens (k)

• A language is said to be LR(k) if it has an LR(k) grammar

• The simplest case is LR(0), which we discuss next
LR is More Powerful, But…

• Any LL(k) language is also in LR(k) (and not vice versa), *i.e.*, LL(k) ⊂ LR(k)

• **But** the lookahead is counted differently in the two cases:
  ‣ With LL(k), the algorithm sees k tokens of the right-hand side of the rule and then must select the derivation rule
  ‣ With LR(k), the algorithm sees all right-hand side of the derivation rule plus k more tokens
    • LR(0) sees the entire right-side
LR is More Powerful, But...

• Any LL(k) language is also in LR(k) (and not vice versa), \( \text{i.e.,} \) \( \text{LL}(k) \subset \text{LR}(k) \)

• **But** the lookahead is counted differently in the two cases:
  - With LL(k), the algorithm sees \( k \) tokens of the right-hand side of the rule and then must select the derivation rule
  - With LR(k), the algorithm sees **all** right-hand side of the derivation rule plus \( k \) more tokens
    • LR(0) sees the entire right-side

• The LR family of parsers is more popularly used today
Example: a Simple LR(0) Grammar

\[ E \rightarrow E \ast B \mid E + B \mid B \]
\[ B \rightarrow 0 \mid 1 \]

- Let us number the rules:
  
  (1) \( E \rightarrow E \ast B \)
  
  (2) \( E \rightarrow E + B \)
  
  (3) \( E \rightarrow B \)
  
  (4) \( B \rightarrow 0 \)
  
  (5) \( B \rightarrow 1 \)
Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[
E \rightarrow E \ast B \mid E + B \mid B
\]

\[
B \rightarrow 0 \mid 1
\]

0 + 0 * 1

B + 0 * 1

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[ 0 + 0 \times 1 \]
\[ B + 0 \times 1 \]
\[ E + 0 \times 1 \]

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → E * B</td>
<td></td>
<td>(1)</td>
</tr>
<tr>
<td>E + B</td>
<td></td>
<td>(2)</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>(3)</td>
</tr>
<tr>
<td>B → 0</td>
<td></td>
<td>(4)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>(5)</td>
</tr>
</tbody>
</table>

Example:

0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

<table>
<thead>
<tr>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → E * B</td>
<td>E + B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B → 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>(5)</td>
<td></td>
</tr>
</tbody>
</table>

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[
\begin{align*}
0 + 0 * 1 \\
B + 0 * 1 \\
E + 0 * 1 \\
E + B * 1 \\
E * 1 \\
E * B \\
E
\end{align*}
\]

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E + 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[ 0 + 0 \ast 1 \]
\[ B + 0 \ast 1 \]
\[ E + 0 \ast 1 \]
\[ E + B \ast 1 \]
\[ E \ast 1 \]
\[ E \ast B \]
\[ E \]

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[ 0 + 0 \times 1 \]
\[ B + 0 \times 1 \]
\[ E + 0 \times 1 \]
\[ E + B \times 1 \]
\[ E \times 1 \]
\[ E \times B \]
\[ E \]

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

\[ 0 + 0 \times 1 \]
\[ B + 0 \times 1 \]
\[ E + 0 \times 1 \]
\[ E + B \times 1 \]
\[ E \times 1 \]
\[ E \times B \]
\[ E \]

Go over the input so far, and upon seeing a right-hand side of a rule, "invoke" the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
\[
0 + 0 \ast 1
\]
\[
B + 0 \ast 1
\]
\[
E + 0 \ast 1
\]
\[
E + B \ast 1
\]
\[
E \ast 1
\]
\[
E \ast B
\]
\[
E
\]

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:

0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Goal: Reduce the Input to the Start Symbol

Example:
0 + 0 * 1
B + 0 * 1
E + 0 * 1
E + B * 1
E * 1
E * B
E

Go over the input so far, and upon seeing a right-hand side of a rule, “invoke” the rule and replace the right-hand side with the left-hand side (which is a single non-terminal)
Shift & Reduce

In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E * B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>E + B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

(1) E → E * B | (2) E + B | (3) B
(4) B → 0    | (5) B → 1

---

(1) E
(2) E * B
(3) E + B
(4) B → 0
(5) B → 1

Stack | Input | Action | E | E | * | B | E | + | B | 1 | B | 0 | 0
In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → E * B</td>
<td>0 + 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>B → 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B → 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shift & Reduce**

E → E * B | E + B | B
(1) (2) (3)

B → 0 | 1
(4) (5)
Shift & Reduce

In each step, we either shift a symbol from the input to the stack, or reduce according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0+0*1 $</td>
<td>shift (2)</td>
</tr>
<tr>
<td>0</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
</tbody>
</table>

**Grammar Rules**

\[
E \rightarrow E \cdot B | E + B | B \\
B \rightarrow 0 | 1
\]

**Sample Computation**

1. **Input:** 0+0*1
   - **Stack:** 0
   - **Input:** 0+0*1
     - **Action:** shift
   - **Stack:** 0
   - **Input:** +0*1
     - **Action:** reduce (4)

**Parse Tree**

```
  E
 /|
/  \
E + B
   /|
   /  \
 B  1
  /|
 /  \
B  0
```

(1) E → E * B | E + B | B
(2) B → 0 | 1
(3) (4) (5)
Shift & Reduce

In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

$$E \rightarrow E \ast B \mid E + B \mid B$$  
$$B \rightarrow 0 \mid 1$$

---

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0 + 0 \ast 1$</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>$+ 0 \ast 1$</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>$+ 0 \ast 1$</td>
<td>reduce (3)</td>
</tr>
</tbody>
</table>

---

Diagram:

```
E E * B E + B B
```

```
0 + 0 * 1
```

```
0 1
```

```
B 0
```

```
0
```

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0 + 0 \ast 1$</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>$+ 0 \ast 1$</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>$+ 0 \ast 1$</td>
<td>reduce (3)</td>
</tr>
</tbody>
</table>
Shift & Reduce

In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+0*1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+0*1 $</td>
<td>shift</td>
</tr>
</tbody>
</table>
In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0 + 0 * 1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 + 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+ 0 * 1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+ 0 * 1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+ 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>0 * 1 $</td>
<td>shift</td>
</tr>
</tbody>
</table>
In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>B</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E</td>
<td>+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+0</td>
<td>*1 $</td>
<td>reduce (4)</td>
</tr>
</tbody>
</table>
In each step, we either shift a symbol from the input to the stack, or reduce according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 + 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+ 0 * 1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+ 0 * 1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+ 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E +</td>
<td>0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E + 0</td>
<td>* 1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E + B</td>
<td>* 1 $</td>
<td>reduce (2)</td>
</tr>
</tbody>
</table>
In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+0*1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+0</td>
<td>*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E+B</td>
<td>*1 $</td>
<td>reduce (2)</td>
</tr>
<tr>
<td>E</td>
<td>*1 $</td>
<td>shift</td>
</tr>
</tbody>
</table>
Shift & Reduce

In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 + 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+ 0 * 1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+ 0 * 1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+ 0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E +</td>
<td>0 * 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E + 0</td>
<td>* 1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E + B</td>
<td>* 1 $</td>
<td>reduce (2)</td>
</tr>
<tr>
<td>E</td>
<td>* 1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E *</td>
<td>1 $</td>
<td>shift</td>
</tr>
</tbody>
</table>
Shift & Reduce

In each step, we either shift a symbol from the input to the stack, or reduce according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+0*1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+0</td>
<td>*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E+B</td>
<td>*1 $</td>
<td>reduce (2)</td>
</tr>
<tr>
<td>E</td>
<td>*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E*</td>
<td>1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E*1</td>
<td>$</td>
<td>reduce (5)</td>
</tr>
</tbody>
</table>
In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>+0*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>+0*1 $</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>+0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>0*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E+0</td>
<td>*1 $</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E+B</td>
<td>*1 $</td>
<td>reduce (2)</td>
</tr>
<tr>
<td>E</td>
<td>*1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E*</td>
<td>1 $</td>
<td>shift</td>
</tr>
<tr>
<td>E*1</td>
<td>$</td>
<td>reduce (5)</td>
</tr>
<tr>
<td>E*B</td>
<td>$</td>
<td>reduce (1)</td>
</tr>
</tbody>
</table>
Shift & Reduce

In each step, we either **shift** a symbol from the input to the stack, or **reduce** according to one of the rules. Example: “0+0*1”.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0+0*1$</td>
<td>shift</td>
</tr>
<tr>
<td>0</td>
<td>$+0*1$</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>B</td>
<td>$+0*1$</td>
<td>reduce (3)</td>
</tr>
<tr>
<td>E</td>
<td>$+0*1$</td>
<td>shift</td>
</tr>
<tr>
<td>E+</td>
<td>$0*1$</td>
<td>shift</td>
</tr>
<tr>
<td>E+0</td>
<td>$*1$</td>
<td>reduce (4)</td>
</tr>
<tr>
<td>E+B</td>
<td>$*1$</td>
<td>reduce (2)</td>
</tr>
<tr>
<td>E</td>
<td>$*1$</td>
<td>shift</td>
</tr>
<tr>
<td>E*</td>
<td>$1$</td>
<td>shift</td>
</tr>
<tr>
<td>E*1</td>
<td>$$</td>
<td>reduce (5)</td>
</tr>
<tr>
<td>E*B</td>
<td>$$</td>
<td>reduce (1)</td>
</tr>
<tr>
<td>E</td>
<td>$$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Shift / Reduce Parser — Intuition

• Gather input token by token
  ‣ until we find a right-hand side of a rule
  ‣ then, replace it with the non-terminal on the left hand side

• Going over a token and recording it in the stack is a shift
  ‣ Each shift moves to a state that records what we’ve seen so far

• A reduce replaces a string on the stack with a nonterminal that derives it
LR(0) Item

For a production rule in the grammar, \[ N \rightarrow \alpha \beta \]

So far we’ve matched \( \alpha \), expecting to see \( \beta \)
LR(0) Item

\[
E \rightarrow E \ast B \mid E + B \mid B \\
B \rightarrow 0 \mid 1
\]

\[
E \rightarrow E \bullet \ast B \\
E \rightarrow E \ast B \bullet
\]
LR(0) Item

\[ E \rightarrow E \ast B \mid E + B \mid B \]
\[ B \rightarrow 0 \mid 1 \]

E → E•* B  \hspace{2cm} \text{Shift Item}

E → E * B•
LR(0) Item

\[ E \rightarrow E \ast B \mid E + B \mid B \]

\[ B \rightarrow 0 \mid 1 \]

**Shift Item**

\[ E \rightarrow E \ast B \]

**Reduce Item**

\[ E \rightarrow E \ast B \bullet \]
Example: Parsing with LR(0) Items

\[
\begin{align*}
Z &\rightarrow \text{expr } \$ \\
\text{expr} &\rightarrow \text{term} \mid \text{expr} + \text{term} \\
\text{term} &\rightarrow \text{ID} \mid ( \text{expr} )
\end{align*}
\]
Example: Parsing with LR(0) Items

\[
\begin{align*}
Z & \rightarrow \text{expr } \$$ \\
\text{expr} & \rightarrow \text{term} \mid \text{expr } + \text{term} \\
\text{term} & \rightarrow \text{ID} \mid ( \text{expr} ) \\
\end{align*}
\]

\[
\begin{align*}
Z & \rightarrow \text{E } \$ \\
\text{E} & \rightarrow \text{T} \mid \text{E } + \text{T} \\
\text{T} & \rightarrow \text{i} \mid ( \text{E} ) \\
\end{align*}
\]

(just a shorthand of the grammar on top)
Example: Parsing with LR(0) Items

input

\[ i + i \] $\$

Z → E $\$
E → T | E + T
T → i | ( E )
Example: Parsing with LR(0) Items

input

\[ i + i \] $\$

\[ Z \rightarrow E \] $

\[ E \rightarrow T \mid E + T \]

\[ T \rightarrow i \mid (E) \]

\[ Z \rightarrow \cdot E \] $\$

14
Example: Parsing with LR(0) Items

input

\[ i + i \ $ \]

\[
\begin{align*}
Z & \rightarrow \ E \ \$ \\
E & \rightarrow \ T \\
E & \rightarrow \ E + T \\
T & \rightarrow \ i \\
T & \rightarrow \ ( \ E )
\end{align*}
\]
Example: Parsing with LR(0) Items

input

\[ i + i \] $ \]

\[
\begin{align*}
Z & \rightarrow E \ $ \\
E & \rightarrow T \mid E + T \\
T & \rightarrow i \mid (E) \\
\end{align*}
\]

Closure

\[
\begin{align*}
Z & \rightarrow \cdot E \ $ \\
E & \rightarrow \cdot T \\
E & \rightarrow \cdot E + T \\
T & \rightarrow \cdot i \\
T & \rightarrow \cdot (E) \\
\end{align*}
\]
input $i + i$ \$

Z \to E \ $
E \to T \mid E + T
T \to i \mid (E)

Z \to \bullet E \$
E \to \bullet T
E \to \bullet E + T
T \to \bullet i
T \to \bullet (E)
input

\[ i + i \]

\[ Z \rightarrow E \$
\[ E \rightarrow T | E + T \]
\[ T \rightarrow i | ( E ) \]

\[ Z \rightarrow \bullet E \$
\[ E \rightarrow \bullet T \]
\[ E \rightarrow \bullet E + T \]
\[ T \rightarrow \bullet i \]
\[ T \rightarrow \bullet (E) \]
input

\[ i + i \]

\[
\begin{align*}
Z &\rightarrow E \,$ \\
E &\rightarrow T | E + T \\
T &\rightarrow i | (E)
\end{align*}
\]
input

Z → E $ 
E → T | E + T 
T → i | ( E )

Shift
input

Z → E $  
E → T | E + T  
T → i | (E)
input

i + i $

Shift

Z → E $
E → T | E + T
T → i | ( E )

Z → •E $
E → •T
E → •E + T
T → •i
T → •( E )

Reduce item!
input \[ i + i \] $ \\

\text{Reduce item!}

\begin{align*}
Z & \rightarrow \text{E} \ \$ \\
E & \rightarrow T \ | \ E + T \\
T & \rightarrow i \ | \ (\text{E})
\end{align*}
input: 

\[ i + i \]

Production Rules:

- **Z** → E $ 
- **E** → T | E + T 
- **T** → i | (E) 

Circular Diagram:
input \quad i + i \; \$
input $i + i$ $\$

Z → E $\$
E → T | E + T
T → i | ( E )

Reduce item!
input: \( i + i \) $

\text{Reduce item!}

\begin{align*}
Z & \rightarrow E \$
E & \rightarrow T \mid E + T \\
T & \rightarrow i \mid (E)
\end{align*}
input: \(i + i\) $
input

i + i $
input

Z → E $  
E → T | E + T  
T → i | ( E )

Z → E •$  
E → E •+ T  
T → i  
T → ( E )

Z → E •$  
E → E •+ T  
T → i  
T → ( E )

Z → E •$  
E → E •+ T  
T → i  
T → ( E )
input

\[ i + i \rightarrow E \]

```
Z → E $  
E → T | E + T  
T → i | ( E )
```

```
Z → E •$  
E → E •+ T
```

```
Z → E $  
E → T | E + T  
T → i | ( E )
```

Shift
input

\[ \text{E} \rightarrow \text{T} \mid \text{E} + \text{T} \]

\[ \text{T} \rightarrow \text{i} \mid (\text{E}) \]

\[ \text{Z} \rightarrow \text{E} \$
\]

\[ \text{E} \rightarrow \text{T} \mid \text{E} + \text{T} \]

\[ \text{E} \rightarrow \text{E} + \text{T} \]

\[ \text{T} \rightarrow \text{i} \mid (\text{E}) \]

\[ \text{Z} \rightarrow \text{E} \$
\]

\[ \text{E} \rightarrow \text{E} + \text{T} \]

\[ \text{T} \rightarrow \text{i} \mid (\text{E}) \]

\[ \text{Z} \rightarrow \text{E} \$
\]

\[ \text{E} \rightarrow \text{E} + \text{T} \]

\[ \text{T} \rightarrow \text{i} \mid (\text{E}) \]

\[ \text{Z} \rightarrow \text{E} \$
\]
input \[i + i \] $
input \[ i + i \]
input

i + (i $

Z → E $
E → T | E + T
T → i | (E)

Z → E $\cdot$
Z → E $\cdot$
E → E $\cdot$
T → i
T → (E)
T → i

E → E $\cdot$
T → i
T → (E)
input

$ i + i $  

Shift

\[
\begin{align*}
    &Z \rightarrow E \  \$ \\
    &E \rightarrow T \mid E + T \\
    &T \rightarrow i \mid (E)
\end{align*}
\]
input

```
i + i
```

Output

```
Z → E $
E → T | E + T
T → i | (E)
```

Production Rules

```
E → E + T
E → E • T
Z → E • $
Z → •E $
E → •T
E → •E + T
T → •i
T → •(E)
```

Shift

�

X
input

\[ i + i \$ \]

\[
\begin{align*}
Z & \rightarrow E \$ \\
E & \rightarrow T \mid E + T \\
T & \rightarrow i \mid (E)
\end{align*}
\]

\[
\begin{align*}
Z & \rightarrow E \cdot $ \\
E & \rightarrow E \cdot + T \\
T & \rightarrow i \\
T & \rightarrow (E)
\end{align*}
\]

\[
\begin{align*}
E & \rightarrow E \cdot + T \\
T & \rightarrow i \\
T & \rightarrow (E)
\end{align*}
\]
input $ i + i $ $

Z \rightarrow E \; $ $\$
E \rightarrow T \mid E + T$ $
T \rightarrow i \mid (E)$

Z \rightarrow E \; $ $\$
E \rightarrow E \cdot + T$ $
T \rightarrow T \cdot i$ $\$
T \rightarrow T \cdot (E)$

Z \rightarrow E \; $ $\$
E \rightarrow E \cdot + T$ $
E \rightarrow T \mid E + T$ $
T \rightarrow T \cdot i$ $\$
T \rightarrow T \cdot (E)$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$

$\$
input \[ i + i \] $
input $i + i$

```
Z → E $
E → T | E + T
T → i | ( E )
```

```
Z → •E $
E → •T
E → •E + T
T → •i
T → •( E )
```
input  \( i + i \) $

\begin{align*}
Z & \rightarrow E \, \$ \\
E & \rightarrow T \mid E + T \\
T & \rightarrow i \mid (E)
\end{align*}

\[
\begin{array}{c}
E \rightarrow T \\
E \rightarrow E + T \\
T \rightarrow i \\
T \rightarrow (E)
\end{array}
\]

\[
\begin{array}{c}
E \rightarrow E + T \\
T \rightarrow i \\
T \rightarrow (E)
\end{array}
\]
input: $i + i$

Production rules:

- $Z \rightarrow E \, \$ \quad (Z → E \, \$)
- $E \rightarrow T \mid E + T \quad (E → T \mid E + T)$
- $T \rightarrow i \mid (E) \quad (T → i \mid (E))$
- $E \rightarrow E + T$ \quad (E → E+T)
- $E \rightarrow i$ \quad (E → i)
- $E \rightarrow \, (E)$ \quad (E → (E))
input $ i + i \$ 

Z → E $ 
E → T | E + T 
T → i | ( E ) 

Reduce item!

Z → E$ 

Z → E$ 

E → E+T 
T → i 
T → ( E ) 

E → E+T 

Z → E$ 
E → E+T 
T → i 
T → ( E ) 

E → E+T 

Z → E$ 
E → T | E + T 
T → i | ( E )
input

```
i + i $
```

```
Z → E $
E → T | E + T
T → i | ( E )
```

```
Z → •E $
E → •T
E → •E + T
T → •i
T → •( E )
```

```
Z → •E $
E → •E + T
```

```
```

```
Z → E•$
E → E•+ T
```
input

\[ i + i \] $\]

**Shift**

- **Z → E $**
- **E → T | E + T**
- **T → i | (E)**

**Parse Tree**

- **Z → •E $**
- **E → •T**
- **E → •E + T**
- **T → •i**
- **T → •(E)**

- **Valid Parse**
  - **Z → E •$**
  - **E → E •+ T**
  - **✓**

- **Invalid Parse**
  - **Z → E •$**
  - **E → E •+ T**
  - **✗**
input \( i + i \) $
Reducing the initial rule means accept

Z → E $
E → T | E + T
T → i | ( E )
View as an LR(0) Automaton

Z → •E $  
E → •E + T  
T → •i  
T → •(E)

E → T•  
T → (•E)  
E → •T  
T → •i  
T → •(E)

T → (•E)  
E → •E + T  
T → •i  
T → •(E)

Z → E•$  
E → E•+ T

Z → E•$  
E → E•+ T

(accept)

E → E + T•  
E → E + T•  
E → E•+T
How does the parser know what to do?

- Pushdown Automaton!
  - A state will keep the info gathered so far
  - A table will tell it “what to do” based on current state and next token
  - Some info will be kept in a stack
LR(0) Parsing

Input

Stack

Automaton

Output

ACTION Table

GOTO Table
Why do we need a stack?

- Suppose so far we have discovered $E \rightarrow B \rightarrow 0$ and $+$; 
  So we have constructed sentential form “$E +$”.
- In the given grammar this can only mean
  
  $$E \rightarrow E + \circ B$$

- Suppose current state $q_6$ represents this situation.
- Now, the next token is $0$, and we need to ignore $q_6$ for a minute, and work on $B \rightarrow 0$ to obtain $E + B$.
- Therefore, we push $q_6$ to the stack, and after identifying $B$, we pop it to continue.
The Stack

- The stack contains states
- For readability we also include variables and tokens (the recognizer does not need them)
- The initial stack contains $q_0$ only
- Apart from $q_0$ at the bottom of the stack, the rest of the stack contains pairs of (state, token) or (state, nonterminal)
The ACTION Table

- At each step we need to decide whether to **shift** the next token to the stack (and move to the appropriate state) or **reduce** a production rule from the grammar.

- The ACTION table tells us what to do based on current state and next token:
  
  **shift** $n$ : shift and move to $q_n$

  **reduce** $m$: reduce according to production rule ($m$)

  (also: accept and error conditions)
The GOTO Table

• Defines what to do on reduce actions
• After reducing a right-hand side to the deriving non-terminal, we need to decide what the next state is
• This is determined by the previous state (which is on the stack) and the variable we got
  ▶ Suppose we reduce according to $N \to \beta$;
  ▶ We remove $\beta$ from the stack, and look at the state $q$ that is now at the top. GOTO[$q, N$] specifies the next state.

Note – this can be a little confusing:
• $q$ is the state after popping $\beta$
• $N$ is the left-hand side of the rule just used in reduce
For example...

<table>
<thead>
<tr>
<th>ACTION</th>
<th>sn = shift to state n</th>
<th>rm = reduce using rule number (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>* + 0 1 $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td>s1 s2</td>
<td>3 4</td>
</tr>
<tr>
<td>q1</td>
<td>r4 r4 r4 r4 r4</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>r5 r5 r5 r5 r5</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5 s6 acc</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3 r3 r3 r3 r3</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>s1 s2</td>
<td>7</td>
</tr>
<tr>
<td>q6</td>
<td>s1 s2</td>
<td>8</td>
</tr>
<tr>
<td>q7</td>
<td>r1 r1 r1 r1 r1</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td>r2 r2 r2 r2 r2</td>
<td></td>
</tr>
</tbody>
</table>

GOTO

(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1
The Algorithm, Formally

• Initialize the stack to $q_0$

• Repeat until halting:
  ‣ Consider $\text{ACTION}[q, t]$ for $q$ at the top of stack and $t$ the next token
    • “shift $n$”:
      • Remove $t$ from the input and push $t$ and then $q_n$ to the stack.
    • “reduce $m$”, where rule $(m)$ is $N \rightarrow \beta$:
      • Remove $|\beta|$ pairs from the stack; let $q$ be the state at the top of the stack.
      • Push $N$ and the state $\text{GOTO}[q, N]$ to the stack.
    • “accept”: halt successfully.
  • empty cell: halt with an error.
Using LR Items to Build the Tables

• Typically a state consists of several LR items
• For example, if we identified a string that is reduced to $E$, then we may be in one of the following LR items:

\[ E \rightarrow E \cdot+ B \quad \text{or} \quad E \rightarrow E \cdot^* B \]

• Therefore one state would be:

\[ q = \{ E \rightarrow E \cdot+ B, E \rightarrow E \cdot^* B \} \]

• But if the current state includes $E \rightarrow E \cdot^+ B$, then we must allow $B$ to be derived too — **Closure**!
Construct the Closure

• Proposition: a closure set of LR(0) items has the following property — if the set contains an item of the form

\[ A \rightarrow \alpha \cdot B \beta \]

then it must also contain an item

\[ B \rightarrow \cdot \delta \]

for each rule of the form \( B \rightarrow \delta \) in the grammar.

• Building the closure set for a given item set is recursive, as \( \delta \) may also begin with a variable.
Closure: an example

The closure of the set C is

$$\text{clos}(C) = \{ E \rightarrow E + \bullet B , \\ B \rightarrow \bullet 0 , \\ B \rightarrow \bullet 1 \}$$

• This will become another parser state
Extended Grammar

• **Goal**: simple termination condition
  ‣ Assume that the initial variable only appears in a single rule. This guarantees that the last reduction can be (easily) detected.
  ‣ Any grammar can be (easily) extended to have such structure.
Extended Grammar

• **Goal**: simple termination condition
  ‣ Assume that the initial variable only appears in a single rule. This guarantees that the last reduction can be (easily) detected.
  ‣ Any grammar can be (easily) extended to have such structure.

Example: the grammar

(1) \( E \rightarrow E \ast B \)
(2) \( E \rightarrow E + B \)
(3) \( E \rightarrow B \)
(4) \( B \rightarrow 0 \)
(5) \( B \rightarrow 1 \)
Extended Grammar

• **Goal**: simple termination condition
  ‣ Assume that the initial variable only appears in a single rule. This guarantees that the last reduction can be (easily) detected.
  ‣ Any grammar can be (easily) extended to have such structure.

Example: the grammar

| (1) E → E * B |
| (2) E → E + B |
| (3) E → B   |
| (4) B → 0   |
| (5) B → 1   |

Can be extended into

| (0) S → E |
| (1) E → E * B |
| (2) E → E + B |
| (3) E → B   |
| (4) B → 0   |
| (5) B → 1   |
The Initial State

- To build the ACTION/GOTO table, we go through all possible states during derivation
- Each state represents a (closure) set of LR(0) items
- The initial state $q_0$ is the closure of the initial rule
- In our example the initial rule is $S \rightarrow \bullet E$, and therefore the initial state is

$$q_0 = \text{clos} \{ S \rightarrow \bullet E \} =$$

$$\{ S \rightarrow \bullet E , E \rightarrow \bullet E * B , E \rightarrow \bullet E + B ,$$

$$E \rightarrow \bullet B , B \rightarrow \bullet 0 , B \rightarrow \bullet 1 \}$$

- We build all possible next states by following a single symbol (token or variable)
The Next States

• For each possible terminal or variable $X$, and each possible state (closure set) $q$,
  1. Find all items in the set of $q$ in which the dot is before an $X$. We denote this set by $q|X$
  2. Move the dot ahead of the $X$ in all items in $q|X$
  3. Find the closure of the obtained set: this is the state into which we move from $q$ upon seeing $X$

• Formally, the next set of a set $C$ and next symbol $X$
  $\triangleright \text{step}(C,X) = \{ N \rightarrow \alpha X\cdot\beta \mid N \rightarrow \alpha\cdot X\beta \in C\}$
  $\triangleright \text{nextSet}(C,X) = \text{clos}(\text{step}(C,X))$
The Next States

Recall that in our example

\[ q_0 = \text{clos}(\{ S \rightarrow \bullet E \}) = \]

\[ \{ S \rightarrow \bullet E , E \rightarrow \bullet E \ast B , E \rightarrow \bullet E + B , \]

\[ E \rightarrow \bullet B , B \rightarrow \bullet 0 , B \rightarrow \bullet 1 \} \]

Let us check which states are reachable from it.
The Next States

Recall that in our example

\[ q_0 = \text{clos}(\{ S \rightarrow E \}) = \{ S \rightarrow E, E \rightarrow E \cdot B, E \rightarrow E + B, E \rightarrow B, B \rightarrow 0, B \rightarrow 1 \} \]

Let us check which states are reachable from it.
States reachable from $q_0$ in the example

$q_0 \xrightarrow{0} q_1$
$q_0\{0 = \{B \rightarrow 0 \}$
$q_1 = \text{clos}(\{B \rightarrow 0 \} = \$
\{B \rightarrow 0 \}$
States reachable from $q_0$ in the example

$q_0 \overset{0}{\rightarrow} q_1$
$q_0|0 = \{B \rightarrow \bullet 0\}$
$q_1 = \text{clos}(\{B \rightarrow 0 \bullet\}) = \{B \rightarrow 0 \bullet\}$

$q_0 \overset{1}{\rightarrow} q_2$
$q_0|1 = \{B \rightarrow \bullet 1\}$
$q_2 = \text{clos}(\{B \rightarrow 1 \bullet\}) = \{B \rightarrow 1 \bullet\}$
States reachable from $q_0$ in the example

$q_0 | 0 = \{B \rightarrow \bullet 0\}$

$q_1 = \text{clos}(\{B \rightarrow 0 \bullet\}) = \{B \rightarrow 0 \bullet\}$

$q_0 | 1 = \{B \rightarrow \bullet 1\}$

$q_2 = \text{clos}(\{B \rightarrow 1 \bullet\}) = \{B \rightarrow 1 \bullet\}$

$q_0 | E = \{S \rightarrow \bullet E , E \rightarrow \bullet E * B , E \rightarrow \bullet E + B\}$

$q_3 = \text{clos}(\{S \rightarrow E \bullet , E \rightarrow E \bullet * B , E \rightarrow E \bullet + B\}) = \{S \rightarrow E \bullet , E \rightarrow E \bullet + B , E \rightarrow E \bullet + B\}$
States reachable from $q_0$ in the example

$q_0 \rightarrow 0$

$q_1$

$q_0 \mid 0 = \{B \rightarrow \cdot 0\}$

$q_1 = \text{clos}(\{B \rightarrow 0 \cdot\}) = \{B \rightarrow 0 \cdot\}$

$q_2$

$q_0 \mid 1 = \{B \rightarrow \cdot 1\}$

$q_2 = \text{clos}(\{B \rightarrow 1 \cdot\}) = \{B \rightarrow 1 \cdot\}$

$q_0 \rightarrow 1$

$q_0 \mid E = \{S \rightarrow \cdot E, E \rightarrow \cdot E \cdot B, E \rightarrow \cdot E + B\}$

$q_3 = \text{clos} (\{S \rightarrow E \cdot, E \rightarrow E \cdot * B, E \rightarrow E \cdot + B\})$

$= \{S \rightarrow E \cdot, E \rightarrow E \cdot + B, E \rightarrow E \cdot + B\}$

$q_0 \rightarrow B$

$q_0 \mid B = \{E \rightarrow \cdot B\}$

$q_4 = \text{clos} (\{E \rightarrow B \cdot\}) = \{E \rightarrow B \cdot\}$
From these new states there are more reachable states

- From $q_1$, $q_2$, $q_4$, there are no steps because the dot is at the end of every item in their sets.

- From state $q_3$ we can reach the following two states —

  \[
  q_5 \iff q_3 | ^* = \{ E \rightarrow E \cdot ^* B \}
  \]

  \[
  q_5 = \text{clos} (\{ E \rightarrow E \cdot ^* B \}) = \{ E \rightarrow E \cdot ^* B , B \rightarrow \cdot 0 , B \rightarrow \cdot 1 \}
  \]

  \[
  q_6 \iff q_3 | ^+ = \{ E \rightarrow E \cdot ^+ B \}
  \]

  \[
  q_6 = \text{clos}(\{ E \rightarrow E \cdot ^+ B \}) = \{ E \rightarrow E \cdot ^+ B , B \rightarrow \cdot 0 , B \rightarrow \cdot 1 \}
  \]
Finally

- From $q_5$ we can proceed with $x=0$, or $x=1$, or $x=B$.
- For $x=0$ we reach $q_1$ again and for $x=1$ we reach $q_2$.
- For $x=B$ we get $q_7$:

  $q_7$

  \[
  \text{clos} \{ E \rightarrow E \ast B \bullet \} = \{ E \rightarrow E \ast B \bullet \}
  \]

- Similarly, from $q_6$ with $x=B$ we get $q_8$:

  $q_8$

  \[
  \text{clos} \{ E \rightarrow E + B \bullet \} = \{ E \rightarrow E + B \bullet \}
  \]

- These two states have no further steps. (Why?)
Automaton
Building the Tables

- A row for each state.
- If $q_j$ was obtained at $q_i$ upon seeing $x$, then in row $q_i$ and column $x$ we write $j$.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
</tr>
<tr>
<td>q_0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q_1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>q_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q_4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q_6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* = $+$ = 0 = 1 = $\in$
Building the tables: accept

- Add **accept** in column $ for each state that has $S \rightarrow E\bullet$ as an item.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>q0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>q1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Tables: Shift

- Any number $n$ in the action table becomes shift $n$.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>q0</td>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>q1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td>s6</td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>q6</td>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building the Tables: Reduce

For any state whose set includes the item $A \rightarrow \alpha$, such that $A \rightarrow \alpha$ is production rule (m):

Fill *all columns* of that state in the ACTION table with *reduce m*.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>q1</td>
<td>r4</td>
<td>4</td>
</tr>
<tr>
<td>q2</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td>acc</td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>s1</td>
<td>7</td>
</tr>
<tr>
<td>q6</td>
<td>s1</td>
<td>8</td>
</tr>
<tr>
<td>q7</td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td>r2</td>
<td></td>
</tr>
</tbody>
</table>
Note on LR(0)

When a reduce is possible, we execute it without checking the next token.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s1</td>
<td>3</td>
</tr>
<tr>
<td>q1</td>
<td>r4</td>
<td>4</td>
</tr>
<tr>
<td>q2</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>s1</td>
<td>7</td>
</tr>
<tr>
<td>q6</td>
<td>s1</td>
<td>8</td>
</tr>
<tr>
<td>q7</td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td>r2</td>
<td></td>
</tr>
</tbody>
</table>
### GOTO/ACTION Table

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td>s3</td>
<td>s2</td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r2</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td>s3</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s7</td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>1</td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>6</td>
</tr>
<tr>
<td>q1</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
</tr>
</tbody>
</table>

### Productions

1. $Z \rightarrow E \ \$ 
2. $E \rightarrow T$ 
3. $E \rightarrow E + T$ 
4. $T \rightarrow i$ 
5. $T \rightarrow ( E )$

### Rules

- $(1) \quad Z \rightarrow E \ \$ 
- $(2) \quad E \rightarrow T$ 
- $(3) \quad E \rightarrow E + T$ 
- $(4) \quad T \rightarrow i$ 
- $(5) \quad T \rightarrow ( E )$

### Actions

- $sn = \text{shift to state } n$ 
- $rm = \text{reduce using rule number } (m)$
GOTO/ACTION Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>s5</td>
</tr>
<tr>
<td>+</td>
<td>s7</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>6</td>
</tr>
<tr>
<td>q0</td>
<td>i</td>
</tr>
<tr>
<td>q1</td>
<td>i q5</td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
</tr>
<tr>
<td>q4</td>
<td>r1</td>
</tr>
<tr>
<td>q5</td>
<td>r1</td>
</tr>
<tr>
<td>q6</td>
<td>r1</td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
</tr>
<tr>
<td>q8</td>
<td>s3</td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
</tr>
</tbody>
</table>

Stack | Input | Action
--- | --- | ---
q0 i i $ s5
q0 i q5 + i $ r4
q0 T q6 + i $ r2
q0 E q1 i $ s3
q0 E q1 + q3 i $ s5
q0 E q1 + q3 i q5 + i $ r4
q0 E q1 + q3 T q4 + i $ r3
q0 E q1 + q3 T q4 + i $ r3
q0 E q1 $ s2
q0 E q1 $ q2 r1
q0 Z accept

top is on the right

(1) Z → E $
(2) E → T
(3) E → E + T
(4) T → i
(5) T → ( E )
### GOTO/ACTION Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>6</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>

#### Input

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0 i q5</td>
<td>+ i $</td>
<td>r4</td>
</tr>
<tr>
<td>q0 T q6</td>
<td>+ i $</td>
<td>r2</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>+ i $</td>
<td>s3</td>
</tr>
<tr>
<td>q0 E q1 + q3</td>
<td>i $</td>
<td>s5</td>
</tr>
<tr>
<td>q0 E q1 + q3 i q5</td>
<td>$</td>
<td>r4</td>
</tr>
<tr>
<td>q0 E q1 + q3 T q4</td>
<td>$</td>
<td>r3</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>$</td>
<td>s2</td>
</tr>
<tr>
<td>q0 E q1 $ q2</td>
<td></td>
<td>r1</td>
</tr>
<tr>
<td>q0 Z</td>
<td></td>
<td>accept</td>
</tr>
</tbody>
</table>

(1) $Z \rightarrow E$
(2) $E \rightarrow T$
(3) $E \rightarrow E + T$
(4) $T \rightarrow i$
(5) $T \rightarrow (E)$

Top is on the right.
GOTO/ACTION Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>E</td>
</tr>
<tr>
<td>+</td>
<td>T</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>

Stack | Input | Action
------|-------|-------
q0    | i + i | s5    |
q0 i q5 | + i $ | r4    |
q0 T q6 | + i $ | r2    |
q0 E q1 | + i $ | s3    |
q0 E q1 + q3 | i $ | s5    |
q0 E q1 + q3 i q5 | $ | r4    |
q0 E q1 + q3 T q4 | $ | r3    |
q0 E q1 | $ | s2    |
q0 E q1 $ q2 | |      |
q0 Z    |      | accept
**GOTO/ACTION Table**

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>E</td>
</tr>
<tr>
<td>+</td>
<td>E</td>
</tr>
<tr>
<td>(</td>
<td>T</td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>

**Stack**

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>i + i $</td>
<td>s5</td>
</tr>
<tr>
<td>q0 i q5</td>
<td>+ i $</td>
<td>r4</td>
</tr>
<tr>
<td>q0 T q6</td>
<td>+ i $</td>
<td>r2</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>+ i $</td>
<td>s3</td>
</tr>
<tr>
<td>q0 E q1 + q3</td>
<td>i $</td>
<td>s5</td>
</tr>
<tr>
<td>q0 E q1 + q3 i q5</td>
<td>$</td>
<td>r4</td>
</tr>
<tr>
<td>q0 E q1 + q3 T q4</td>
<td>$</td>
<td>r3</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>$</td>
<td>s2</td>
</tr>
<tr>
<td>q0 E q1 $ q2</td>
<td></td>
<td>r1</td>
</tr>
<tr>
<td>q0 Z</td>
<td></td>
<td>accept</td>
</tr>
</tbody>
</table>

*top is on the right*

**Production Rules**

1. $Z \rightarrow E$
2. $E \rightarrow T$
3. $E \rightarrow E + T$
4. $T \rightarrow i$
5. $T \rightarrow ( E )$
# GOTO/ACTION Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GOTO</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
</table>

Stack | Input | Action |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>i + i $</td>
<td>s5</td>
</tr>
<tr>
<td>q0 i q5</td>
<td>+ i $</td>
<td>r4</td>
</tr>
<tr>
<td>q0 T q6</td>
<td>+ i $</td>
<td>r2</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>+ i $</td>
<td>s3</td>
</tr>
<tr>
<td>q0 E q1 + q3</td>
<td>i $</td>
<td>s5</td>
</tr>
<tr>
<td>q0 E q1 + q3 i q5</td>
<td>$</td>
<td>r4</td>
</tr>
<tr>
<td>q0 E q1 + q3 T q4</td>
<td>$</td>
<td>r3</td>
</tr>
<tr>
<td>q0 E q1</td>
<td>$</td>
<td>s2</td>
</tr>
<tr>
<td>q0 E q1 $ q2</td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>q0 Z</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>
### GOTO/ACTION Table

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>i</td>
<td>1 6</td>
</tr>
<tr>
<td>q0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>q0</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td>s3</td>
<td>s2</td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td>4</td>
</tr>
<tr>
<td>q3</td>
<td>s7</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td>8</td>
</tr>
<tr>
<td>q7</td>
<td>s7</td>
<td>6</td>
</tr>
<tr>
<td>q8</td>
<td>s3</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td>s9</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td></td>
</tr>
</tbody>
</table>

**Stack** | **Input** | **Action**
---|---|---
q0 | i + i $ | s5
q0 i q5 | + i $ | r4
q0 T q6 | + i $ | r2
q0 E q1 | + i $ | s3
q0 E q1 + q3 | i $ | s5
q0 E q1 + q3 i q5 | $ | r4
q0 E q1 + q3 T q4 | $ | r3
q0 E q1 | $ | s2
q0 E q1 $ q2 | | r1
q0 Z | | accept

(1) $Z \rightarrow E$
(2) $E \rightarrow T$
(3) $E \rightarrow E + T$
(4) $T \rightarrow i$
(5) $T \rightarrow (E)$

Top is on the right
GOTO/ACTION Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>E 1</td>
</tr>
<tr>
<td>+</td>
<td>E 1</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>

(1) $Z \rightarrow E$
(2) $E \rightarrow T$
(3) $E \rightarrow E + T$
(4) $T \rightarrow i$
(5) $T \rightarrow (E)$

Stack | Input | Action |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>i + i</td>
<td>$s_5$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>i $q_5$</td>
<td>+ i</td>
</tr>
<tr>
<td>$q_0$</td>
<td>T $q_6$</td>
<td>+ i</td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$</td>
<td>+ i</td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$ $q_3$</td>
<td>i</td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$ $q_3$ i $q_5$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$ $q_3$ T $q_4$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td>E $q_1$ $q_2$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

top is on the right
Are we done?

• Can make a transition diagram for any grammar
• Can make a GOTO table for every grammar
Are we done?

• Can make a transition diagram for any grammar
• Can make a GOTO table for every grammar

• ...but the states are not always clear on what to do

⇒ Cannot make a deterministic ACTION table for every grammar
LR(0) Conflicts

\[
\begin{align*}
Z & \rightarrow E \, $ \\
E & \rightarrow T \\
E & \rightarrow E + T \\
T & \rightarrow i \\
T & \rightarrow (E) \\
T & \rightarrow i[E]
\end{align*}
\]
LR(0) Conflicts

q₀

Z → •E $  
E → •T  
E → •E + T  
T → •i  
T → •( E )  
T → •i[ E ]

Z → E $  
E → T  
E → E + T  
T → i  
T → ( E )  
T → i[ E ]
LR(0) Conflicts

Z → •E $ 
E → •T 
E → •E + T 
T → •i 
T → •( E ) 
T → •i[ E ]

Z → E $ 
E → T 
E → E + T 
T → i 
T → ( E ) 
T → i[ E ]
LR(0) Conflicts

Z → •E $  
E → •T  
E → •E + T  
T → •i  
T → •( E )  
T → •i[ E ]

T → •E $  
E → •T  
E → •E + T  
T → •i  
T → •i[ E ]

q₀

q₅

shift/reduce conflict
View in Action/Goto Table

- shift/reduce conflict...

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>[</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q₁</td>
<td></td>
<td>s₃</td>
<td></td>
<td></td>
<td>s₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>q₄</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td></td>
</tr>
<tr>
<td>q₅</td>
<td>r₄</td>
<td>r₄</td>
<td>r₄</td>
<td>r₄</td>
<td>r₄</td>
<td>r₄</td>
<td>r₄</td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td>s₃</td>
<td></td>
<td>s₉</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₉</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td></td>
</tr>
</tbody>
</table>
LR(0) Conflicts

Z → E $  
E → T  
E → V + T  
T → i  
V → i  
T → ( E )
LR(0) Conflicts

Z → •E $ 
E → •T 
E → •E + T 
T → •i 
V → •i 
T → •(E) 

q₀

Z → E $ 
E → T 
E → V + T 
T → i 

V → i 
T → (E)
LR(0) Conflicts

Z → •E $
E → •T
E → •E + T
T → •i
V → •i
T → •( E )

Z → E $
E → T
E → V + T
T → i
V → i
T → ( E )
LR(0) Conflicts

Z → •E $  
E → •T  
E → •E + T  
T → •i  
V → •i  
T → •( E )

Z → E $  
E → T  
E → V + T  
T → i  
V → i  
T → ( E )
LR(0) Conflicts

\[ Z \rightarrow \bullet E \, $ \]
\[ E \rightarrow \bullet T \]
\[ E \rightarrow \bullet E + T \]
\[ T \rightarrow \bullet i \]
\[ V \rightarrow \bullet i \]
\[ T \rightarrow \bullet ( E ) \]

\[ Z \rightarrow \bullet E \, $ \]
\[ E \rightarrow T \]
\[ E \rightarrow V + T \]
\[ T \rightarrow i \]
\[ V \rightarrow i \]

\[ T \rightarrow i \bullet \]
\[ V \rightarrow i \bullet \]

reduce/reduce conflict
View in Action/Goto Table

- reduce/reduce conflict...

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q₁</td>
<td></td>
<td>s₃</td>
<td></td>
<td></td>
<td>s₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
<td>r₁</td>
</tr>
<tr>
<td>q₃</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>q₄</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
<td>r₃</td>
</tr>
<tr>
<td>q₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
</tr>
<tr>
<td>q₆</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
</tr>
<tr>
<td>q₇</td>
<td>s₅</td>
<td></td>
<td>s₇</td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td>s₃</td>
<td></td>
<td>s₉</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₉</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
</tr>
</tbody>
</table>
View in Action/Goto Table

- reduce/reduce conflict...

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q₁</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>q₃</td>
<td>s5</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>q₄</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>q₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
<td>r₄/r₅</td>
</tr>
<tr>
<td>q₆</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
<td>r₂</td>
</tr>
<tr>
<td>q₇</td>
<td>s5</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td>s3</td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₉</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
<td>r₅</td>
</tr>
</tbody>
</table>

Can there be a shift/shift conflict?
LR(0) vs. \(\varepsilon\)-Rules

- Whenever a nonterminal has an \(\varepsilon\) production, it will be reduced as soon as it is reached in the grammar (without looking at the next token).
- If the variable has another production with a terminal prefix, there is an inherent shift/reduce conflict.

\[
\begin{align*}
  A &\rightarrow \varepsilon \\
  A &\rightarrow a\ A
\end{align*}
\]

\(\times\) Not good

- Both are in the closure of any item of the form \(\{P \rightarrow \alpha\cdot A\beta\}\)

\[
\begin{align*}
  A &\rightarrow \varepsilon \\
  A &\rightarrow A\ a
\end{align*}
\]

\(\checkmark\) This is fine

- No such thing as a shift/goto conflict

\[
\begin{align*}
  A &\rightarrow \cdot \\
  A &\rightarrow \cdot a\ A
\end{align*}
\]

- Reduce item

\[
\begin{align*}
  A &\rightarrow \cdot A\ a
\end{align*}
\]

- No such thing as a shift/goto conflict

\[
\begin{align*}
  A &\rightarrow \cdot a\ A
\end{align*}
\]

- Shift item
Coming Up

Yet some more LR parsing
THEORY OF COMPILATION

LECTURE 04

SYNTAX ANALYSIS
BOTTOM-UP PARSING
You are here

Source text

Lexical Analysis

Syntax Analysis

Parsing

Semantic Analysis

IR Optimization

Code Generation

Executable code
Reminder – Parser Classes

- Top-down (predictive)
- Bottom-up (shift-reduce)
Reminder – LR(0) Parsing

Stack

Input

Automaton

GOTO Table

ACTION Table

Output
(0) $S \rightarrow E$
(1) $E \rightarrow E \ast B$
(2) $E \rightarrow E + B$
(3) $E \rightarrow B$
(4) $B \rightarrow 0$
(5) $B \rightarrow 1$

LR(0) Parsing Algorithm
Reminder – LR(0) Conflicts

Z → E $ 
E → T 
E → E + T 
T → i 
T → ( E ) 
T → i[ E ]
Reminder – LR(0) Conflicts

$q_0$

\[
\begin{align*}
Z & \rightarrow \cdot E \; $ \\
E & \rightarrow \cdot T \\
E & \rightarrow \cdot E + T \\
T & \rightarrow \cdot i \\
T & \rightarrow \cdot ( E ) \\
T & \rightarrow \cdot i [ E ]
\end{align*}
\]

\[
\begin{align*}
Z & \rightarrow E \; $ \\
E & \rightarrow T \\
E & \rightarrow E + T \\
T & \rightarrow i \\
T & \rightarrow ( E ) \\
T & \rightarrow i [ E ]
\end{align*}
\]
Reminder – LR(0) Conflicts

Z → i E $
E → i T
E → i E + T
T → i
T → i ( E )
T → i [ E ]

Z → E $
E → T
E → E + T
T → i
T → i ( E )
T → i [ E ]
Reminder – LR(0) Conflicts

\[
\begin{align*}
Z & \rightarrow \cdot E \ $ \\
E & \rightarrow \cdot T \\
E & \rightarrow \cdot E + T \\
T & \rightarrow \cdot i \\
T & \rightarrow \cdot ( E ) \\
T & \rightarrow \cdot i[ E ] \\
\end{align*}
\]

\[
\begin{align*}
T & \rightarrow \cdot ( E ) \\
T & \rightarrow i \cdot [ E ] \\
T & \rightarrow \cdot ( E ) \\
T & \rightarrow \cdot i[ E ] \\
\end{align*}
\]

shift/reduce conflict
Reminder – LR(0) Conflicts

Z → E $ 
E → T 
E → V + T 
T → i 
V → i 
T → ( E )
Reminder – LR(0) Conflicts

**Grammar Rules:**

- $Z \rightarrow E\,\$ 
- $E \rightarrow T$
- $E \rightarrow E + T$
- $T \rightarrow i$
- $V \rightarrow i$
- $T \rightarrow (E)$

**Highlighted Rule:**

- $V \rightarrow i$
Reminder – LR(0) Conflicts

q₀

Z → E $  
E → T  
E → E + T  
T → i  
V → i  
T → ( E )  

Z → E $  
E → T  
E → V + T  
T → i  
V → i  
T → ( E )
Reminder – LR(0) Conflicts

Z → •E $
E → •T
E → •E + T
T → •i
V → •i
T → •( E )

q₀

T → ...
( → q₅
i
V → ...

Z → E $
E → T
E → V + T
T → i
V → i
T → ( E )

q₅

T → i•
V → i•
Reminder – LR(0) Conflicts

\[ Z \rightarrow \cdot E \$
E \rightarrow \cdot T
E \rightarrow \cdot E + T
T \rightarrow \cdot i
V \rightarrow \cdot i
T \rightarrow \cdot ( E ) \]

\[ T \rightarrow \cdot i \]
\[ V \rightarrow \cdot i \]

reduce/reduce conflict
### Back to Action/Goto Table

- Remember? Reductions ignore the input...

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td>s3</td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SLR Grammars

• A string should only be reduced to a nonterminal N if the look-ahead is a token that can follow N

• A reduce item $N \rightarrow \alpha \cdot$ is applicable only when the look-ahead is in FOLLOW(N)

• Differs from LR(0) only on the original “reduce” rows

• Allows us to sometimes not reduce, instead shift (or do nothing = error)
Rule 1 can only be used with the end of input “$” sign.
### GOTO/ACTION Table

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q1</td>
<td></td>
<td>s3</td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td>s3</td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The tokens that can follow $E$ are `'+'`, `'('`, and `'$'.`

- (1) $Z \rightarrow E \; \$$
- (2) $E \rightarrow T$
- (3) $E \rightarrow E \; T$
- (4) $T \rightarrow i$
- (5) $T \rightarrow ( \; E \; )$
The tokens that can follow E are ‘+’ ‘)’ and ‘$’.

<table>
<thead>
<tr>
<th>States</th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q1</td>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td>r3</td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td>r2</td>
<td></td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td>s3</td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# GOTO/ACTION Table

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>Goto/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>s5</td>
<td>s7</td>
</tr>
<tr>
<td>q1</td>
<td>s3</td>
<td>s2</td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td>r1</td>
</tr>
<tr>
<td>q3</td>
<td>s5</td>
<td>s7</td>
</tr>
<tr>
<td>q4</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>q5</td>
<td>r4</td>
<td>r4</td>
</tr>
<tr>
<td>q6</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>q7</td>
<td>s5</td>
<td>s7</td>
</tr>
<tr>
<td>q8</td>
<td>s3</td>
<td>s9</td>
</tr>
<tr>
<td>q9</td>
<td>r5</td>
<td>r5</td>
</tr>
</tbody>
</table>

- **GOTO**:
  - (1) $Z \rightarrow E \; $*
  - (2) $E \rightarrow T$
  - (3) $E \rightarrow E + T$
  - (4) $T \rightarrow i$
  - (5) $T \rightarrow ( \; E \; )$

- **ACTION**:
  - (1) $E \rightarrow T$
  - (3) $E \rightarrow E + T$

*Same for T*
Now let’s add “$T \rightarrow i \ [ E ]$”
Now let’s add “$T \rightarrow i \{ E \}$”
Now let’s add “T → i [ E ]”

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>+</th>
<th>( )</th>
<th>$</th>
<th>[ ]</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>q₁</td>
<td></td>
<td>s3</td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td>s5</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>q₄</td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₅</td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td>s10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td></td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td>s5</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td>s3</td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₉</td>
<td></td>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₀</td>
<td>s5</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

(1) Z → E $  
(2) E → T  
(3) E → E + T  
(4) T → i  
(5) T → ( E )  
(6) T → i [ E ]
SLR: check next token when reducing
SLR: check next token when reducing

- Simple LR(1), or SLR(1), or SLR.
- Example demonstrates elimination of a shift/reduce conflict.
- Can eliminate reduce/reduce conflicts when conflicting rules’ left-hand sides satisfy:
  \[ \text{FOLLOW}(T) \cap \text{FOLLOW}(V) = \emptyset. \]
- But cannot resolve all conflicts.
Consider this non-LR(0) grammar

(0) $S' \rightarrow S$
(1) $S \rightarrow L = R$
(2) $S \rightarrow R$
(3) $L \rightarrow *\ R$
(4) $L \rightarrow id$
(5) $R \rightarrow L$
Shift/reduce conflict

- $S \rightarrow L \cdot = R$ vs. $R \rightarrow L \cdot$
- FOLLOW(R) contains "="

$S \Rightarrow L = R \Rightarrow * R = R$

$\Rightarrow$ SLR cannot resolve the conflict either
Resolving the Conflict

• In SLR: a reduce item $N \rightarrow \alpha\bullet$ is applicable when the lookahead is in \textsc{follow}(N).
• But there is a whole \textit{sentential form} that we have discovered so far.
• We can ask what the next token may be given all \textit{previous reductions}
• For example, even looking at the FOLLOW of the entire sentential form is more restrictive than looking at the FOLLOW of the last variable.
• In a way, \textsc{follow}(N) merges look-ahead for all possible occurrences of $N$:

  \[
  \text{FOLLOW}(\sigma N) \subseteq \text{FOLLOW}(N)
  \]

• \textit{LR(1)} keeps look-ahead with \textit{each} LR item
LR(1) Item

So far we’ve matched \( \alpha \), expecting to see \( \beta \), followed by the lookahead \( \sigma \)
LR(1) Item

So far we’ve matched $\alpha$, expecting to see $\beta$, followed by the lookahead $\sigma$
LR(1) Item

- Example: the production $L \rightarrow \text{id}$ yields the following LR(1) items

\[
\begin{align*}
&\text{(0) } S' \rightarrow S \\
&\text{(1) } S \rightarrow L \rightarrow R \\
&\text{(2) } S \rightarrow R \\
&\text{(3) } L \rightarrow * R \\
&\text{(4) } L \rightarrow \text{id} \\
&\text{(5) } R \rightarrow L
\end{align*}
\]

- SLR:
  Reduce only when next token is in FOLLOW(L)

- LR(1):
  Refines FOLLOW by using the RHS of the rules

\[
\begin{align*}
&[L \rightarrow \bullet \text{id}] \\
&[L \rightarrow \bullet \text{id}, *] \\
&[L \rightarrow \bullet \text{id}, =] \\
&[L \rightarrow \bullet \text{id}, \text{id}] \\
&[L \rightarrow \bullet \text{id}, \$] \\
&[L \rightarrow \text{id} \bullet, *] \\
&[L \rightarrow \text{id} \bullet, =] \\
&[L \rightarrow \text{id} \bullet, \text{id}] \\
&[L \rightarrow \text{id} \bullet, \$]
\end{align*}
\]
LR(1) Item

- Example: the production $L \rightarrow \text{id}$ yields the following LR(1) items

- SLR: Reduce only when next token is in FOLLOW($L$)
- LR(1): Refines FOLLOW by using the RHS of the rules
Creating the states for LR(1)

• We start with the initial state:

q₀ will be the closure of: (S’ → • S , $)

• Closure for LR(1):

• For every [A → α • Bβ , c] in the state:
  ‣ for every production B → δ and every token b ∈ FIRST(βc)
  ‣ [B → • δ , b] should also be in the state
Closure of $(S' \rightarrow \bullet S, \$$)

- We would like to add rules that start with $S$, but keep track of possible lookahead.

(0) $S' \rightarrow S$
(1) $S \rightarrow L = R$
(2) $S \rightarrow R$
(3) $L \rightarrow * R$
(4) $L \rightarrow id$
(5) $R \rightarrow L$
Closure of \( (S' \rightarrow \bullet S, \$) \)

- We would like to add rules that start with \( S \), but keep track of possible lookahead.
  - \( (S' \rightarrow \bullet S, \$) \)

---

(0) \( S' \rightarrow S \)
(1) \( S \rightarrow L = R \)
(2) \( S \rightarrow R \)
(3) \( L \rightarrow * R \)
(4) \( L \rightarrow \text{id} \)
(5) \( R \rightarrow L \)
Closure of \((S' \rightarrow \bullet S, \$)\)

- We would like to add rules that start with \(S\), but keep track of possible lookahead.
  - \((S' \rightarrow \bullet S, \$)\)
  - \((S \rightarrow \bullet L = R, \$)\)  
    - Rules for \(S\)
  - \((S \rightarrow \bullet R, \$)\)
Closure of \((S' \rightarrow \bullet S, \$)\)

• We would like to add rules that start with \(S\), but keep track of possible lookahead.

\[
\begin{align*}
(S' & \rightarrow \bullet S, \$) \\
(S & \rightarrow \bullet L = R, \$) & \text{ – Rules for } S \\
(S & \rightarrow \bullet R, \$) \\
(L & \rightarrow \bullet * R, =) & \text{ – Rules for } L \\
(L & \rightarrow \bullet id, =)
\end{align*}
\]
Closure of \((S' \rightarrow \bullet S, \$)\)

• We would like to add rules that start with \(S\), but keep track of possible lookahead.

- \((S' \rightarrow \bullet S, \$)\)
- \((S \rightarrow \bullet L = R, \$)\) – Rules for \(S\)
- \((S \rightarrow \bullet R, \$)\)
- \((L \rightarrow \bullet * R, =)\) – Rules for \(L\)
- \((L \rightarrow \bullet \text{id}, =)\)
- \((R \rightarrow \bullet L, \$)\) – Rules for \(R\)
Closure of \((S' \rightarrow \bullet S, \$)\)

- We would like to add rules that start with \(S\), but keep track of possible lookahead.

- \((S' \rightarrow \bullet S, \$)\)
- \((S \rightarrow \bullet L = R, \$)\) – Rules for \(S\)
- \((S \rightarrow \bullet R, \$)\)
- \((L \rightarrow \bullet * R, =)\) – Rules for \(L\)
- \((L \rightarrow \bullet id, =)\)
- \((R \rightarrow \bullet L, \$)\) – Rules for \(R\)
- \((L \rightarrow \bullet * R, \$)\) – More rules for \(L\)
- \((L \rightarrow \bullet id, \$)\)

(0) \(S' \rightarrow S\)
(1) \(S \rightarrow L = R\)
(2) \(S \rightarrow R\)
(3) \(L \rightarrow * R\)
(4) \(L \rightarrow id\)
(5) \(R \rightarrow L\)
The State Machine

0
(S’ → · S, $)
(S → · L = R, $)
(S → · R, $)
(L → · * R, =)
(L → · id, =)
(R → · L, $)
(L → · id, $)
(L → · * R, $)

(0) S’ → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , = )
(L → · id , = )
(R → · L , $)
(L → · id , $)
(L → · * R , $)

*
The State Machine

0
(S’ $)
(S → S , $)
(S → R , $)
(L → R , =)
(L → id , =)
(R → L , $)
(L → id , $)
(L → R , $)

4
(L → id , $)
(R → L , $)
(L → id , $)
(L → R , $)
(L → * R , $)

(0) S’ → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → · S, $)
(S → · L = R, $)
(S → · R, $)
(L → · * R, =)
(L → · id, =)
(R → · L, $)
(L → · id, $)
(L → · * R, $)

4
(L → · R, =)
(R → · L, =)
(L → · * R, =)
(L → · id, =)
(L → · R, $)
(L → · * R, $)
(R → · L, $)
(L → · * R, $)
(L → · id, $)
The State Machine

0
(S' → · S, $)
(S → · L = R, $)
(S → · R, $)
(L → · * R, =)
(L → · id, =)
(R → · L, $)
(L → · id, $)
(L → · * R, $)

4
(L → * · R, =)
(R → · L, =)
(L → · * R, =)
(L → · id, =)
(L → * · R, $)
(R → · L, $)
(L → · * R, $)
(L → · id, $)
The State Machine

0
(S' → · S, $)
(S → · L = R, $)
(S → · R, $)
(L → · * R, =)
(L → · id, =)
(R → · L, $)
(L → · id, $)
(L → · * R, $)

3
(S → R ·, $)

R

4
(L → * · R, =)
(R → · L, =)
(L → · * R, =)
(L → · id, =)
(L → * · R, $)
(L → * · R, $)
(R → · L, $)
(L → · * R, $)
(L → · id, $)

The State Machine

(0) S' → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S’ → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , = )
(L → · id , = )
(R → · L , $ )
(L → · id , $ )
(L → · * R , $ )

3
(S → R · , $)

4
(L → * · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → * · R , $)
(R → · L , $)
(L → · * R , $)
(L → · id , $)

The State Machine

(0) S’ → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

(0) S' → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → · S ,$)
(S → · L = R ,$)
(S → · R ,$)
(L → · * R ,=)
(L → · id ,=)
(R → · L ,$)
(L → · id ,$)
(L → · * R ,$)

1
(S' → S · ,$)
(S → R · ,$)

2
(S → R · ,$)

3
(S → R · ,$)

4
(L → * · R ,=)
(L → · * R ,=)
(L → · id ,=)
(L → * · R ,$)
(R → · L ,=)
(R → · L ,$)
(L → · * R ,$)
(L → · id ,$)

(0) S' → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → S , $)
(S → L = R , $)
(S → R , $)
(L → * R , =)
(L → id , =)
(R → L , $)
(L → id , $)
(L → * R , $)

1
(S → S , $)
(L → * R , =)
(R → L , =)
(L → id , =)
(R → L , $)
(L → * R , $)
(L → id , $)

2
(S → L = R , $)
(L → L , $)
(L → * R , $)
(R → L , $)
(L → * R , $)
(L → id , $)

3
(S → R · , $)
(L → * R , =)
(R → L , =)
(L → * R , =)
(L → id , =)
(R → L , $)
(L → * R , $)
(L → id , $)

4
(L → * R , =)
(R → L , =)
(L → * R , =)
(R → L , $)
(L → * R , $)
(R → L , $)
(L → * R , $)
(L → id , $)
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · id , $)
(L → · * R , $)

1
(S → R · , $)
(S' → S · , $)

2
(S → L · = R , $)
(S → L · , $)
(R → L · , $)
(L → · id , $)
(L → * · R , $)
(R → · L , $)
(L → · id , $)

3
(S → R · , $)
(R → · L , $)
(L → · id , =)

4
(L → * · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → * · R , $)
(R → · L , $)
(L → · * R , $)
(L → · id , $)
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · id , $)
(L → · * R , $)

1
(S' → S · , $)
(S → R · , $)

2
(S → L · = R , $)
(R → L · , $)
(L → · * R , =)
(L → · id , =)
(R → L · , $)
(L → · id , $)
(L → · * R , $)

3
(S → R · , $)

4
(L → · * R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(R → · R , $)
(L → · id , $)
(R → · R , $)
(L → · id , $)
(L → · id , $)

5
(L → id · , $)
(L → id · , =)

(0) S' → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · id , $)
(L → · * R , $)

1
(S' → S · , $)

2
(S → L · = R , $)
(R → L · , $)

3
(S → R · , $)

4
(L → * · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → * · R , $)
(R → · L , $)
(L → · id , $)

5
(L → id · , $)
(L → id · , =)

(0) S' → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · id , $)
(L → · * R , $

1
(S' → S · , $)
(S → R · , $)

2
(S → L · = R , $)
(R → L · , $)

3
(S → R · , $)

4
(L → * · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → * · R , $)
(R → · L , $)

5
(L → id · , $)
(L → id · , =)

6
(L → L · , $)
(L → · * R , $)
(L → · id , $)

7
(L → * R · , =)
(L → * R · , $)
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , = )
(L → · id , = )
(R → · L , $ )
(L → · id , $ )
(L → · * R , $ )

1
(S' → S · , $)
(S → L · = R , $)
(R → L · , $)
(L → · * R , =)
(L → · id , =)
(R → · L , =)
(L → · id , $)
(L → · * R , $)

2
(S → L · = R , $)
(R → L · , $)
(L → · * R , =)
(L → · id , =)
(R → · L , =)
(L → · id , $)
(L → · * R , $)

3
(S → R · , $)
(R → L · , $)
(L → · * R , =)
(L → · id , $)
(L → · * R , $)

4
(L → · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · * R , $)
(L → · id , $)

5
(L → id · , $)
(L → id · , =)
(L → id , $)

6
(L → id · , $)
(L → id · , =)

7
(L → · R · , =)
(L → · R · , $)
The State Machine

0
(S' → · S, $)
(S → · L = R, $)
(S → · R, $)
(L → · * R, =)
(L → · id, =)
(R → · L, $)
(L → · id, $)
(L → · * R, $)

1
(S → S → · S, $)

2
(S → L → · L = R, $)
(R → L → · L, $)
(L → · * R, $)
(L → id · , =)
(L → * · R, $)

3
(S → R → · R, $)

4
(L → · * R, =)
(R → · L, =)
(L → · * R, =)
(L → · id, =)
(L → * · R, $)
(R → · L, $)
(L → · * R, $)
(L → · id, $)

5
(L → id · , $)
(L → id · , =)

6
(S → L → · L = R, $)
(R → L → · L, $)
(L → · * R, $)
(L → id · , =)

7
(L → * R → · R, =)
(L → * R → · R, $)

(0) S’ → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S' → ∙ S, $)
(S → ∙ L = R, $)
(S → ∙ R, $)
(L → ∙ * R, =)
(L → ∙ id, =)
(R → ∙ L, $)
(L → ∙ id, $)
(L → ∙ * R, $)

1
(S → L → ∙ R, =)
(L → ∙ id, =)
(R → ∙ L, $)
(L → ∙ id, $)
(L → ∙ * R, $)

2
(S → L → ∙ R, $)
(R → ∙ L, $)
(L → ∙ id, $)
(L → ∙ * R, $)
(L → * ∙ R, =)

3
(S → R → ∙ R, $)
(R → R → ∙ L, $)

4
(L → * ∙ R, =)
(R → ∙ L, =)
(L → ∙ * R, =)
(L → ∙ id, =)
(L → * ∙ R, $)

5
(L → id → ∙ R, $)
(L → id → ∙ L, =)
(L → id → ∙ R, =)

6
(S → S' → ∙ S, $)

7
(L → ∗ R → ∙ R, =)
(L → ∗ R → ∙ R, $)

8
(R → L → ∙ R, =)
(R → L → ∙ R, $)
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , = )
(L → · id , = )
(R → · L , $ )
(L → · id , $ )
(L → · * R , $ )

1
(S → S · , $)

2
(S → R · , $)

3
(S → R · , $)

4
(L → * · R , = )
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → * · R , $)
(R → · L , $)
(L → · * R , $)
(L → · id , $)

5
(L → id · , $)
(L → id · , =)

6

(L → * R · , =)
(R → L · , $)

7

(R → L · , =)

8
(R → L · , $)

(0) S’ → S
(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L
The State Machine

0
(S’ → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , = )
(L → · id , = )
(R → · L , $ )
(L → · id , $ )
(L → · * R , $ )

1
(S → L · = R , $)
(S → R · , $)

2
(S → L · = R , $)
(R → L · , $)
(L → · * R , =)
(L → · id , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)

3
(S → R · , $)
(L → · * R , $)

4
(L → * · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)

5
(L → id · , $)
(L → id · , =)
(L → · * R , $)
(R → · L , $)

6
(S → L = · R , $)
(L → id · , $)
(R → · L , $)
(L → · id , $)

7
(L → * R · , =)
(L → * R · , $)
(R → L · , =)
(L → * R · , $)

8
(R → L · , $)
The State Machine

0
(S' → · S , $)
(S → · L = R , $)
(S → · R , $)
(L → · * R , =)
(L → · id , =)
(R → · L , $)
(L → · id , $)
(L → · * R , $)

1
(S' → S · , $)
(S → L · = R , $)
(R → · L , $)
(L → · id , $)
(L → · * R , $)

2
(S → L · = R , $)
(R → L · , $)
(L → · * R , $)
(L → · id , $)

3
(S → R · , $)

4
(L → · · R , =)
(R → · L , =)
(L → · * R , =)
(L → · id , =)
(L → · · R , $)
(R → · L , $)
(L → · * R , $)
(L → · id , $)

5
(L → id · , $)
(L → id , $)

6
(S → L = · R , $)
(R → · L , $)
(L → · * R , $)
(L → · id , $)

7
(L → · R · , =)
(L → · R · , $)

8
(R → L · , =)
(R → L · , $)

9
(S → L = R · , $)
The State Machine

1. $(S' \rightarrow S \cdot, \$)$
2. $(S \rightarrow L \cdot = R, \$) \quad (R \rightarrow L \cdot, \$)$
3. $(S \rightarrow R \cdot, \$)$
4. $(L \rightarrow id \cdot, \$) \quad (L \rightarrow id \cdot, =)$
5. $(L \rightarrow id \cdot, \$)$ \quad (L \rightarrow id \cdot, =)$
6. $(S \rightarrow L \equiv R, \$) \quad (R \rightarrow \cdot L, \$) \quad (L \rightarrow \cdot * R, \$) \quad (L \rightarrow \cdot id, \$)$
7. $(L \rightarrow * R \cdot, \$) \quad (L \rightarrow * R \cdot, =)$
8. $(R \rightarrow L \cdot, \$) \quad (R \rightarrow L \cdot, =)$
9. $(S \rightarrow L = R \cdot, \$)$
10. $(L \rightarrow \cdot L, \$)$ \quad (L \rightarrow \cdot * R, \$) \quad (L \rightarrow \cdot id, \$)$
11. $(L \rightarrow id \cdot, \$)$ \quad (R \rightarrow L \cdot, \$)$
12. $(R \rightarrow L \cdot, \$)$
13. $(L \rightarrow * R \cdot, \$)$
Back to the conflict

- Is there a conflict now?

(0) $S' \rightarrow S$
(1) $S \rightarrow L = R$
(2) $S \rightarrow R$
(3) $L \rightarrow * R$
(4) $L \rightarrow \text{id}$
(5) $R \rightarrow L$

$q_2$

$S \rightarrow L \ast = R, \$
$R \rightarrow L \ast, \$

$q_6$

$S \rightarrow L = \ast R, \$
$R \rightarrow \ast L, \$
$L \rightarrow \ast * R, \$
$L \rightarrow \ast \text{id}, \$

Building the Tables

• Similarly to LR(0) and SLR, we start with the automaton.
• Turn each transition in a token column to a shift.
• The variables’ columns form the GOTO section.
• The “acc” is put in the $ column, for any state that contains $(S' \rightarrow S\bullet, \$)$.
• For any state that contains an item of the form $(A \rightarrow \beta\bullet, a)$, where $A \rightarrow \beta$ is rule number $(m)$, use “reduce $m$” for the row of this state and the column of token $a$. 
# Building the Table

## ACTION

<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th>*</th>
<th>=</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s5</td>
<td>s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>s6</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>r2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td>s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>r4</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s11</td>
<td>s10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>r5</td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>s11</td>
<td>s10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>r5</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>r1</td>
<td></td>
</tr>
</tbody>
</table>

## GOTO

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. $(0) S' \rightarrow S$
2. $(1) S \rightarrow L = R$
3. $(2) S \rightarrow R$
4. $(3) L \rightarrow * R$
5. $(4) L \rightarrow id$
6. $(5) R \rightarrow L$
Bottom-up Parsing

- LR(k)
- SLR
- LALR (variant of LR(1))

- All follow the same pushdown-based algorithm
- Differ on type of “LR Items”

\[
\begin{align*}
LR(0) & : N \rightarrow \alpha \cdot \beta \\
SLR(1) & : N \rightarrow \alpha \cdot \beta, \sigma \\
LR(1) & : N \rightarrow \alpha \cdot \beta, \sigma
\end{align*}
\]
Chomsky Hierarchy

- Regular
- Context free
- Context sensitive
- Recursively enumerable

- Finite-state automaton
- Non-deterministic pushdown automaton
- Linear-bounded non-deterministic Turing machine
- Turing machine
Grammar Hierarchy

Non-ambiguous CFG

LR(1)
LALR(1)
SLR(1)
LR(0)
Grammar Hierarchy

Non-ambiguous CFG

LR(1)
LALR(1)
LL(1)
SLR(1)
LR(0)
Building the Parse Tree

• Done at the time of reduce.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$</td>
</tr>
<tr>
<td>q_8</td>
<td>r2</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
(0) & S \rightarrow E \\
(1) & E \rightarrow E \ast B \\
(2) & E \rightarrow E + B \\
(3) & E \rightarrow B \\
(4) & B \rightarrow 0 \\
(5) & B \rightarrow 1 \\
\end{align*}
\]

```java
{ 
    result = new Node("E");
    result.addChild(stack[top-6]);
    result.addChild(stack[top-4]);
    result.addChild(stack[top-2]);
    pop(6);
    next = GOTO[stack[top-1], "E"]; 
    push(result);
    push(next);
}
```
Building the Parse Tree

- Done at the time of **reduce**.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>q₈</td>
<td>r₂</td>
<td>r₂</td>
</tr>
</tbody>
</table>

```java
{ 
    result = new Node("E");
    result.addChild(stack[top-6]);
    result.addChild(stack[top-4]);
    result.addChild(stack[top-2]);
    pop(6);
    next = GOTO[stack[top-1], "E"]; 
    push(result);
    push(next);
}
```
Building the Parse Tree

- Done at the time of **reduce**.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>T</td>
</tr>
<tr>
<td>q8</td>
<td>r2</td>
<td></td>
</tr>
</tbody>
</table>

```java
{ 
    result = new Node("E");
    result.addChild(stack[top-6]);
    result.addChild(stack[top-4]);
    result.addChild(stack[top-2]);
    pop(6);
    next = GOTO[stack[top-1], "E"]; 
    push(result);
    push(next);
}
```
Building the Parse Tree

• Done at the time of `reduce`.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>q_8</td>
<td>r2</td>
<td>r2</td>
</tr>
</tbody>
</table>

```java
{ 
    result = new Node("E");
    result.addChild(stack[top-6]);
    result.addChild(stack[top-4]);
    result.addChild(stack[top-2]);
    pop(6);
    next = GOTO[stack[top-1], "E"]; 
    push(result);
    push(next);
}
```
Building the Parse Tree

• Done at the time of reduce.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>q8</td>
<td>r2</td>
<td>r2</td>
</tr>
</tbody>
</table>

result = new Node("E");
result.addChild(stack[top-6]);
result.addChild(stack[top-4]);
result.addChild(stack[top-2]);
pop(6);
next = GOTO[stack[top-1], "E"];
push(result);
push(next);
Building the Parse Tree

- Done at the time of *reduce*.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>q_8</td>
<td>r2</td>
<td>r2</td>
</tr>
</tbody>
</table>

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

```java
{ 
result = $0 = new Node("E");
result.addChild($1);
result.addChild($2);
result.addChild($3);
pop(6);
next = GOTO[stack[top-1], "E"];
push($0);
push(next);
}
```

```java
result = $0 = new Node("E");
result.addChild($1);
result.addChild($2);
result.addChild($3);
```
Building the **Abstract** Syntax Tree

- Generally — just “skip over” the creation of some internal nodes and you get an AST

```plaintext
E → E + B {  
  $0 = \text{new Node}("+");  
  $0$.addChild($1);  
  $0$.addChild($3);  
}
E → B {  
  $0 = $1;  
}
E → E + B {  
  $0 = \text{new Node}("#");  
  $0$.value = 0;  
}
```

### Production Rules

- (0) $S \rightarrow E$
- (1) $E \rightarrow E \ast B$
- (2) $E \rightarrow E + B$
- (3) $E \rightarrow B$
- (4) $B \rightarrow 0$
- (5) $B \rightarrow 1$
Building the Abstract Syntax Tree

• Generally — just “skip over” the creation of some internal nodes and you get an AST

```
E → E + B {  
    $0 = new Node("+");  
    $0.addChild($1);  
    $0.addChild($3);  
}
E → B {  
    $0 = $1;  
}
E → E + B {  
    $0 = new Node("#");  
    $0.value = 0;  
}
E → E * B
E → E + B
E → B
B → 0
B → 1
```

(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1
Building the **Abstract** Syntax Tree

- Generally — just “skip over” the creation of some internal nodes and you get an AST

```
E → E + B {  
  $0 = \texttt{new Node} ("+");  
  $0$.addChild($1);  
  $0$.addChild($3);  
}
E → B {  
  $0 = $1;  
}
E → E + B {  
  $0 = \texttt{new Node} ("#");  
  $0$.value = 0;  
}
```

(0) S → E  
(1) E → E * B  
(2) E → E + B  
(3) E → B  
(4) B → 0  
(5) B → 1
Summary

✓ Bottom up derivation
✓ LR(k) can decide on a reduce after seeing the entire right side of the rule plus k look-ahead tokens.
  ✓ particularly LR(0) – must reduce without lookahead.
✓ Using a table and a stack to derive.
✓ Definition of LR Items and the automaton.
✓ Creating the table from the automaton.
✓ LR(0), SLR, LR(1) – different kinds of LR items, same basic algorithm
  • LALR: in the tutorial.
Summary

✓ Bottom up derivation
✓ LR(k) can decide on a reduce after seeing the entire right side of the rule plus k look-ahead tokens.
  ✓ particularly LR(0) – must reduce without lookahead.
✓ Using a table and a stack to derive.
✓ Definition of LR Items and the automaton.
✓ Creating the table from the automaton.
✓ LR(0), SLR, LR(1) – different kinds of LR items, same basic algorithm
  • LALR: in the tutorial.
 Semantic Analysis 