Syntax Analysis, VII

*The Canonical LR(1) Table Construction*

Comp 412

---

Copyright 2017, Keith D. Cooper & Linda Torczon, all rights reserved.

Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.
LR(1) Items

An LR(1) item is a pair $[P, \delta]$, where

- $P$ is a production $A \rightarrow \beta$ with a $\bullet$ at some position in the RHS
- $\delta$ is a single symbol lookahead ($\text{symbol} \equiv \text{word or EOF}$)

The $\bullet$ in an item indicates the position of the top of the stack

- $[A \rightarrow \bullet, \beta, \gamma, a]$ means that the input seen so far is consistent with the use of $A \rightarrow \beta$ immediately after the symbol on top of the stack. We call an item like this a **possibility**.

- $[A \rightarrow \beta, \bullet, \gamma, a]$ means that the input seen so far is consistent with the use of $A \rightarrow \beta$ at this point in the parse, and that the parser has already recognized $\beta$ (that is, $\beta$ is on top of the stack). We call an item like this a **partially complete** item.

- $[A \rightarrow \beta, \gamma, \bullet, a]$ means that the parser has seen $\beta, \gamma$, and that a lookahead symbol of $a$ is consistent with reducing to $A$. This item is **complete**.

The intermediate representation of the LR(1) table construction algorithm.

**LR(k)** parsers rely on items with a lookahead of $\leq k$ symbols. That leads to LR(k) items, with correspondingly longer $\delta$. 

COMP 412, Fall 2017
**Review**

**LR(1) Items**

The production $A \rightarrow \beta$, where $\beta = B_1 B_2 B_3$ with lookahead $a$, can give rise to 4 items

$[A \rightarrow \bullet B_1 B_2 B_3, a], [A \rightarrow B_1 \cdot B_2 B_3, a], [A \rightarrow B_1 B_2 \bullet B_3, a], \& [A \rightarrow B_1 B_2 B_3 \cdot, a]$

The set of LR(1) items for a grammar is *finite*.

**What’s the point of all these lookahead symbols?**

- Carry them along to help choose the correct reduction
- Lookaheads are bookkeeping, unless item has $\bullet$ at right end
  - Has no direct use in $[A \rightarrow \beta \bullet, a]$
  - In $[A \rightarrow \beta \bullet, a]$, a lookahead of $a$ implies a reduction by $A \rightarrow \beta$
  - For $\{ [A \rightarrow \beta \bullet, a], [B \rightarrow \gamma \cdot \delta, b] \}$, $a \Rightarrow reduce$ to $A$; $\text{FIRST}(\delta) \Rightarrow shift$

$\Rightarrow$ Limited right context is enough to pick the actions

---

COMP 412, Fall 2017
LR(1) Table Construction

High-level overview

1 Build the Canonical Collection of Sets of LR(1) Items, $I$
   a Begin in an appropriate state, $s_0$
      ♦ $[S' \rightarrow S, \text{EOF}]$, along with any equivalent items
      ♦ Derive equivalent items as $\text{closure}(s_0)$
   b Repeatedly compute, for each $s_k$ and each $X$, $\text{goto}(s_k, X)$
      ♦ If the set is not already in the collection, add it
      ♦ Record all the transitions created by $\text{goto}( )$
     This eventually reaches a fixed point

2 Fill in the table from the Canonical Collection of Sets of LR(1) items

The sets in the canonical collection form the states of the Control DFA.
The construction traces the DFA’s transitions
# LR(1) Table Construction

## High-level overview

1. Build the Canonical Collection of Sets of LR(1) Items, \( I \)
   a. Begin in an appropriate state, \( s_0 \)
      - \([S' \rightarrow \cdot S, \text{EOF}]]\), along with any equivalent items
      - Derive equivalent items as closure( \( s_0 \) )
   b. Repeatedly compute, for each \( s_k \) and each \( X \), goto( \( s_k, X \) )
      - If the set is not already in the collection, add it
      - Record all the transitions created by goto( )
      This eventually reaches a fixed point

2. Fill in the table from the Canonical Collection of Sets of LR(1) items

---

**Let’s build the tables for the left-recursive SheepNoise grammar**  
*(S’ is Goal)*

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Goal</td>
<td>→ SheepNoise</td>
</tr>
<tr>
<td>1</td>
<td>SheepNoise</td>
<td>→ SheepNoise baa</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>baa</td>
</tr>
</tbody>
</table>

 COMP 412, Fall 2017 4
Computing Closures

*Closure*(s) adds all the *possibilities* for the items already in s

- Any item \([A \rightarrow \beta \bullet B \delta, a]\) where \(B \in NT\) implies \([B \rightarrow \bullet \tau, x]\) for each production that has \(B\) on the *lhs*, and each \(x \in \text{FIRST}(\delta a)\)
- Since \(\beta B \delta\) is valid, any way to derive \(\beta B \delta\) is valid, too

The Algorithm

```
*Closure* (s)
while (s is still changing)
  \(\forall\) items \([A \rightarrow \beta \bullet B \delta, a] \in s\)
    lookahead \(\leftarrow \text{FIRST}(\delta a)\) // \(\delta\) might be \(\varepsilon\)
  \(\forall\) productions \(B \rightarrow \tau \in P\)
  \(\forall\) \(b \in\) lookahead
    if \([B \rightarrow \bullet \tau, b] \notin s\)
      then \(s \leftarrow s \cup \{ [B \rightarrow \bullet \tau, b] \}\)
```

- Classic fixed-point method
- Halts because \(s \subseteq I\), the set of all items (finite)
- Worklist version is faster
- *Closure* “fills out” a state \(s\)

Generate new lookaheads. See note on p. 128
Example From SheepNoise

Initial step builds the item \([\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}]\) and takes its Closure( )

**Closure( \([\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}]\) )**

<table>
<thead>
<tr>
<th>Item</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}])</td>
<td>Original item</td>
</tr>
<tr>
<td>([\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa, EOF}])</td>
<td>Iter 1, (\delta_a) is (\text{EOF})</td>
</tr>
<tr>
<td>([\text{SheepNoise} \rightarrow \bullet \text{baa, EOF}])</td>
<td>Iter 1, (\delta_a) is (\text{EOF})</td>
</tr>
<tr>
<td>([\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa, baa}])</td>
<td>Iter 2, (\delta_a) is (\text{baa EOF})</td>
</tr>
<tr>
<td>([\text{SheepNoise} \rightarrow \bullet \text{baa, baa}])</td>
<td>Iter 2, (\delta_a) is (\text{baa EOF})</td>
</tr>
</tbody>
</table>

So, \(S_0\) is

\[
\{ \begin{array}{c}
[\text{Goal} \rightarrow \bullet \text{SheepNoise},\text{EOF}],
[\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa,EOF}],
[\text{SheepNoise} \rightarrow \bullet \text{baa,EOF}],
[\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa,baa}],
[\text{SheepNoise} \rightarrow \bullet \text{baa,baa}] \\
\end{array} \}
\]

This is the left-recursive SheepNoise; EaC2e shows the right-recursive version.
Computing Gotos

\textbf{Goto}(s,x)\ computes\ the\ state\ that\ the\ parser\ would\ reach\ if\ it\ recognized\ an\ x\ while\ in\ state\ s

- \textbf{Goto}( \{ [A\rightarrow\beta\cdot X\delta,a] \}, X )\ produces\ \{ [A\rightarrow\beta X\cdot \delta,a] \}\ \textit{(obviously)}
- It finds all such items & uses \textit{Closure()}\ to fill out the state

The Algorithm

\begin{verbatim}
Goto( s , X )
    new <- \emptyset
    \forall items [A\rightarrow\beta\cdot X\delta,a] \in s
        new <- new \cup \{ [A\rightarrow\beta X\cdot \delta,a] \}
    return Closure( new )
\end{verbatim}

- \textbf{Goto}( )\ models\ a\ transition\ in\ the\ automaton
- Straightforward computation
- \textbf{Goto}( )\ is\ not\ a\ fixed-point\ method\ (but\ it\ calls\ \textit{Closure()})

\textit{Goto} in this construction is analogous to \textit{Move} in the subset construction.
Assume that $S_0$ is

\[
\{ \text{[Goal} \rightarrow \bullet \text{SheepNoise, EOF]}, \text{[SheepNoise} \rightarrow \bullet \text{SheepNoise baa, EOF]}, \\
\text{[SheepNoise} \rightarrow \bullet \text{ baa, EOF]}, \text{[SheepNoise} \rightarrow \bullet \text{SheepNoise baa, baa]}, \\
\text{[SheepNoise} \rightarrow \bullet \text{ baa, baa] } \}
\]

\( \text{Goto} (S_0, \text{ baa } ) \)

- Loop produces

\[
\begin{array}{|c|c|}
\hline
\text{Item} & \text{Source} \\
\hline
\text{[SheepNoise} \rightarrow \text{ baa } \bullet, \text{ EOF]} & \text{Item 3 in } s_0 \\
\text{[SheepNoise} \rightarrow \text{ baa } \bullet, \text{ baa]} & \text{Item 5 in } s_0 \\
\hline
\end{array}
\]

- **Closure** adds nothing since \( \bullet \) is at end of \text{rhs} in each item

In the construction, this produces \( s_2 \)

\[
\{ \text{[SheepNoise} \rightarrow \text{ baa } \bullet, \{\text{EOF, baa}\}] \}
\]
Building the Canonical Collection

Start from \( s_0 = \text{Closure}( [S' \rightarrow \bullet S, \text{EOF}] ) \)

Repeatedly construct new states, until all are found

**The Algorithm**

\[
\begin{align*}
  s_0 & \leftarrow \text{Closure} ( \{ [S' \rightarrow \bullet S, \text{EOF}] \} ) \\
  S & \leftarrow \{ s_0 \} \\
  k & \leftarrow 1 \\
  \text{while} (S \text{ is still changing}) \\
  & \quad \forall s_j \in S \text{ and } \forall x \in (T \cup NT) \\
  & \quad \quad s_k \leftarrow \text{Goto}(s_j, x) \\
  & \quad \quad \text{record } s_j \rightarrow s_k \text{ on } x \\
  & \quad \text{if } s_k \notin S \text{ then} \\
  & \quad \quad S \leftarrow S \cup \{ s_k \} \\
  & \quad \quad k \leftarrow k + 1
\end{align*}
\]

- Fixed-point computation
- Loop adds to \( S \) (monotone)
- \( S \subseteq 2^\text{ITEMS} \), so \( S \) is finite
- Worklist version is faster because it avoids duplicated effort

This membership / equality test requires careful and/or clever implementation.
Example from SheepNoise

Starts with $S_0$

$S_0 : \{ [Goal \rightarrow \bullet \ SheepNoise, EOF],
[S\ sheepNoise \rightarrow \bullet \ sheepNoise \ baa, EOF],
[S\ sheepNoise \rightarrow \bullet \ baa, EOF],
[S\ sheepNoise \rightarrow \bullet \ sheepNoise \ baa, baa],
[S\ sheepNoise \rightarrow \bullet \ baa, baa] \}$

Iteration 1 computes

$S_1 = \text{Goto}(S_0, \text{SheepNoise}) =
\{ [Goal \rightarrow \text{SheepNoise} \bullet, EOF],
[S\ sheepNoise \rightarrow \text{SheepNoise} \bullet \ sheepNoise \ baa, EOF],
[S\ sheepNoise \rightarrow \text{SheepNoise} \bullet \ baa, baa] \}$

$S_2 = \text{Goto}(S_0, \ baa) = \{ [\text{SheepNoise} \rightarrow \ baa \bullet, EOF],
[S\ sheepNoise \rightarrow \ baa \bullet, baa] \}$

Iteration 2 computes

$S_3 = \text{Goto}(S_1, \ baa) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise} \ baa \bullet, EOF],
[S\ sheepNoise \rightarrow \text{SheepNoise} \ baa \bullet, baa] \}$

Nothing more to compute, since \bullet is at the end of every item in $S_3$.

COMP 412, Fall 2017

| 0 | Goal | $\rightarrow$ | SheepNoise |
| 1 | SheepNoise | $\rightarrow$ | SheepNoise baa |
| 2 | | $\rightarrow$ | baa | 10 |
Example from SheepNoise

$S_0 : \{ [Goal \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise} \text{ baa}, \text{EOF}],
[\text{SheepNoise} \rightarrow \bullet \text{ baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise} \text{ baa, baa}],
[\text{SheepNoise} \rightarrow \bullet \text{ baa, baa}] \}$

$S_1 = \textbf{Goto}(S_0, \text{SheepNoise}) = \$
$\{ [\text{Goal} \rightarrow \text{SheepNoise} \bullet , \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{ baa}, \text{EOF}],
[\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{ baa, baa}] \}$

$S_2 = \textbf{Goto}(S_0, \text{ baa}) = \{ [\text{SheepNoise} \rightarrow \text{ baa} \bullet , \text{EOF}], [\text{SheepNoise} \rightarrow \text{ baa} \bullet , \text{ baa}] \}$

$S_3 = \textbf{Goto}(S_1, \text{ baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise} \text{ baa} \bullet , \text{EOF}],
[\text{SheepNoise} \rightarrow \text{SheepNoise} \text{ baa} \bullet , \text{ baa}] \}$

<table>
<thead>
<tr>
<th>State</th>
<th>SN</th>
<th>baa</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_1$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>—</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>$S_3$</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

*Goto Relationships*
Filling in the ACTION and GOTO Tables

The Table Construction Algorithm

\[ \forall \text{ set } S_x \subseteq S \]
\[ \forall \text{ item } i \subseteq S_x \]
\[ \text{if } i \text{ is } [A \rightarrow \beta \cdot \underline{a} \delta, b] \text{ and } \text{goto}(S_x, a) = S_k, a \in T \]
\[ \text{then } \text{ACTION}[x, a] \leftarrow "shift \ k" \]
\[ \text{else if } i \text{ is } [S' \rightarrow S \cdot, EOF] \]
\[ \text{then } \text{ACTION}[x, EOF] \leftarrow "accept" \]
\[ \text{else if } i \text{ is } [A \rightarrow \beta \cdot, \underline{a}] \]
\[ \text{then } \text{ACTION}[x, a] \leftarrow "reduce A \rightarrow \beta" \]
\[ \forall \ n \in NT \]
\[ \text{if } \text{goto}(S_x, n) = S_k \]
\[ \text{then } \text{GOTO}[x, n] \leftarrow k \]

**Many items generate no table entry**

\[ \rightarrow \text{ Placeholder before a } NT \text{ does not generate an ACTION table entry} \]
\[ \rightarrow \textbf{Closure}() \text{ instantiates FIRST}(X) \text{ directly for } [A \rightarrow \beta \cdot X \delta, a] \]
Example from SheepNoise

\( S_0 : \{ [\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{baa}], [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{baa}] \} \)

\( S_1 = \text{Goto}(S_0, \text{SheepNoise}) = \{ [\text{Goal} \rightarrow \text{SheepNoise } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{ baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{ baa, baa}] \} \)

\( S_2 = \text{Goto}(S_0, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{baa}] \} \)

\( S_3 = \text{Goto}(S_1, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise baa } \bullet, \text{baa}] \} \)

\( \bullet \text{ before } T \Rightarrow \text{shift } k \)

so, ACTION\([s_0, \text{baa}]\) is “shift \( S_2 \)” (clause 1)
(items define same entry)
Example from SheepNoise

\[ S_0 : \{ [\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa, baa}], [\text{SheepNoise} \rightarrow \bullet \text{baa, baa}] \} \]

\[ S_1 = \text{Goto}(S_0, \text{SheepNoise}) = \{ [\text{Goal} \rightarrow \text{SheepNoise } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{ baa, EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{ baa, baa}] \} \]

\[ S_2 = \text{Goto}(S_0, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{baa}] \} \]

\[ S_3 = \text{Goto}(S_1, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise baa } \bullet, \text{baa}] \} \]

so, ACTION\[S_1,\text{baa}\] is “shift \(S_3\)” (clause 1)
Example from SheepNoise

\[ S_0 : \{ [\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{EOF}],
\]
\[ [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{baa}],
\]
\[ [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{baa}] \} \]

\[ S_1 = \text{Goto}(S_0, \text{SheepNoise}) =
\]
\[ \{ [\text{Goal} \rightarrow \text{SheepNoise} \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{baa}, \text{EOF}],
\]
\[ [\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{baa}, \text{baa}] \} \]

\[ S_2 = \text{Goto}(S_0, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{baa} \bullet, \text{EOF}],
\]
\[ [\text{SheepNoise} \rightarrow \text{baa} \bullet, \text{baa}] \} \]

\[ S_3 = \text{Goto}(S_1, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise baa} \bullet, \text{EOF}],
\]
\[ [\text{SheepNoise} \rightarrow \text{SheepNoise baa} \bullet, \text{baa}] \} \]

so, ACTION[S_1, EOF] is “accept” (clause 2)
Example from SheepNoise

\[ S_0 : \{ [Goal \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise } \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise } \text{baa}, \text{baa}], [\text{SheepNoise} \rightarrow \bullet \text{baa, baa} ] \} \]

\[ S_1 = \text{Goto}(S_0, \text{SheepNoise}) = \{ [\text{Goal} \rightarrow \text{SheepNoise } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \bullet \text{baa, baa}] \} \]

\[ S_2 = \text{Goto}(S_0, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{baa } \bullet, \text{baa}] \} \]

\[ S_3 = \text{Goto}(S_1, \text{baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise } \text{baa } \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise } \text{baa } \bullet, \text{baa}] \} \]

so, ACTION[S_2, EOF] is “reduce 2” (clause 3) (baa, too)

ACTION[S_3, EOF] is “reduce 1” (clause 3) (baa, too)
Building the Goto Table

\[ S_0 : \{ [\text{Goal} \rightarrow \bullet \text{SheepNoise}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{ baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \bullet \text{SheepNoise baa, baa}], [\text{SheepNoise} \rightarrow \bullet \text{ baa, baa}] \} \]

\[ S_1 = \text{Goto}(S_0, \text{SheepNoise}) = \{ [\text{Goal} \rightarrow \text{SheepNoise} \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{ baa}, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise} \bullet \text{ baa, baa}] \} \]

\[ S_2 = \text{Goto}(S_0, \text{ baa}) = \{ [\text{SheepNoise} \rightarrow \text{ baa} \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{ baa} \bullet, \text{ baa}] \} \]

\[ S_3 = \text{Goto}(S_1, \text{ baa}) = \{ [\text{SheepNoise} \rightarrow \text{SheepNoise baa} \bullet, \text{EOF}], [\text{SheepNoise} \rightarrow \text{SheepNoise baa} \bullet, \text{ baa}] \} \]

The Goto table holds just the entries for nonterminal symbols.

(\textit{ignore the column for baa})

<table>
<thead>
<tr>
<th>State</th>
<th>SN</th>
<th>baa</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_0 )</td>
<td>( S_1 )</td>
<td>( S_2 )</td>
</tr>
<tr>
<td>( S_1 )</td>
<td>—</td>
<td>( S_3 )</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

\textit{Goto Relationships}

COMP 412, Fall 2017
ACTION & GOTO Tables

Here are the tables for the left-recursive *SheepNoise* grammar

The tables

<table>
<thead>
<tr>
<th>State</th>
<th>EOF</th>
<th>baa</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td><em>shift</em> 2</td>
</tr>
<tr>
<td>1</td>
<td><em>accept</em></td>
<td><em>shift</em> 3</td>
</tr>
<tr>
<td>2</td>
<td><em>reduce</em> 2</td>
<td><em>reduce</em> 2</td>
</tr>
<tr>
<td>3</td>
<td><em>reduce</em> 1</td>
<td><em>reduce</em> 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th><em>SheepNoise</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

The grammar

0  \textit{Goal} \rightarrow \textit{SheepNoise}  \\
1  \textit{SheepNoise} \rightarrow \textit{SheepNoise} \textit{baa}  \\
2  | \textit{baa}  \\

Remember, this is the left-recursive SheepNoise; EaC2e shows the right-recursive version.
What can go wrong?

What if a set \( s \) contains \([A \rightarrow \beta \cdot a \gamma, b]\) and \([B \rightarrow \beta \cdot, a]\)?

- First item generates “shift”, second generates “reduce”
- Both define \( \text{ACTION}[s, a] \) — cannot do both actions
- This is a fundamental ambiguity, called a \textit{shift/reduce error}
- Modify the grammar to eliminate it ((if-then-else))
- Shifting will often resolve it correctly

What if a set \( s \) contains \([A \rightarrow \gamma \cdot, a]\) and \([B \rightarrow \gamma \cdot, a]\)?

- Each generates “reduce”, but with a different production
- Both define \( \text{ACTION}[s, a] \) — cannot do both reductions
- This is a fundamental ambiguity, called a \textit{reduce/reduce conflict}
- Modify the grammar to eliminate it (PL/I’s overloading of (...))

\textit{In either case, the grammar is not LR(1)}
Implementing the Construction

Building the Canonical Collection

Start from \( s_0 = \text{closure}(\ [S' \rightarrow \bullet S, \text{EOF}] ) \)

Repeatedly construct new states, until all are found.

The algorithm

\[
\begin{align*}
  s_0 &\leftarrow \text{closure}(\ [S' \rightarrow \bullet S, \text{EOF}] ) \\
  S &\leftarrow \{ s_0 \} \\
  k &\leftarrow 1 \\
  \text{while } (S \text{ is still changing}) &\text{ do} \\
  &\forall s_j \in S \text{ and } \forall x \in (T \cup NT) \\
  &\quad s_k \leftarrow \text{goto}(s_j, x) \\
  &\quad \text{record } s_j \rightarrow s_k \text{ on } x \\
  &\quad \text{if } s_k \notin S \text{ then} \\
  &\quad \quad S \leftarrow S \cup \{ s_k \} \\
  &\quad \quad k \leftarrow k + 1
\end{align*}
\]

Remember this comment about implementing the equality test at the bottom of the algorithm to build the Canonical Collection of Sets of LR(1) Items?

- Only need to compare core items — the rest will follow
- Represent items as a triple (R,P,L)
  - R is the rule or production
  - P is the position of the placeholder
  - L is the lookahead symbol
- Order items, then
  1. Compare set cardinalities
  2. Compare (in order) by R, P, L

This membership / equality test requires careful and/or clever implementation.

STOP