CS415 Compilers
Syntax Analysis
Part 5

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University
Roadmap for the remainder of the course

• Fourth homework:
  Due Monday, March 28

• Project #2 – Bottom-up parser and compiler
  Will be posted next week, due April 13 (tentative)

• Project #3 – Peephole optimizer for ILOC
  Will be posted April 13, due May 2 (tentative)

• Second midterm on Wednesday, April 6 (60 minutes in class)

• Final exam on May 10 (60 minutes at assigned location)

• At least 3 more homeworks
Bottom-up Parsing (Syntax Analysis)

EAC Chapters 3.4
The skeleton parser
- uses ACTION & GOTO tables
- does \(|words| shifts
- does \(|derivation| reductions
- does 1 accept

```java
stack.push(INVALID); stack.push(s0);
not_found = true;
token = scanner.next_token();
do while (not_found) {
  s = stack.top();
  if (ACTION[s,token] == "reduce A→β") {
    stack.popnum(2*|β|); // pop 2*|β| symbols
    s = stack.top();
    stack.push(A);
    stack.push(GOTO[s,A]);
  }
  else if (ACTION[s,token] == "shift s") {
    stack.push(token); stack.push(s);
    token ← scanner.next_token();
  }
  else if (ACTION[s,token] == "accept" & token == EOF )
    then not_found = false;
  else report a syntax error and recover;
}
report success;
```
Building LR(1) Parsers

How do we generate the ACTION and GOTO tables?
• Use the grammar to build a model of the DFA
• Use the model to build ACTION & GOTO tables
• If construction succeeds, the grammar is LR(1)

The Big Picture
• Model the state of the parser
• Use two functions $\text{goto}(s, X)$ and $\text{closure}(s)$
  → $\text{goto}()$ is analogous to $\text{move}()$ in the subset construction
  → $\text{closure}()$ adds information to round out a state
• Build up the states and transition functions of the DFA
• Use this information to fill in the ACTION and GOTO tables
The LR(1) table construction algorithm uses LR(1) items to represent valid configurations of an LR(1) parser.

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

- \(P\) is a production \(A \rightarrow \beta\) with a \(\cdot\) at some position in the \(rhs\)
- \(\delta\) is a lookahead string of length \(\leq \kappa\) (words or \(EOF\))

The \(\cdot\) in an item indicates the position of the top of the stack.

LR(1):

- \([A \rightarrow \cdot \beta \gamma, a]\) means that the input seen so far is consistent with the use of \(A \rightarrow \beta \gamma\) immediately after the symbol on top of the stack.
- \([A \rightarrow \beta \cdot \gamma, a]\) means that the input seen so far is consistent with the use of \(A \rightarrow \beta \gamma\) at this point in the parse, and that the parser has already recognized \(\beta\).
- \([A \rightarrow \beta \gamma \cdot, a]\) means that the parser has seen \(\beta \gamma\), and that a lookahead symbol of \(a\) is consistent with reducing to \(A\).
The production $A \rightarrow \beta$, where $\beta = B_1B_1B_1$ with lookahead $a$, can give rise to 4 items:

- $[A \rightarrow \cdot B_1B_2B_3,a]$
- $[A \rightarrow B_1 \cdot B_2B_3,a]$
- $[A \rightarrow B_1B_2 \cdot B_3,a]$
- $[A \rightarrow B_1B_2B_3 \cdot ,a]$

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

What’s the point of all these lookahead symbols?

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

⇒ Limited right context is enough to pick the actions (unique, i.e., deterministic choice).
High-level overview

1. **Build the canonical collection of sets of LR(1) Items, I**
   a. Begin in an appropriate state, $s_0$
      - Assume: $S' \rightarrow S$, and $S'$ is unique start symbol that does not occur on any RHS of a production (extended CFG - ECFG)
      - $[S' \rightarrow \cdot S, EOF]$, 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 collection of sets of LR(1) items**

   The canonical collection completely encodes the transition diagram for the handle-finding DFA
Computing Closures

Closure(s) adds all the items implied by items already in s

• Any item \([A \rightarrow B \delta, a] \) implies \([B \rightarrow \tau, x] \) for each production with \(B \) on the lhs, and each \(x \in \text{FIRST}(\delta a) \)

The algorithm

\[
\begin{align*}
\text{Closure}(s) & \\
\text{while ( s is still changing )} & \\
& \forall \text{ items } [A \rightarrow B \delta, a] \in s \\
& \forall \text{ productions } B \rightarrow \tau \in P \\
& \forall b \in \text{FIRST}(\delta a) \quad // \quad \delta \text{ might be } \varepsilon \\
& \quad \text{if } [B \rightarrow \tau, b] \notin s \\
& \quad \text{then add } [B \rightarrow \tau, b] \text{ to } s
\end{align*}
\]

- Classic fixed-point method
- Halts because \(s \subseteq \text{ITEMS} \)
- Closure "fills out" a state
Computing Gotos

$Goto(s, x)$ computes the state that the parser would reach if it recognized an $X$ while in state $s$

- $Goto(\{ [A\rightarrow \beta \cdot X \delta, a] \}, X)$ produces $[A\rightarrow \beta X \cdot \delta, a]$  \hspace{1cm} (easy part)
- Should also includes $\text{closure}( [A\rightarrow \beta X \cdot \delta, a] )$ \hspace{1cm} (fill out the state)

The algorithm

```
Goto(s, X)
new \leftarrow \emptyset
\forall \text{ items } [A\rightarrow \beta \cdot X \delta, a] \in s
\text{new} \leftarrow \text{new} \cup [A\rightarrow \beta X \cdot \delta, a]
\text{return closure(new)}
```

- Not a fixed-point method!
- Straightforward computation
- Uses $\text{closure( )}$

$Goto()$ moves forward
Building the Canonical Collection

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

Repeatedly construct new states, until all are found

The algorithm

\[
\begin{align*}
cc_0 & \leftarrow \text{closure}( [S' \rightarrow \bullet S, \text{EOF}] ) \\
CC & \leftarrow \{ cc_0 \} \\
\text{while ( new sets are still being added to } CC) & \\
\text{for each unmarked set } cc_j \in CC & \\
\text{mark } cc_j \text{ as processed} & \\
\text{for each } x \text{ following a } \bullet \text{ in an item in } cc_j & \\
\text{temp } & \leftarrow \text{goto}(cc_j, x) \\
\text{if temp } & \notin CC \\
\text{then } CC & \leftarrow CC \cup \{ \text{temp} \} \\
\text{record transitions from } cc_j \text{ to temp on } x
\end{align*}
\]

- Fixed-point computation
  (worklist version)
- Loop adds to \( CC \)
- \( CC \subseteq 2^\text{ITEMS} \), so \( CC \) is finite
Simplified, right recursive expression grammar

1: Goal → Expr
2: Expr → Term - Expr
3: Expr → Term
4: Term → Factor * Term
5: Term → Factor
6: Factor → ident

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Expr</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Term</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Factor</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>{ - }</td>
</tr>
<tr>
<td>*</td>
<td>{ * }</td>
</tr>
<tr>
<td>ident</td>
<td>{ ident }</td>
</tr>
</tbody>
</table>
Another Example (grammar & sets)

1: \( \text{Goal} \rightarrow \text{Expr} \)
2: \( \text{Expr} \rightarrow \text{Term} - \text{Expr} \)
3: \( \text{Expr} \rightarrow \text{Term} \)
4: \( \text{Term} \rightarrow \text{Factor} * \text{Term} \)
5: \( \text{Term} \rightarrow \text{Factor} \)
6: \( \text{Factor} \rightarrow \text{ident} \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Expr</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Term</td>
<td>{ ident }</td>
</tr>
<tr>
<td>Factor</td>
<td>{ ident }</td>
</tr>
<tr>
<td>-</td>
<td>{ - }</td>
</tr>
<tr>
<td>*</td>
<td>{ * }</td>
</tr>
<tr>
<td>ident</td>
<td>{ ident }</td>
</tr>
</tbody>
</table>

\( s_0 \leftarrow \text{closure( [Goal} \rightarrow \cdot \text{Expr} , \text{EOF}]) = \)

\[
\begin{align*}
\{ & \{ \text{Goal} \rightarrow \cdot \text{Expr} , \text{EOF} \}, \{ \text{Expr} \rightarrow \cdot \text{Term} - \text{Expr} , \text{EOF} \}, \\
& \{ \text{Expr} \rightarrow \cdot \text{Term} , \text{EOF} \}, \{ \text{Term} \rightarrow \cdot \text{Factor} * \text{Term} , \text{EOF} \}, \\
& \{ \text{Term} \rightarrow \cdot \text{Factor} * \text{Term} , - \}, \{ \text{Term} \rightarrow \cdot \text{Factor} , \text{EOF} \}, \\
& \{ \text{Term} \rightarrow \cdot \text{Factor} , - \}, \{ \text{Factor} \rightarrow \cdot \text{ident} , \text{EOF} \}, \\
& \{ \text{Factor} \rightarrow \cdot \text{ident} , - \}, \{ \text{Factor} \rightarrow \cdot \text{ident} , * \} \} \}
\]
Iteration 1

\[ s_1 \leftarrow \text{goto}(s_0, \text{Expr}) \]
\[ s_2 \leftarrow \text{goto}(s_0, \text{Term}) \]
\[ s_3 \leftarrow \text{goto}(s_0, \text{Factor}) \]
\[ s_4 \leftarrow \text{goto}(s_0, \text{ident}) \]

Iteration 2

\[ s_5 \leftarrow \text{goto}(s_2, \text{-}) \]
\[ s_6 \leftarrow \text{goto}(s_3, \text{*}) \]

Iteration 3

\[ s_7 \leftarrow \text{goto}(s_5, \text{Expr}) \]
\[ s_8 \leftarrow \text{goto}(s_6, \text{Term}) \]
Example  (Summary)

\[ S_0 : \{ \begin{array}{l}
  [\text{Goal} \rightarrow \cdot \text{Expr} , \text{EOF}],
  [\text{Expr} \rightarrow \cdot \text{Term} \cdot \text{-} \text{Expr} , \text{EOF}],
  [\text{Expr} \rightarrow \cdot \text{Term} , \text{EOF}],
  [\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{*} \text{Term} , \text{EOF}],
  [\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{*} \text{Term} , -],
  [\text{Term} \rightarrow \cdot \text{Factor} , \text{EOF}],
  [\text{Term} \rightarrow \cdot \text{Factor} , -],
  [\text{Factor} \rightarrow \cdot \text{ident} , \text{EOF}],
  [\text{Factor} \rightarrow \cdot \text{ident} , -],
  [\text{Factor} \rightarrow \cdot \text{ident} , \cdot , *] \end{array} \} \]

\[ S_1 : \{ [\text{Goal} \rightarrow \text{Expr} \cdot , \text{EOF}] \} \]

\[ S_2 : \{ [\text{Expr} \rightarrow \text{Term} \cdot \cdot \text{-} \text{Expr} , \text{EOF}],
  [\text{Expr} \rightarrow \text{Term} \cdot , \text{EOF}] \} \]

\[ S_3 : \{ [\text{Term} \rightarrow \text{Factor} \cdot \cdot \text{*} \text{Term} , \text{EOF}],
  [\text{Term} \rightarrow \text{Factor} \cdot \cdot \text{*} \text{Term} , -],
  [\text{Term} \rightarrow \text{Factor} \cdot , \text{EOF}],
  [\text{Term} \rightarrow \text{Factor} \cdot , -] \} \]

\[ S_4 : \{ [\text{Factor} \rightarrow \text{ident} \cdot , \text{EOF}],
  [\text{Factor} \rightarrow \text{ident} \cdot , -],
  [\text{Factor} \rightarrow \text{ident} \cdot , \cdot , *] \} \]

\[ S_5 : \{ [\text{Expr} \rightarrow \text{Term} \cdot \cdot \cdot \text{-} \text{Expr} , \text{EOF}],
  [\text{Expr} \rightarrow \text{Term} \cdot \cdot \text{*} \text{Term} , \text{EOF}],
  [\text{Expr} \rightarrow \cdot \text{Term} , \text{EOF}],
  [\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{*} \text{Term} , -],
  [\text{Term} \rightarrow \cdot \text{Factor} , -],
  [\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{*} \text{Term} , \text{EOF}],
  [\text{Term} \rightarrow \cdot \text{Factor} , \text{EOF}],
  [\text{Factor} \rightarrow \cdot \text{ident} , \cdot , *],
  [\text{Factor} \rightarrow \cdot \text{ident} , -],
  [\text{Factor} \rightarrow \cdot \text{ident} , \text{EOF}] \} \]
Example (Summary)

\[ S_6 : \{ [\text{Term} \rightarrow \text{Factor} \cdot \text{Term}, \text{EOF}], [\text{Term} \rightarrow \text{Factor} \cdot \text{Term}, -], \\
[\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{Term}, \text{EOF}], [\text{Term} \rightarrow \cdot \text{Factor} \cdot \text{Term}, -], \\
[\text{Term} \rightarrow \cdot \text{Factor}, \text{EOF}], [\text{Term} \rightarrow \cdot \text{Factor}, -], \\
[\text{Factor} \rightarrow \cdot \text{ident}, \text{EOF}], [\text{Factor} \rightarrow \cdot \text{ident}, -], [\text{Factor} \rightarrow \cdot \text{ident}, *] \} \]

\[ S_7 : \{ [\text{Expr} \rightarrow \text{Term} - \text{Expr} \cdot, \text{EOF}] \} \]

\[ S_8 : \{ [\text{Term} \rightarrow \text{Factor} \cdot \text{Term} \cdot, \text{EOF}], [\text{Term} \rightarrow \text{Factor} \cdot \text{Term} \cdot, -] \} \]
### The Goto Relationship (from the construction)

<table>
<thead>
<tr>
<th>State</th>
<th>Expr</th>
<th>Term</th>
<th>Factor</th>
<th>-</th>
<th>*</th>
<th>ident</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (DFA)

- State $s_0$: Start state.
- Transition: $s_0 \xrightarrow{ident} s_4$.
- Transition: $s_0 \xrightarrow{term} s_2$.
- Transition: $s_0 \xrightarrow{expr} s_1$.
- Transition: $s_2 \xrightarrow{-} s_7$.
- Transition: $s_2 \xrightarrow{term} s_1$.
- Transition: $s_4 \xrightarrow{ident} s_5$.
- Transition: $s_4 \xrightarrow{factor} s_3$.
- Transition: $s_5 \xrightarrow{ident} s_4$.
- Transition: $s_5 \xrightarrow{factor} s_7$.
- Transition: $s_1 \xrightarrow{expr} s_0$.
- Transition: $s_3 \xrightarrow{*} s_6$.
- Transition: $s_6 \xrightarrow{term} s_7$.
- Transition: $s_7 \xrightarrow{expr} s_0$.
Filling in the ACTION and GOTO Tables

The algorithm

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

**x is the number of the state for s_x**

Many items generate no table entry

* e.g., \([A \rightarrow \beta \cdot B\alpha, a]\) does not, but closure ensures that all the rhs' for B are in s_x
The algorithm produces the following table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ident</td>
<td>EOF</td>
</tr>
<tr>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>s 4</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s 5</td>
</tr>
<tr>
<td>3</td>
<td>r 5</td>
</tr>
<tr>
<td>4</td>
<td>r 6</td>
</tr>
<tr>
<td>5</td>
<td>s 4</td>
</tr>
<tr>
<td>6</td>
<td>s 4</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r 4</td>
</tr>
</tbody>
</table>

Plugs into the skeleton LR(1) parser
What can go wrong?

What if 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 **shift/reduce error**
- Modify the grammar to eliminate it (if-then-else)
- Shifting will often resolve it correctly

What is 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 fundamental ambiguity is called a **reduce/reduce error**
- Modify the grammar to eliminate it

In either case, the grammar is not LR(1)
Computing Closures - LR(0) items

Closure(s) adds all the items implied by items already in s

- Any item \([A \rightarrow \beta \cdot B \delta]\) implies \([B \rightarrow \cdot \tau]\) for each production with \(B\) on the lhs

The algorithm

\[
\text{Closure}(s) \quad \text{while (s is still changing)} \\
\forall \text{ items } [A \rightarrow \beta \cdot B \delta] \in s \\
\forall \text{ productions } B \rightarrow \tau \in P \\
\text{ if } [B \rightarrow \cdot \tau] \notin s \\
\text{ then add } [B \rightarrow \cdot \tau] \text{ to } s
\]

- Classic fixed-point method
- Halts because \(s \subseteq \text{ITEMS}\)
Closure "fills out" a state
Context-Sensitive Analysis