Building LR Tables

How to construct the ACTION and GOTO tables?

• Define “items”
• Define “viable prefix”
• Define the “closure function”
  \[
  \text{Set-of-items} \rightarrow \text{Set-of-items}
  \]
• Define the GOTO function
• Work with a set of sets of items
  \[
  \text{A collection of sets of items}
  \]
  \[
  \text{CC = Cannonical Collection of LR items}
  \]
• Describe how to construct CC
• Given all this, describe how to construct the tables
LR(0) Items

**Given:** A grammar, G
Items look like productions
... augmented with a dot in the righthand side.

**Grammar:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( E \rightarrow E \ + \ T )</td>
</tr>
<tr>
<td>2.</td>
<td>( E \rightarrow T )</td>
</tr>
<tr>
<td>3.</td>
<td>( T \rightarrow T \ * \ F )</td>
</tr>
<tr>
<td>4.</td>
<td>( T \rightarrow F )</td>
</tr>
<tr>
<td>5.</td>
<td>( F \rightarrow ( \ E \ ) )</td>
</tr>
<tr>
<td>6.</td>
<td>( F \rightarrow id )</td>
</tr>
</tbody>
</table>

**The Items:**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( E \rightarrow \cdot E \ + \ T )</td>
<td>( T \rightarrow \cdot F )</td>
<td></td>
</tr>
<tr>
<td>( E \rightarrow E \cdot + T )</td>
<td>( T \rightarrow F \cdot )</td>
<td></td>
</tr>
<tr>
<td>( E \rightarrow E \ + \cdot T )</td>
<td>( F \rightarrow \cdot ( E ) )</td>
<td></td>
</tr>
<tr>
<td>( E \rightarrow E + T \cdot )</td>
<td>( F \rightarrow ( \cdot E ) )</td>
<td></td>
</tr>
<tr>
<td>( T \rightarrow \cdot T \ * \ F )</td>
<td>( F \rightarrow ( E \cdot ) )</td>
<td></td>
</tr>
<tr>
<td>( T \rightarrow T \ * \ F )</td>
<td>( F \rightarrow ( E ) \cdot )</td>
<td></td>
</tr>
<tr>
<td>( T \rightarrow T \ * \ F )</td>
<td>( F \rightarrow id )</td>
<td></td>
</tr>
<tr>
<td>( T \rightarrow T \ * \ F )</td>
<td>( F \rightarrow id \cdot )</td>
<td></td>
</tr>
</tbody>
</table>

**Special Case:**

**Rule:**
\( A \rightarrow \varepsilon \)

**Yields one item:**
\( A \rightarrow \cdot \)
LR(1) Items

*Just like before, except...*
- Look-ahead symbol
- Terminal symbol from grammar

**Grammar:**

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T \ast F$
4. $T \rightarrow F$
5. $F \rightarrow ( E )$
6. $F \rightarrow \text{id}$

**Examples:**

- $E \rightarrow \cdot E + T , )$
- $E \rightarrow \cdot E + T , \$\$
- $E \rightarrow E \cdot + T , )$
- $E \rightarrow E \cdot + T , \$
- ...
Intuition behind LR(1) Items

F → ( • E ) , )
We were hoping / expecting to see an F next, followed by a )
and we have already seen a (.
We are on the path to finding an F, followed by a ).
Using rule 5, one way to find an F is to find ( E ) next.
So now we are looking for E ), followed by a ).

F → ( E ) • , )
We were looking for an F, followed by a )
and we have found ( E )
If a ) comes next then the parse is going great!
... Now reduce, using rule F → ( E )
Intuition behind LR(1) Items

F → • ( E ), )

It would be legal at this point in the parse
to see an F, followed by a ).

Using rule 5, one way to find an F is to find ( E ) next.
So, among other possibilities, we are looking for ( E ), followed by a ).
If a ( comes next, then let’s scan it and keep going,
looking for E ), followed by a ).
If we get E ) later, then we will be able to reduce it to F
... but we may get something different (although perfectly legal).

E → • T, )

It would be legal at this point in the parse
to see an E, followed by a ).

Using rule 2, one way to find an E is to find T next.
So, among other possibilities, we are looking for a T followed by a ).
And how can we find a T followed by a )?

T → • T * F, )
T → • F, )
Syntax Analysis - Part 3

**Step 1**

Augment the grammar by adding...
- A new start symbol, $S'$
- A new rule $S' \rightarrow S$

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow ( E )$
6. $F \rightarrow id$

0. $S' \rightarrow E$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow ( E )$
6. $F \rightarrow id$

Our goal is to find an $S'$, followed by $\$.$

$S' \rightarrow \cdot E, \$ $

Whenever we are about to reduce using rule 0...
Accept! Parse is finished!
Let’s say we have this item:
\[ E \rightarrow \cdot T, ) \]

What are the ways to find a \( T \)?
\[ T \rightarrow F \]
\[ T \rightarrow T * F \]

We are looking for a \( T \), followed by a \( ) \), so we’ll need to add these items:
\[ T \rightarrow \cdot F, ) \]
\[ T \rightarrow \cdot T * F, ) \]

We can find a \( T \) followed by a \( ) \) if we find an \( F \) following by a \( ) \).

How can we find that?
\[ F \rightarrow \cdot ( E ), ) \]
\[ F \rightarrow \cdot id, ) \]

We can also find a \( T \) followed by a \( ) \) if we find an \( T * F \) following by a \( ) \).

To find that, we need to first find another \( T \), but followed by \( * \).
\[ T \rightarrow \cdot F, * \]
\[ T \rightarrow \cdot T * F, * \]

So we should also look for a \( F \) followed by a \( * \).
\[ F \rightarrow \cdot ( E ), * \]
\[ F \rightarrow \cdot id, * \]
The CLOSURE Function

**Given:**
I = a set of items

**Output:**
CLOSURE(I) = a new set of items

**Algorithm:**
result = {}
add all items in I to result
repeat
  for every item $A \rightarrow \beta \cdot C \delta, a$ in result do
    for each rule $C \rightarrow \gamma$ in the grammar do
      for each $b$ in $\text{FIRST}(\delta a)$ do
        add $C \rightarrow \cdot \gamma, b$ to result
      endFor
    endFor
  endFor
until we can’t add anything more to result
CLOSURE Function Example

**Example:** Let $I_1 = \{ E \rightarrow E \cdot + T, \)
\T \rightarrow T \cdot * F, \)
F \rightarrow \text{id} \cdot, \)
F \rightarrow ( E ) \cdot, \)
\}

**Compute:** $\text{CLOSURE} ( I_1 ) = \{ 0. S' \rightarrow E \)
1. E \rightarrow E + T \)
2. E \rightarrow T \)
3. T \rightarrow T * F \)
4. T \rightarrow F \)
5. F \rightarrow ( E ) \)
6. F \rightarrow \text{id} \}$
CLOSURE Function Example

Example: Let \( I_1 = \{ \begin{align*}
E & \rightarrow E \cdot + T, \\
T & \rightarrow T \cdot * F, \\
F & \rightarrow id \cdot, \\
F & \rightarrow (E) \cdot, 
\end{align*} \}
\)

Compute: \( \text{CLOSURE} (I_1) = \{ \begin{align*}
E & \rightarrow E \cdot + T, \\
T & \rightarrow T \cdot * F, \\
F & \rightarrow id \cdot, \\
F & \rightarrow (E) \cdot, 
\end{align*} \}
\)

Start by adding all items in \( I_1 \):

0. \( S' \rightarrow E \)
1. \( E \rightarrow E + T \)
2. \( E \rightarrow T \)
3. \( T \rightarrow T * F \)
4. \( T \rightarrow F \)
5. \( F \rightarrow (E) \)
6. \( F \rightarrow id \)
CLOSURE Function Example

Example: Let \( I_1 = \{ \begin{align*}
E &\rightarrow E \cdot + T, \\
T &\rightarrow T \cdot * F, \\
F &\rightarrow \text{id} \cdot, \\
F &\rightarrow (E) \cdot, 
\end{align*} \} \)

Compute: \( \text{CLOSURE} (I_1) = \{ \begin{align*}
E &\rightarrow E \cdot + T, \\
T &\rightarrow T \cdot * F, \\
F &\rightarrow \text{id} \cdot, \\
F &\rightarrow (E) \cdot, 
\end{align*} \} \)

Start by adding all items in \( I_1 \)...

Is the dot in front of a non-terminal?
**CLOSURE Function Example**

**Example:** Let $I_1 = \{ \begin{align*}
E & \rightarrow E \cdot + T, \\
T & \rightarrow T \cdot * F, \\
F & \rightarrow \text{id} \cdot, \\
F & \rightarrow (E) \cdot, 
\end{align*} \}$

**Compute:** CLOSURE ($I_1$) = \{
\begin{align*}
E & \rightarrow E \cdot + T, \\
T & \rightarrow T \cdot * F, \\
F & \rightarrow \text{id} \cdot, \\
F & \rightarrow (E) \cdot, 
\end{align*} \}

Is the dot in front of a non-terminal? No... no more items are added.

\[
\begin{align*}
0. & \quad S' \rightarrow E \\
1. & \quad E \rightarrow E + T \\
2. & \quad E \rightarrow T \\
3. & \quad T \rightarrow T * F \\
4. & \quad T \rightarrow F \\
5. & \quad F \rightarrow (E) \\
6. & \quad F \rightarrow \text{id}
\end{align*}
\]
CLOSURE Function Example

*Example:* Let \( I_2 = \{ T \rightarrow \cdot F, \) 
\( T \rightarrow \cdot T \ast F, \) \}

*Compute:* CLOSURE ( \( I_2 \) ) = {
Start by adding...
CLOSURE Function Example

Example: Let $I_2 = \{ \begin{array}{l} T \rightarrow \cdot F, \\ T \rightarrow \cdot T \ast F, \end{array} \}$

Compute: CLOSURE ($I_2$) = 

Start by adding all items in $I_2$...

1. $T \rightarrow \cdot F, 
2. T \rightarrow \cdot T \ast F,$

Look at (1) first...

0. $S' \rightarrow E$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T \ast F$
4. $T \rightarrow F$
5. $F \rightarrow ( E )$
6. $F \rightarrow id$
**CLOSURE Function Example**

Example: Let $I_2 = \{ \begin{align*} T &\rightarrow \cdot F, \\
T &\rightarrow \cdot T \ast F, \end{align*} \}$

Compute: CLOSURE ($I_2$) = { 
Start by adding all items in $I$...

(1) $T \rightarrow \cdot F, \\
(2) T \rightarrow \cdot T \ast F,$

Look at (1) first. Look at each $F$ rule. For every $b$ in FIRST ($\epsilon$) = {}...

0. $S' \rightarrow E$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T \ast F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$
**CLOSURE Function Example**

**Example:** Let \( I_2 = \{ \ T \rightarrow \cdot F, \ T \rightarrow \cdot T \cdot F, \} \)

**Compute:** \( \text{CLOSURE}(I_2) = \{ \)

Start by adding all items in \( I \... \)

(1) \( T \rightarrow \cdot F, \)

(2) \( T \rightarrow \cdot T \cdot F, \)

Look at (1) first. Look at each \( F \) rule. For every \( b \) in \( \text{FIRST}(\varepsilon) = \{ \} \)...

(3) \( F \rightarrow \cdot (E), \)

(4) \( F \rightarrow \cdot \text{id}, \)

Look at (2) next...
CLOSURE Function Example

Example: Let $I_2 = \{ T \rightarrow \cdot F, \\ T \rightarrow \cdot T \ast F, \} \}

Compute: CLOSURE ( $I_2$ ) = {
Start by adding all items in $I$...

(1) $T \rightarrow \cdot F, \\ (2) T \rightarrow \cdot T \ast F, \\$

Look at (1) first. Look at each $F$ rule. For every $b$ in FIRST ($\epsilon$) = {()}...

(3) $F \rightarrow \cdot (E)$, \\ (4) $F \rightarrow \cdot \text{id}$,

Look at (2) next. Look at each $T$ rule. For every $b$ in FIRST ($\ast F$) = {\ast}...

(5) $T \rightarrow \cdot F, \ast$ \\ (6) $T \rightarrow \cdot T \ast F, \ast$

Look at (3) and (4) next...
**CLOSURE Function Example**

**Example:**  Let $I_2 = \{ \text{T } \rightarrow \cdot \text{F}, \, )$

$\text{T } \rightarrow \cdot \text{T } \ast \text{F}, \, ) \} \}$

**Compute:**  CLOSURE ( $I_2$ ) = {

Start by adding all items in $I$...

1. $\text{T } \rightarrow \cdot \text{F}, \, )$
2. $\text{T } \rightarrow \cdot \text{T } \ast \text{F}, \, )$

Look at (1) first.  Look at each F rule.  For every $b$ in FIRST (ε) = {}...

3. $\text{F } \rightarrow \cdot ( \text{E } ), \, )$
4. $\text{F } \rightarrow \cdot \text{id }, \, )$

Look at (2) next.  Look at each T rule.  For every $b$ in FIRST (*F) = { * }...

5. $\text{T } \rightarrow \cdot \text{F}, \, *$
6. $\text{T } \rightarrow \cdot \text{T } \ast \text{F}, \, *$

Look at (3) and (4) next.  The dot is not in front of a non-terminal.

Look at (5) next...
CLOSURE Function Example

Example: Let \( I_2 = \{ \ T \rightarrow \cdot F, \ \ T \rightarrow \cdot T \cdot F, \ \} \) 

Compute: \( \text{CLOSURE} (I_2) = \{ \)

Start by adding all items in \( I \)...

(1) \( T \rightarrow \cdot F, \) 
(2) \( T \rightarrow \cdot T \cdot F, \) 

Look at (1) first. Look at each \( F \) rule. For every \( b \) in FIRST (\( \varepsilon \)) = \{ \} ...

(3) \( F \rightarrow \cdot (E), \) 
(4) \( F \rightarrow \cdot \text{id}, \) 

Look at (2) next. Look at each \( T \) rule. For every \( b \) in FIRST (\( *F \)) = \{ \} ...

(5) \( T \rightarrow \cdot F, \cdot \) 
(6) \( T \rightarrow \cdot T \cdot F, \cdot \) 

Look at (3) and (4) next. The dot is not in front of a non-terminal. 

Look at (5) next. Look at each \( F \) rule. For every \( b \) in FIRST (\( \varepsilon \cdot \)) = \{ \} ...

(7) \( F \rightarrow \cdot (E), \cdot \) 
(8) \( F \rightarrow \cdot \text{id}, \cdot \) 

Look at (6) next...
CLOSURE Function Example

Example: Let $I_2 = \{ \ T \rightarrow \cdot F, \ \ T \rightarrow \cdot T \ast F, \ \} \}

Compute: CLOSURE ( $I_2$ ) = {

Start by adding all items in I...

(1) $T \rightarrow \cdot F, \$
(2) $T \rightarrow \cdot T \ast F, \$

Look at (1) first. Look at each $F$ rule. For every $b$ in FIRST ($\varepsilon$)) = {() ...}

(3) $F \rightarrow \cdot (E),$
(4) $F \rightarrow \cdot id,$

Look at (2) next. Look at each $T$ rule. For every $b$ in FIRST ($\ast F$) = {$\ast$} ...

(5) $T \rightarrow \cdot F, \ast$
(6) $T \rightarrow \cdot T \ast F, \ast$

Look at (3) and (4) next. The dot is not in front of a non-terminal.
Look at (5) next. Look at each $F$ rule. For every $b$ in FIRST ($\varepsilon \ast$) = {$\ast$} ...

(7) $F \rightarrow \cdot (E), \ast$
(8) $F \rightarrow \cdot id, \ast$

Look at (6) next. Look at each $T$ rule. For every $b$ in FIRST ($\ast F \ast$) = {$\ast$} ...
We already added (5) and (6)

Look at (7) and (8) next...
CLOSURE Function Example

Example: Let $I_2 = \{ \; T \rightarrow \cdot F, \; \) \\
\; T \rightarrow \cdot T \cdot F, \; \) \} \}

Compute: CLOSURE ( $I_2$ ) = \\
Start by adding all items in $I$...

(1) $T \rightarrow \cdot F, \; )$
(2) $T \rightarrow \cdot T \cdot F, \; )$
Look at (1) first. Look at each $F$ rule. For every $b$ in FIRST ($\varepsilon$) = $\{\}$... 
(3) $F \rightarrow \cdot ( \; E \; ), \; )$
(4) $F \rightarrow \cdot \; id, \; )$
Look at (2) next. Look at each $T$ rule. For every $b$ in FIRST ($*F$) = $\{*\}$...
(5) $T \rightarrow \cdot F, \; *$
(6) $T \rightarrow \cdot T \cdot F, \; *$
Look at (3) and (4) next. The dot is not in front of a non-terminal.
Look at (5) next. Look at each $F$ rule. For every $b$ in FIRST ($\varepsilon*$) = $\{*\}$...
(7) $F \rightarrow \cdot ( \; E \; ), \; *$
(8) $F \rightarrow \cdot \; id, \; *$
Look at (6) next. Look at each $T$ rule. For every $b$ in FIRST ($*F*$) = $\{*\}$...
We already added (5) and (6)
Look at (7) and (8) next. The dot is not in front of a non-terminal.

CLOSURE Function Example

**Example:** Let $I_3 = \{ E \rightarrow \cdot E + T, ) \}$

**Compute:** $\text{CLOSURE} ( I_3 ) = \{ \$

E \rightarrow \cdot E + T, )$

Look at $E$ rules. For every $b$ in $\text{FIRST} ( +T ) = \{ + \}$...

$E \rightarrow \cdot E + T, +$

$E \rightarrow \cdot T, +$

Look at $E$ rules. For every $b$ in $\text{FIRST} ( +T+ )$... (Nothing new added)

Look at $T$ rules. For every $b$ in $\text{FIRST} ( \varepsilon+ ) = \{ + \}$

$T \rightarrow \cdot T * F, +$

$T \rightarrow \cdot F, +$

Look at $T$ rules. For every $b$ in $\text{FIRST} ( *F+ ) = \{ * \}$

$T \rightarrow \cdot T * F, *$

$T \rightarrow \cdot F, *$

Look at $F$ rules. For every $b$ in $\text{FIRST} ( \varepsilon+ ) = \{ + \}$

$F \rightarrow \cdot ( E ), +$

$F \rightarrow \cdot \text{id}, +$

Look at $F$ rules. For every $b$ in $\text{FIRST} ( \varepsilon* ) = \{ * \}$

$F \rightarrow \cdot ( E ), *$

$F \rightarrow \cdot \text{id}, *$

$\}$
Let $I$ be a set of items...
Let $X$ be a grammar symbol (terminal or non-terminal)...

function $\text{GOTO}(I,X)$ returns a set of items
result = {}
look at all items in $I$...
if $A \rightarrow \alpha \cdot X \delta, a$ is in $I$
then add $A \rightarrow \alpha X \cdot \delta, a$ to result
result = $\text{CLOSURE}(\text{result})$

In other words, move the dot past the $X$ in any items where it is in front of an $X$

...and take the CLOSURE of whatever items you get
The GOTO Function

Let I be a set of items...
Let X be a grammar symbol (terminal or non-terminal)...

function GOTO(I, X) returns a set of items
    result = {}
    look at all items in I...
    if A → α • X δ, a is in I
        then add A → α X • δ, a to result
    result = CLOSURE(result)

Intuition:
- I is a set of items indicating where we are so far, after seeing some prefix γ of the input.
- I describes what we might legally see next.
- Assume we get an X next.
- Now we have seen some prefix γX of the input.
- GOTO(I, X) tells what we could legally see after that.
- GOTO(I, X) is the set of all items that are “valid” for prefix γX.

In other words, move the dot past the X in any items where it is in front of an X
...and take the CLOSURE of whatever items you get
**GOTO Function Example**

**Example:** Let $I_4 = \{ \ E \rightarrow T \cdot , \ $ $T \rightarrow T \cdot * F , \ ) \ }$

**Compute:** GOTO ( $I_4, *$ ) = \{ 

Is the $\cdot$ in front of $*$ in any of the items?

$T \rightarrow T \cdot * F , \ )$

Now take the closure...

$F \rightarrow \cdot ( E ) , \ )$

$F \rightarrow \cdot id , \ )$

\}

**Intuition:**

We found a $T$ and then we found a $*$. What are we looking for next?

$T \rightarrow T \cdot * F , \ )$

**Means:** We are now looking for an $F$ followed by $)$

$F \rightarrow \cdot ( E ) , \ )$

**Means:** We could find that by finding ( $E$ ) followed by $)$

$F \rightarrow \cdot id , \ )$

**Means:** We could find that by finding ( $E$ ) followed by $)$

---

© Harry H. Porter, 2005
**GOTO Function Example**

**Example:** Let $I_5 = \{ \ E \rightarrow \cdot T, \ )$

\[
T \rightarrow \cdot T \ast F, \ )
\]

**Compute:** $GOTO ( I_5, T) = \{$

Is the $\cdot$ in front of $T$ in any of the items?

$E \rightarrow T \cdot, )$

$T \rightarrow T \cdot * F, )$

Now take the closure...

Is the $\cdot$ in from of any non-terminal? Nothing more added...

\}

**Intuition:**

We were looking for a $T$. Then we found it. What are we looking for next?

$E \rightarrow T \cdot, )$

Means: We are now looking for $

T \rightarrow T \cdot * F, )$

Means: We are now looking for $\ast F$ followed by $

0. S' \rightarrow E$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T \ast F$
4. $T \rightarrow F$
5. $F \rightarrow ( E )$
6. $F \rightarrow id$
Constructing the Canonical Collection

Each $CC_i$ will be a set of items. We will build up a collection of these.

$CC = \{ \text{The Canonical Collection of LR(1) items} \}
= \text{a set of sets of items}
= \{ CC_0, CC_1, CC_2, CC_3, \ldots, CC_N \}$
Constructing the Canonical Collection

Each $CC_i$ will be a set of items.
We will build up a collection of these.

$CC = \text{"The Canonical Collection of LR(1) items"}$
$= \text{a set of sets of items}$
$= \{ CC_0, CC_1, CC_2, CC_3, \ldots CC_N \}$

Algorithm to construct $CC$, the Canonical Collection of LR(1) Items:

1. let $CC_0 = \text{CLOSURE}\left( \{ S' \rightarrow \cdot S, \, \$ \} \right)$
2. add $CC_0$ to $CC$
3. repeat
   1. let $CC_i$ be some set of items already in $CC$
   2. for each $X$ (that follows a $\cdot$ in some item in $CC_i$)...
      1. compute $CC_k = \text{GOTO}(CC_i, X)$
      2. if $CC_k$ is not already in $CC$ then
         1. add it
         2. set $\text{MOVE}(CC_i, X) = CC_k$
      endIf
   endFor
4. until nothing more can be added to $CC$
Example:

\[
CC_0 = \text{CLOSURE( } \{ S' \rightarrow \cdot E, $ \} ) \\
= \{ \begin{align*}
S' & \rightarrow \cdot E, $ \\
E & \rightarrow \cdot E + T, $ \\
E & \rightarrow \cdot E + T, + \\
E & \rightarrow \cdot T, $ \\
E & \rightarrow \cdot T, + \\
T & \rightarrow \cdot T * F, $ \\
T & \rightarrow \cdot T * F, + \\
T & \rightarrow \cdot T * F, * \\
T & \rightarrow \cdot F, $ \\
T & \rightarrow \cdot F, + \\
T & \rightarrow \cdot F, * \\
F & \rightarrow \cdot ( E ), $ \\
F & \rightarrow \cdot ( E ), + \\
F & \rightarrow \cdot ( E ), * \\
F & \rightarrow \cdot \text{id}, $ \\
F & \rightarrow \cdot \text{id}, + \\
F & \rightarrow \cdot \text{id}, * 
\end{align*} \}
\]
Example:

\[
CC_0 = \text{CLOSURE}( \{ S' \rightarrow \cdot E, \$ \} )
= \{ S' \rightarrow \cdot E, \$
    \quad E \rightarrow \cdot E + T, \$
    \quad E \rightarrow \cdot E + T, +
    \quad E \rightarrow \cdot T, \$
    \quad E \rightarrow \cdot T, +
    \quad T \rightarrow \cdot T + F, \$
    \quad T \rightarrow \cdot T + F, +
    \quad T \rightarrow \cdot T + F, *
    \quad T \rightarrow \cdot F, \$
    \quad T \rightarrow \cdot F, +
    \quad T \rightarrow \cdot F, *
    \quad F \rightarrow \cdot ( E ), \$
    \quad F \rightarrow \cdot ( E ), +
    \quad F \rightarrow \cdot ( E ), *
    \quad F \rightarrow \cdot \text{id}, \$
    \quad F \rightarrow \cdot \text{id}, +
    \quad F \rightarrow \cdot \text{id}, *
\}
\]

The \( \cdot \) is before \( E, T, F, \text{id}, \) and \( ( \ldots \) ...

Next, we’ll compute...

\[
\begin{align*}
\text{GOTO (} CC_0, E \text{)} & \Rightarrow CC_1 \\
\text{GOTO (} CC_0, T \text{)} & \Rightarrow CC_2 \\
\text{GOTO (} CC_0, F \text{)} & \Rightarrow CC_3 \\
\text{GOTO (} CC_0, \text{id} \text{)} & \Rightarrow CC_5 \\
\text{GOTO (} CC_0, ( \text{)} \text{)} & \Rightarrow CC_4
\end{align*}
\]
GOTO \((CC_0, E) = CC_1\)

Advance • past \(E\) in the items containing \(•E\)

\[
CC_1 = \{
S' \rightarrow E \cdot, \$
E \rightarrow E \cdot + T, \$
E \rightarrow E \cdot + T, +
\}
\]

And take the closure... (Nothing more added.)

**Intuition:**

We will reduce by \(S' \rightarrow E\) if the next symbol is \$. Otherwise, we will look for a + next.

The • is in front of +. We’ll come back to \(CC_1\) later.
GOTO \((CC_0, T) = CC_2\)

Advance • past \(T\)

\[CC_2 = \{\]
\[\text{E } \rightarrow \text{T} \cdot, \; \$
\[\text{E } \rightarrow \text{T} \cdot, \; +
\[\text{T } \rightarrow \text{T} \cdot \; ^* \; \text{F}, \; $
\[\text{T } \rightarrow \text{T} \cdot \; ^* \; \text{F}, \; +
\[\text{T } \rightarrow \text{T} \cdot \; ^* \; \text{F}, \; ^*
\]

And take the closure...

\[\}

**Intuition:**

We will reduce by \(T \rightarrow F\) if the next symbol is $ or +. Otherwise, we will look for \(^*\).

The • is in front of \(^*\). We’ll come back to \(CC_2\) later.
GOTO \( (CC_0, F) = CC_3 \)

Advance • past \( F \)

\[ CC_3 = \{
   \begin{align*}
   T & \rightarrow F \cdot, \$ \\
   T & \rightarrow F \cdot, + \\
   T & \rightarrow F \cdot, * \\
   \end{align*}
\]

And take the closure...

}\]

**Intuition:**

We will reduce by \( T \rightarrow F \) if the next symbol is \( $, +, \) or \( * \).

The • is not in front of any symbol; no further “GOTO”s.
GOTO \((CC_0, \text{id}) = CC_5\)

Advance • past \text{id}

\[CC_5 = \{
F \rightarrow \text{id}\cdot, \$
F \rightarrow \text{id}\cdot, +
F \rightarrow \text{id}\cdot, *
\}

And take the closure...

**Intuition:**

We will reduce after seeing an \text{id}, if the next symbol is +, *, or $.

The • is not in front of any symbol; no further “GOTO”s.
Syntax Analysis - Part 3

Advance • past (  

\[ \text{GOTO (CC}_0, () = CC_4 \]

\[ \begin{align*} 
\text{CC}_4 &= \{ F \rightarrow (\cdot E ), \$, \\
&\quad F \rightarrow (\cdot E ), + \\
&\quad F \rightarrow (\cdot E ), * \}
\end{align*} \]

And take the closure...

\[ \begin{align*} 
E &\rightarrow \cdot E + T, ) \\
E &\rightarrow \cdot E + T, + \\
E &\rightarrow \cdot T, ) \\
E &\rightarrow \cdot T, + \\
T &\rightarrow \cdot T \ast F, ) \\
T &\rightarrow \cdot T \ast F, + \\
T &\rightarrow \cdot T \ast F, * \\
T &\rightarrow \cdot F, ) \\
T &\rightarrow \cdot F, + \\
T &\rightarrow \cdot F, * \\
F &\rightarrow \cdot (\ E ), ) \\
F &\rightarrow \cdot (\ E ), + \\
F &\rightarrow \cdot (\ E ), * \\
F &\rightarrow \cdot \text{id}, ) \\
F &\rightarrow \cdot \text{id}, + \\
F &\rightarrow \cdot \text{id}, * \\
\end{align*} \]

The • is before \( E, T, F, (\), and \text{id}...

Next, we’ll compute...

\[ \begin{align*} 
\text{GOTO (CC}_4, E) &\Rightarrow \text{CC}_8 \\
\text{GOTO (CC}_4, T) &\Rightarrow \text{CC}_9 \\
\text{GOTO (CC}_4, F) &\Rightarrow \text{CC}_{10} \\
\text{GOTO (CC}_4, ( ) &\Rightarrow \text{CC}_{11} \\
\text{GOTO (CC}_4, \text{id}) &\Rightarrow \text{CC}_{12} \\
\end{align*} \]
Syntax Analysis - Part 3

**GOTO** (CC₁, + ) = CC₆

CC₁ = {
    S' → E • , $
    E → E • + T , $
    E → E • + T , +
}

Advance • past + in the items containing • +

CC₆ = {
    E → E + • T , $
    E → E + • T , +

And take the closure...
    T → • T * F , $
    T → • T * F , +
    T → • T * F , *
    T → • F , $
    T → • F , +
    T → • F , *
    F → • ( E ) , $
    F → • ( E ) , +
    F → • ( E ) , *
    F → • id , $
    F → • id , +
    F → • id , *
}

© Harry H. Porter, 2005
GOTO \((CC_2, \ast) = CC_7\)

\[\begin{align*}
CC_2 &= \{ \\
    &E \rightarrow T \ast, \$ \\
    &E \rightarrow T \ast, + \\
    &T \rightarrow T \ast F, \$ \\
    &T \rightarrow T \ast F, + \\
    &T \rightarrow T \ast F, \ast
\} \\
\text{Advance } \ast \text{ past } \ast \\
CC_7 &= \{ \\
    &T \rightarrow T \ast \ast F, \$ \\
    &T \rightarrow T \ast \ast F, + \\
    &T \rightarrow T \ast \ast F, \ast
\} \\
\text{And take the closure...} \\
F &\rightarrow \ast (E), \$ \\
F &\rightarrow \ast (E), + \\
F &\rightarrow \ast (E), \ast \\
F &\rightarrow \ast \text{id}, \$ \\
F &\rightarrow \ast \text{id}, + \\
F &\rightarrow \ast \text{id}, \ast
\} \\
\textbf{Intuition:} \\
\text{We have found } T \ast. \\
\text{Next, look for a } F \text{ followed by } \$, +, \text{ or } \ast.
Do this one next
Syntax Analysis - Part 3

GOTO \((CC_4, E) = CC_8\)

\[
CC_8 = \{
F \rightarrow (E\cdot), $ \\
F \rightarrow (E\cdot), + \\
F \rightarrow (E\cdot), * \\
E \rightarrow E\cdot + T, ) \\
E \rightarrow E\cdot + T, + \\
\}
\]

And take the closure...

0. \(S' \rightarrow E\)
1. \(E \rightarrow E + T\)
2. \(E \rightarrow T\)
3. \(T \rightarrow T * F\)
4. \(T \rightarrow F\)
5. \(F \rightarrow (E)\)
6. \(F \rightarrow id\)
GOTO (CC₄₃, T) = CC₉

CC₉ = {
    E → T •, )
    E → T •, +
    T → T • F, )
    T → T • F, +
    T → T • F, *
}
And take the closure...

0. S' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → ( E )
6. F → id
Syntax Analysis - Part 3

GOTO \((CC_4, F) = CC_{10}\)

\[\begin{align*}
CC_{10} &= \{ \\
&\quad T \rightarrow F \cdot, ) \\
&\quad T \rightarrow F \cdot, + \\
&\quad T \rightarrow F \cdot, * \\
\&\quad \text{And take the closure...} \\
&\quad \}
\end{align*}\]

0. \(S' \rightarrow E\)
1. \(E \rightarrow E + T\)
2. \(E \rightarrow T\)
3. \(T \rightarrow T \ast F\)
4. \(T \rightarrow F\)
5. \(F \rightarrow (E)\)
6. \(F \rightarrow id\)
GOTO (CC₄, id) = CC₁₂

CC₁₂ = {
    F → id •, )
    F → id •, +
    F → id •, *
}

And take the closure...
CC_{11} = \{
F \rightarrow ( \cdot E ),
F \rightarrow ( \cdot E ), +
F \rightarrow ( \cdot E ), *
\}

And take the closure...
E \rightarrow \cdot E + T,
E \rightarrow \cdot E + T, +
E \rightarrow \cdot T,
E \rightarrow \cdot T, +
T \rightarrow \cdot T * F,
T \rightarrow \cdot T * F, +
T \rightarrow \cdot T * F, *
T \rightarrow \cdot F,
T \rightarrow \cdot F, +
T \rightarrow \cdot F, *
F \rightarrow \cdot ( E ),
F \rightarrow \cdot ( E ), +
F \rightarrow \cdot ( E ), *
F \rightarrow \cdot id,
F \rightarrow \cdot id, +
F \rightarrow \cdot id, *
\}

GOTO (CC_{4}, \cdot) = CC_{11}

CC_{11} is similar to, but not quite the same as CC_{0}

The \cdot is before E, T, F, id, and ( ... 

Next, we’ll compute...
GOTO (CC_{11}, E) \Rightarrow CC_{18}
GOTO (CC_{11}, T) \Rightarrow CC_{9}
GOTO (CC_{11}, F) \Rightarrow CC_{10}
GOTO (CC_{11}, id) \Rightarrow CC_{12}
GOTO (CC_{11}, \cdot) \Rightarrow CC_{11}
GOTO (CC\textsubscript{11}, E) = CC\textsubscript{18}

CC\textsubscript{18} = {
    F \rightarrow ( E \cdot ), 
    F \rightarrow ( E \cdot ), +
    F \rightarrow ( E \cdot ), *
    E \rightarrow E \cdot + T, )
    E \rightarrow E \cdot + T, +
}

And take the closure...

0. S' \rightarrow E
1. E \rightarrow E + T
2. E \rightarrow T
3. T \rightarrow T \ast F
4. T \rightarrow F
5. F \rightarrow ( E )
6. F \rightarrow id
GOTO \((CC_{11}, T) = CC_9\)

\[ CC_9 = \{
  E \rightarrow T \cdot, \)
  E \rightarrow T \cdot, +
  T \rightarrow T \cdot * F, \)
  T \rightarrow T \cdot * F, +
  T \rightarrow T \cdot * F, *
\]

And take the closure...

\[ \}
\]

We have seen \(CC_9\) before!
GOTO (CC_{11}, F) = CC_{10}

CC_{10} = \{
    T \rightarrow F \cdot, 
    T \rightarrow F \cdot, +
    T \rightarrow T \ast \cdot, *
\}

And take the closure...

We have seen CC_{10} before!
0. S' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → ( E )
6. F → id
Viable Prefixes

Consider a right-sentential form

\[ S \Rightarrow_{RM} X A f f f \Rightarrow_{RM} X X B C D E f f f \Rightarrow_{RM} \ldots \]

Rule:

\[ A \rightarrow BCDE \]

A viable prefix is a prefix of a right sentential form that does not extend to the right end of the handle.

\[ X X B C D E f f f \]

A Viable prefix
Viable Prefixes

Consider a right-sentential form

\[ S \Rightarrow_{RM}^{\ast} XXAfff \Rightarrow_{RM} XXBCDEfff \Rightarrow_{RM} \ldots \]

Rule:

A \rightarrow BCDE

A viable prefix is a prefix of a right sentential form that does not extend to the right end of the handle.

\[ XXBCDEfff \]

A Viable prefix

Given a viable prefix, we can always add terminals to get a right-sentential form!

Why?

Assume that XXBC is a viable prefix that we’ve shifted onto the stack.
Assume that we have some more terminals dddeeefff in the input.
If this string is legal, there must be rules that allow

\[ D \Rightarrow^{\ast} ddd \text{ and } E \Rightarrow^{\ast} eee \]

\[ S \Rightarrow_{RM}^{\ast} XXBCDEfff \Rightarrow_{RM} XXBCDeefff \Rightarrow_{RM} XXBCdddeeefff \]

As long as we have a viable prefix, just keep shifting!
The Main Idea of LR Parsing

As long as what is on the stack is a viable prefix...

• The unseen terminals might be what is required to make STACK || REMAINING-INPUT into a right-sentential form.
• We are on course to finding a rightmost derivation.

The key ideas of LR parsing:

Construct a DFA to recognize viable prefixes!

Every path in the DFA (from start to any final or non-final state) describes a viable prefix.

Each state is a set of items.
If the DFA has an edge from the current state labeled with a terminal
And the edge label = the lookahead symbol
Do a shift: Add this terminal to the viable prefix.
When the dot is at the end of one of the items in a state...
If the next symbol = the lookahead symbol...
Do a reduction. A → XYZ
Example

Consider this viable prefix. Trace through the DFA.

( T * ( ( E ) ) Ends up in state 21, a final state.

If next token is +, *, or ) then reduce by F → ( E )

The stack shows our path through the DFA.
“How we got to state 21”
Example
Consider this viable prefix. Trace throught the DFA.
(T * ( (E)) Ends up in state 21, a final state.
If next token is +, *, or ) then reduce by $F \rightarrow (E)$.

0. $S' \rightarrow E$
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

Back up along our path... to where we were before we saw (E). Then take the F edge to get to state 10.
Example
Consider this viable prefix. Trace through the DFA.

\(( T \ast ( ( E ) \) )\) Ends up in state 21, a final state.
If next token is +, *, or ) then reduce by \( F \rightarrow (E) \)

\(( T \ast ( F) \) Ends up in state 10, a final state.
If next token is +, *, or ) then reduce by \( T \rightarrow F \)

Back up along our path... to where we were before we saw \((E)\).
Then take the \( F \) edge to get to state 10.
**Example**

Consider this viable prefix. Trace through the DFA.

( \( T \ast ( ( E ) \) ) Ends up in state 21, a final state.
  If next token is +, *, or ) then reduce by \( F \rightarrow ( E ) \)

( \( T \ast ( F \) ) Ends up in state 10, a final state.
  If next token is +, *, or ) then reduce by \( T \rightarrow F \)

0. \( S' \rightarrow E \)
1. \( E \rightarrow E + T \)
2. \( E \rightarrow T \)
3. \( T \rightarrow T \ast F \)
4. \( T \rightarrow F \)
5. \( F \rightarrow ( E ) \)
6. \( F \rightarrow id \)

Back up along our path...
To where we were before we saw \( F \).
Then take the \( T \) edge to get to state 9.
Example

Consider this viable prefix. Trace through the DFA.

( T * ( ( E ) Ends up in state 21, a final state.
  If next token is +, *, or ) then reduce by F → (E )

( T * ( F Ends up in state 10, a final state.
  If next token is +, *, or ) then reduce by T → F

( T * ( T Ends up in state 9, a final state.
  If next token is +, or ) then reduce by E → T

Assume next token is *... Need to shift

Back up along our path... to where we were before we saw F. Then take the T edge to get to state 9.
Example

Consider this viable prefix. Trace through the DFA.

( T * ( ( E ) Ends up in state 21, a final state.
  If next token is +, *, or ) then reduce by F → ( E )
( T * ( F Ends up in state 10, a final state.
  If next token is +, *, or ) then reduce by T → F
( T * ( T Ends up in state 9, a final state.
  If next token is +, or ) then reduce by E → T
Assume next token is *... Need to shift and goto state 17

There is an edge from state 9 labeled *. Take the * edge to get to state 17.

Now we’ve got this viable prefix:
( T * ( T *
Other Examples

Here are some viable prefixes.
Trace through the DFA for each of these!

( E + T * F
  Goes to state 20, a final state.
  If next token is $, +, or * then reduce by T→T*F
E + ( ( T * id
  Goes to state 12, a final state.
  If next token is +, *, or ) then reduce by F→id
( E + ( E + ( E + ( E + ( T * ( T * ( T * ( T * ( T *
  Goes to state 17, not a final state.
  We must get F, (, or id next
( E + ( T
  Goes to state 9
  If next token is ( or + then reduce by E→T
  Else okay to see *

0. S' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → ( E )
6. F → id
Algorithm to Construct ACTION and GOTO Tables

**Input:** Grammar G, augmented with \( S' \rightarrow S \)

**Output:** ACTION and GOTO Tables

Construct “Canonical Collection of LR(1) items” along with MOVE function.

\[ \text{CCC} = \{ \text{CC}_0, \text{CC}_1, \text{CC}_2, \text{CC}_3, ..., \text{CC}_N \} \]

*There will be N states, one per set of items \{ 0, 1, 2, 3,... N \}*

for each \( \text{CC}_i \) do

  for each item in \( \text{CC}_i \) do

    if the item has the form \( A \rightarrow \beta \cdot c \gamma, a \)
      and \( \text{MOVE}(\text{CC}_i, c) = \text{CC}_j \) then
      set \( \text{ACTION}[i,c] \) to “Shift j”
    
    elseIf the item has the form \( A \rightarrow \beta \cdot, a \) then
      set \( \text{ACTION}[i,a] \) to “Reduce \( A \rightarrow \beta \)”
    
    elseIf the item has the form \( S' \rightarrow S \cdot, \$$ \) then
      set \( \text{ACTION}[i,\$$] \) to “Accept”

  endFor

  for each nonterminal \( A \) do

    if \( \text{MOVE}(\text{CC}_i, A) = \text{CC}_j \) then
      set \( \text{GOTO}[i,A] \) to \( j \)

  endIf

endFor

endFor
The SLR Table Construction Algorithm

With SLR, we do not have the lookahead symbol.

**LR(1) items:**
- \( F \rightarrow ( \cdot E ) \),
- \( F \rightarrow ( \cdot E ) \), +
- \( F \rightarrow ( \cdot E ) \), *

**LR(0) items:**
- \( F \rightarrow ( \cdot E ) \)

Some information is lost.
Some states in \( \mathbb{C} \) collapse into one state.
There are fewer states in \( \mathbb{C} \)
\( \Rightarrow \) Fewer rows in the resulting tables.
CC₄ = \{  
  F \rightarrow ( \cdot E ), $ 
  F \rightarrow ( \cdot E ), + 
  F \rightarrow ( \cdot E ), * 
  E \rightarrow \cdot E + T, ) 
  E \rightarrow \cdot E + T, + 
  E \rightarrow \cdot T, ) 
  E \rightarrow \cdot T, + 
  T \rightarrow \cdot T * F, ) 
  T \rightarrow \cdot T * F, + 
  T \rightarrow \cdot T * F, * 
  T \rightarrow \cdot F, ) 
  T \rightarrow \cdot F, + 
  T \rightarrow \cdot F, * 
  F \rightarrow \cdot ( E ), ) 
  F \rightarrow \cdot ( E ), + 
  F \rightarrow \cdot ( E ), * 
  F \rightarrow \cdot \text{id}, ) 
  F \rightarrow \cdot \text{id}, + 
  F \rightarrow \cdot \text{id}, * 
\} 

CC₁₁ = \{  
  F \rightarrow ( \cdot E ), ) 
  F \rightarrow ( \cdot E ), + 
  F \rightarrow ( \cdot E ), * 
  E \rightarrow \cdot E + T, ) 
  E \rightarrow \cdot E + T, + 
  E \rightarrow \cdot T, ) 
  E \rightarrow \cdot T, + 
  T \rightarrow \cdot T * F, ) 
  T \rightarrow \cdot T * F, + 
  T \rightarrow \cdot T * F, * 
  T \rightarrow \cdot F, ) 
  T \rightarrow \cdot F, + 
  T \rightarrow \cdot F, * 
  F \rightarrow \cdot ( E ), ) 
  F \rightarrow \cdot ( E ), + 
  F \rightarrow \cdot ( E ), * 
  F \rightarrow \cdot \text{id}, ) 
  F \rightarrow \cdot \text{id}, + 
  F \rightarrow \cdot \text{id}, * 
\} 

With SLR, these combine into one state

\{  
  F \rightarrow ( \cdot E ) 
  E \rightarrow \cdot E + T 
  E \rightarrow \cdot T 
  T \rightarrow \cdot T * F 
  T \rightarrow \cdot F 
  F \rightarrow \cdot ( E ) 
  F \rightarrow \cdot \text{id} 
\}
The SLR Table Construction Algorithm

The CLOSURE function is basically the same, but simpler.
The GOTO function is basically the same, but simpler.
The Construction of the Canonical Collection is the same.
The Construction of the ACTION and GOTO tables is a little different.

...\[\text{\underline{elseIf}}\] the item has the form \text{A} \rightarrow \beta \cdot, a \text{ then}\nset ACTION[i, a] to “Reduce A \rightarrow \beta”
...

...\[\text{\underline{elseIf}}\] the item has the form \text{A} \rightarrow \beta \cdot \text{then}\nfor all b in FOLLOW(A) do\nset ACTION[i, b] to “Reduce A \rightarrow \beta”
endFor
...

Sometimes SLR may try to put two actions in one table entry
...while the LR(1) tables would have more states, more rows, and no conflicts.
**SLR: The CLOSURE Function**

**Given:**

I = a set of LR(0) items

**Output:**

CLOSURE(I) = a new set of items

**Algorithm:**

result = {}
add all items in I to result
repeat
    for every item $A \rightarrow \beta \cdot C \delta$ in result do
        for each rule $C \rightarrow \gamma$ in the grammar do
            add $C \rightarrow \cdot \gamma$ to result
        endFor
    endFor
until we can’t add anything more to result
SLR: The GOTO Function

Let $I$ be a set of items...
Let $X$ be a grammar symbol (terminal or non-terminal)...

```
function GOTO(I,X) returns a set of items
    result = {}
    look at all items in I...
    if $A \rightarrow \alpha \cdot X \delta$ is in I
        then add $A \rightarrow \alpha X \cdot \delta$ to result
    result = CLOSURE(result)
```

- In other words, move the dot past the $X$ in any items where it is in front of an $X$
- ...and take the CLOSURE of whatever items you get
SLR: Constructing the Canonical Collection

Each \( CC_i \) will be a set of items.
We will build up a collection of these.

\[
CC = \text{"The Canonical Collection of LR(0) items"}
= \text{a set of sets of items}
= \{ CC_0, CC_1, CC_2, CC_3, \ldots, CC_N \}
\]

Algorithm to construct \( CC \), the Canonical Collection of LR(1) Items:

1. let \( CC_0 = \text{CLOSURE} (\{ S' \rightarrow \cdot S, \$ \}) \)
2. add \( CC_0 \) to \( CC \)
3. repeat
   1. let \( CC_i \) be some set of items already in \( CC \)
   2. for each \( X \) (that follows a \( \cdot \) in some item in \( CC_i \))...
      1. compute \( CC_k = \text{GOTO}(CC_i, X) \)
      2. if \( CC_k \) is not already in \( CC \) then
         1. add it
         2. set \( \text{MOVE}(CC_i, X) = CC_k \)
      3. endIf
   3. endFor
4. until nothing more can be added to \( CC \)
SLR: Algorithm to Construct ACTION and GOTO Tables

**Input:** Grammar G, augmented with $S' \rightarrow S$

**Output:** ACTION and GOTO Tables

Construct “Canonical Collection of LR(0) items” along with MOVE function.

$$CC = \{ CC_0, CC_1, CC_2, CC_3, \ldots, CC_N \}$$

for each $CC_i$ do

  for each item in $CC_i$ do

    if the item has the form $A \rightarrow \beta \cdot c \gamma$

      and $\text{MOVE}(CC_i, c) = CC_j$ then

        set $\text{ACTION}[i, c]$ to “Shift j”

    elseif the item has the form $A \rightarrow \beta \cdot \gamma$ then

      for all $b$ in $\text{FOLLOW}(A)$ do

        set $\text{ACTION}[i, b]$ to “Reduce $A \rightarrow \beta$”

      endFor

    elseif the item has the form $S' \rightarrow S \cdot \gamma$ then

      set $\text{ACTION}[i, \$]$ to “Accept”

  endFor

  for each nonterminal $A$ do

    if $\text{MOVE}(CC_i, A) = CC_j$ then

      set $\text{GOTO}[i, A]$ to $j$

    endIf

  endFor

endFor
YAPP

Yet Another PCAT Parser

An SLR Parser Generator

**INPUT:**
- A Grammar
- A String to Parse

**ACTION:**
- Build the parsing tables using the SLR algorithm
- Parse the string

**YAPP is written in PCAT!**

```
cs.pdx.edu/~harry/compilers/yapp
```

≈ 2100 lines of PCAT code.
Can be compiled by your compiler!!!

**Example Input:**
- A Grammar for PCAT (109 rules)
- String = the YAPP program itself
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

\[ \text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \]

\[ \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t; \]
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

\[
\text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \quad \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t;
\]

**Synthesized Attributes:**
The attributes are computed bottom-up in the parse tree.
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

\[
\text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \quad \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t;
\]

**Synthesized Attributes:**

The attributes are computed bottom-up in the parse tree.
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

\[ \text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \quad \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t; \]

_Synthesized Attributes:_

The attributes are computed bottom-up in the parse tree.
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

$$\text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \quad \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t;$$

**Synthesized Attributes:**
The attributes are computed bottom-up in the parse tree.

```
Expr  t = 123
  +
Expr  t = 100
  +  --
Term  t = 23
  --
```

Put the attributes on the stack, along with grammar symbols.
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

Expr_0 → Expr_1 + Term    Expr_0.t = Expr_1.t + Term.t;

**Synthesized Attributes:**
The attributes are computed bottom-up in the parse tree.

To reduce by Expr → Expr + Term
- Perform the attribute computation
- Pop the stack
- Push the new non-terminal with its attribute.
Attributes in a Shift-Reduce Parser

An attribute can be associated with each grammar symbol.

\[ \text{Expr}_0 \rightarrow \text{Expr}_1 + \text{Term} \quad \text{Expr}_0.t = \text{Expr}_1.t + \text{Term}.t; \]

**Synthesized Attributes:**
The attributes are computed bottom-up in the parse tree.

To reduce by \( \text{Expr} \rightarrow \text{Expr} + \text{Term} \):
- Perform the attribute computation
- Pop the stack
- Push the new non-terminal with its attribute.
**YACC**

**Yet Another Compiler Compiler**
Unix tool to create an LALR parser.
Works with “Lex” tool: Calls `yylex()` to get next token.

```
grammar.y
```

```
lex.l
```

```
YACC Tool
```

```
lex.yy.c
```

```
y.tab.c
```

```
“C” compiler
```

```
a.out
```

```
Source
```

```
Output
```
An Example YACC Grammar

```c
#include <stdio.h>
#define YYSTYPE double
%
%token NUMBER
%

S : S E '/n' { printf("%g\n", $2); } | S '/n' | ;
E : E '+' T { $$ = $1 + $3; } | T { $$ = $1; } ;
T : T '*' F { $$ = $1 * $3; } | F { $$ = $1; } ;
F : NUMBER { $$ = $1; } | '(\ E ')' { $$ = $2; } ;
%
#include lex.yy.c
%
```

### An attribute is associated with each symbol
This tells what type the attribute has.

### The token types (from lexer)

### The grammar rules:
```
S  →  S E /n
    →  S /n
    →  ε
E  →  E + T
    →  T
T  →  T * F
    →  F
F  →  NUMBER
    →  ( E )
```

### Epsilon
```
S  →  S E '/n
    →  S '/n
    →  ε
E  →  E '+' T
    →  T
T  →  T '*' F
    →  F
F  →  NUMBER
    →  '(' E ')'```

### End of rule

### This material copied as is to yy.tab.c
An Example YACC Grammar

%{
#include <stdio.h>
define YYSTYPE double
%
%token NUMBER
%
S : S E `/n' { printf("%g\n", $2); }
  | S `/n'
  |
E : E `+' T { $$ = $1 + $3; }
  | T { $$ = $1; }
  ;
T : T `*' F { $$ = $1 * $3; }
  | F { $$ = $1; }
  ;
F : NUMBER { $$ = $1; }
  | `( E `)\' { $$ = $2; }
  ;
%
#include lex.yy.c
%

Actions are executed when reductions are done!
The attribute of each symbol is referred to by position:

E → E + T

$$ $1 $2 $3

The attributes (such as $$ and $1) are replaced by the appropriate code
How the $ Notation in YACC Works

\[
\begin{align*}
\text{Expr}_0 & \rightarrow \text{Expr}_1 + \text{Term} \\
\text{Expr}_0.t & = \text{Expr}_1.t + \text{Term}.t; \\
\$$ & = $1 + $3;
\end{align*}
\]