Bottom-up Parsing
Part 4

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University
Announcements

• Office hours this Thursday (tomorrow): 12:30 – 2:00pm

• Homework #5 has been posted

• Midterm will be returned in recitation, starting today

Midterm sample solutions are available on sakai/Resources

You have until Thursday, April 11, to challenge your grade. Please check with sample solution first if you have any questions.
The LR(1) table construction algorithm uses LR(1) items to represent valid configurations of an LR(1) parser.

An LR(k) item is a pair \([P, δ]\), where
- \(P\) is a production \(A → β\) with a • at some position in the \(rhs\)
- \(δ\) is a lookahead string of length \(≤ k\) (words or EOF)

The • in an item indicates the position of the top of the stack.

**LR(1):**

\([A → • βγ,a]\) means that the input seen so far is consistent with the use of \(A → βγ\) immediately after the symbol on top of the stack.

\([A → β • γ,a]\) means that the input seen so far is consistent with the use of \(A → βγ\) at this point in the parse, and that the parser has already recognized \(β\).

\([A → βγ •,a]\) means that the parser has seen \(βγ\), and that a lookahead symbol of \(a\) is consistent with reducing to \(A\).
Closure(s) adds all the items implied by items already in s

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

The algorithm

```
Closure(s)
    while (s is still changing)
        \(\forall \text{ items } [A \rightarrow \beta \cdot B \delta, a] \in s\)
        \(\forall \text{ productions } B \rightarrow \tau \in P\)
        \(\forall \ b \in \text{FIRST}(\delta a) // \delta \text{ might be } \varepsilon\)
        if \([B \rightarrow \cdot \tau, b] \notin s\)
            then add \([B \rightarrow \cdot \tau, b]\) to s
```
Review - Computing Gotos

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

- Goto( { [A→β•Xδ,a] }, X ) produces [A→βX•δ,a] (easy part)
- Should also includes closure( [A→βX•δ,a] ) (fill out the state)

The algorithm

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

- Not a fixed-point method!
- Straightforward computation
- Uses closure()

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

Repeatedly construct new states, until all are found

The algorithm

\[
cc_0 \leftarrow \text{closure}([S'\rightarrow \bullet S, \text{EOF}])
\]

\[
CC \leftarrow \{ cc_0 \}
\]

while (new sets are still being added to CC)

for each unmarked set $cc_j \in CC$

mark $cc_j$ as processed

for each $x$ following a $\bullet$ in an item in $cc_j$

$\text{temp} \leftarrow \text{goto}(cc_j, x)$

if $\text{temp} \not\in CC$

then $CC \leftarrow CC \cup \{ \text{temp} \}$

record transitions from $cc_j$ to temp on $x$

- 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>{ iden }</td>
</tr>
<tr>
<td>−</td>
<td>{ − }</td>
</tr>
<tr>
<td>*</td>
<td>{ * }</td>
</tr>
<tr>
<td>ident</td>
<td>{ ident }</td>
</tr>
</tbody>
</table>
Initialization Step

\[ S_0 \leftarrow \text{closure}( \{ [\text{Goal} \rightarrow \cdot \text{Expr} \ , \ EOF] \} ) \]

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

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

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

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

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

[\text{Term} \rightarrow \cdot \text{Factor} \ , \ -],

[\text{Factor} \rightarrow \cdot \text{id} \ , \ EOF],

[\text{Factor} \rightarrow \cdot \text{id} \ , \ -], \ [\text{Factor} \rightarrow \cdot \text{id} \ , \ *] \} \]

\[ S \leftarrow \{ s_0 \} \]
Example (building the collection)

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

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

Iteration 3

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

$S_0 : \{ [Goal \rightarrow \cdot Expr, EOF], [Expr \rightarrow \cdot Term - Expr, EOF],$

$[Expr \rightarrow \cdot Term, EOF], [Term \rightarrow \cdot Factor * Term, EOF],$

$[Term \rightarrow \cdot Factor * Term, -], [Term \rightarrow \cdot Factor, EOF],$

$[Term \rightarrow \cdot Factor, -], [Factor \rightarrow \cdot ident, EOF],$

$[Factor \rightarrow \cdot ident, -], [Factor \rightarrow \cdot ident, *] \}$

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

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

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

$[Term \rightarrow Factor \cdot, EOF], [Term \rightarrow Factor \cdot, -] \}$

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

$S_5 : \{ [Expr \rightarrow Term - \cdot Expr, EOF], [Expr \rightarrow \cdot Term - Expr, EOF],$

$[Expr \rightarrow \cdot Term, EOF], [Term \rightarrow \cdot Factor * Term, -],$

$[Term \rightarrow \cdot Factor, -], [Term \rightarrow \cdot Factor * Term, EOF],$

$[Term \rightarrow \cdot Factor, EOF], [Factor \rightarrow \cdot ident, *],$

$[Factor \rightarrow \cdot ident, -], [Factor \rightarrow \cdot ident, EOF] \}$
Example (Summary)

\[ \begin{align*}
S_6 : & \{ \text{[Term} \to \text{Factor} \ast \cdot \text{Term} \text{, EOF]}, \text{[Term} \to \text{Factor} \ast \cdot \text{Term} \text{, -}] \\
& \text{[Term} \to \cdot \text{Factor} \ast \text{Term} \text{, EOF]}, \text{[Term} \to \cdot \text{Factor} \ast \text{Term} \text{, -}] \\
& \text{[Term} \to \cdot \text{Factor} \text{, EOF]}, \text{[Term} \to \cdot \text{Factor} \text{, -}] \\
& \text{[Factor} \to \cdot \text{ident} \text{, EOF]}, \text{[Factor} \to \cdot \text{ident} \text{, -}], \text{[Factor} \to \cdot \text{ident} \ast \text{]} \} \\
S_7 : & \{ \text{[Expr} \to \text{Term} - \text{Expr} \ast, \text{EOF}] \}
\end{align*}\]

\[ \begin{align*}
S_8 : & \{ \text{[Term} \to \text{Factor} \ast \text{Term} \ast, \text{EOF]}, \text{[Term} \to \text{Factor} \ast \text{Term} \ast, \text{ -}] \}
\end{align*}\]
## 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>5</td>
<td></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)
Filling in the ACTION and GOTO Tables

The algorithm

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

\[ x \text{ is the number of the state for } s_x \]

Many items generate no table entry

\[ \text{e.g., } [A \rightarrow \beta \cdot B\alpha, a] \]
does not, but closure ensures that all the rhs’ for \( B \) are in \( s_x \)
Example (Filling in the tables)

The algorithm produces the following table

<table>
<thead>
<tr>
<th></th>
<th>ACTION</th>
<th></th>
<th></th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ident</td>
<td>-</td>
<td>*</td>
<td>EOF</td>
</tr>
<tr>
<td>0</td>
<td>s 4</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>acc</td>
</tr>
<tr>
<td>2</td>
<td>s 5</td>
<td></td>
<td>r 3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>r 5</td>
<td>s 6</td>
<td>r 5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r 6</td>
<td>r 6</td>
<td>r 6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s 4</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>s 4</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>r 2</td>
</tr>
<tr>
<td>8</td>
<td>r 4</td>
<td></td>
<td>r 4</td>
<td></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 $ACTION[s, a]$, cannot do both actions
- This is a fundamental ambiguity, called a *shift/reduce error*
- Modify the grammar to eliminate it
- Shifting will often resolve it correctly

What is set $s$ contains $[A \rightarrow \gamma \cdot a, a]$ and $[B \rightarrow \gamma \cdot a]$?

- Each generates “reduce”, but with a different production
- Both define $ACTION[s, a]$, 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 $LR(1)$
Shrinking the Tables

Three options:

• **Combine terminals such as** number & identifier, + & -, *, & /
  → Directly removes a column, may remove a row
  → For expression grammar, 198 (vs. 384) table entries

• **Combine rows or columns** *(table compression)*
  → Implement identical rows once & remap states
  → Requires extra indirection on each lookup
  → Use separate mapping for ACTION & for GOTO

• **Use another construction algorithm**
  → Both LALR(1) and SLR(1) produce smaller tables
  → Implementations are readily available
LR(k) versus LL(k)

Finding Reductions

LR(k) \Rightarrow \text{Each reduction in the parse is detectable with}
- the complete left context,
- the reducible phrase, itself, and
- the } k \text{ terminal symbols to its right}

LL(k) \Rightarrow \text{Parser must select the next rule based on}
- The complete left context
- The next } k \text{ terminals

Thus, LR(k) examines more context
## Summary

**Advantages**

- Easy to implement
- Good locality (fast)
- Simplicity
- Easy to embed actions (code access)

**Disadvantages**

- Hand-coded
- High maintenance
- Right associativity

<table>
<thead>
<tr>
<th>Method</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-down recursive</td>
<td>Easy to implement</td>
<td>Hand-coded</td>
</tr>
<tr>
<td>descent</td>
<td>Good locality (fast)</td>
<td>High maintenance</td>
</tr>
<tr>
<td></td>
<td>Simplicity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Easy to embed actions (code access)</td>
<td></td>
</tr>
<tr>
<td>LR(1)</td>
<td>Fast</td>
<td>Large working sets</td>
</tr>
<tr>
<td></td>
<td>Deterministic langs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Automatable (tool support)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Left associativity</td>
<td></td>
</tr>
</tbody>
</table>
LR(0) versus SLR(1) versus LR(1)

Example:

\[ S' \rightarrow S \]
\[ S \rightarrow S ; a \mid a \]

LR(0) ?

LR(1) ?

SLR(1) ? SIMPLE LR(1)

**SLR(1):** add FOLLOW(A) to each LR(0) item \([A \rightarrow \gamma^*]\) as its second component: \([A \rightarrow \gamma^*, a], \forall a \in \text{FOLLOW}(A)\);

Note: Can also add to other items, but does not really matter.
LALR(1) versus LR(1)

Example: $S' \rightarrow S$
$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$
$A \rightarrow c$
$B \rightarrow c$

LR(0) ?

LR(1) ?

LALR(1) ?

LALR(1): Merge two sets of LR(1) items (states), if they have the same core.

core of set of LR(1) items: set of LR(0) items derived by ignoring the lookahead symbols
s0 = Closure(\{[S' \rightarrow .S, \text{eof}]\})

s1 = Closure( GoTo (s0, a)) =
\{[S \rightarrow a . A \text{d}, \text{eof}],
[S \rightarrow a . B \text{e}, \text{eof}],
[A \rightarrow .c, d], [ B \rightarrow .c, e]\}

s2 = Closure( GoTo (s0, b)) =
\{[S \rightarrow b . A \text{e}, \text{eof}],
[S \rightarrow b . B \text{d}, \text{eof}],
[A \rightarrow .c, e], [ B \rightarrow .c, d]\}

s3 = Closure( GoTo (s1, c)) =
\{[A \rightarrow c . , d],
[B \rightarrow c . , e]\}

s4 = Closure( GoTo (s2, c)) =
\{[A \rightarrow c . , e],
[B \rightarrow c . , d]\}

There are other states that are not listed here!

Grammar is LR(1), but not LALR(1), since collapsing s3 and s4 (same core) will introduce reduce-reduce conflict.
LALR(1) versus LR(1)

Example:

\[
S' \rightarrow S \\
S \rightarrow aAd \mid bBd \mid aBe \mid bAe \\
A \rightarrow c \\
B \rightarrow c
\]

LR(0) ?

LR(1) ?

LALR(1) ?

**LALR(1):** Merge two sets of LR(1) items (states), if they have the same core.

**core** of set of LR(1) items: set of LR(0) items derived by ignoring the lookahead symbols

**FACT:** collapsing LR(1) states into LALR(1) states cannot introduce shift/reduce conflicts
Hierarchy of Context-Free Languages

Context-free languages

↓

Deterministic languages (LR(k))

↓

LR(k) \equiv LR(1)

LL(k) languages

↓

LL(1) languages

Simple precedence languages

↓

Operator precedence languages

The inclusion hierarchy for context-free languages
Hierarchy of Context-Free Grammars

- Operator precedence includes some ambiguous grammars
- LL(1) is a subset of SLR(1)

The inclusion hierarchy for context-free grammars
Error Recovery

Context-Sensitive Analysis

Read EaC: Chapters 4.1 - 4.3