More Bottom-Up Parsing

Lecture 7

Dr. Sean Peisert – ECS 142 – Spring 2009
Status

- Project 1 Back By Wednesday (ish)
- savior lexer in ~cs142/s09/bin
- Project 2 Due Friday, Apr. 24, 11:55pm
- My office hours 3pm today
- Discussion section 4:10pm today
The Stack

- Shift pushes a terminal on the stack
- Reduce pops 0 or more symbols off of the stack (production rhs) and pushes a non-terminal on the stack (production lhs)
When to Shift vs. Reduce?

- Decide based on the left string (the stack)
- Idea: use a finite automaton (DFA) to decide when to shift or reduce
  - The DFA input is the stack
  - The language consists of terminals and non-terminals
- We run the DFA on the stack and we examine the resulting state \( X \) and the token \( T \) after ▶️
  - If \( X \) has a transition labeled \( T \) then shift
  - If \( X \) is labeled with “\( A \rightarrow \beta \) on \( T \)” then reduce
• int + (int) + (int)\$ shift
int • + (int) + (int)\$ E → int
E • + (int) + (int)\$ shift(x3)
E + (int•) + (int)\$ E → int
E + (E•) + (int)\$ shift
E + (E)• + (int)E \rightarrow E + (E)
E• + (int)$ shift(x3)
E + (int•)$ E → int
E + (E)$ shift
E + (E)$ E → E + (E)
E•$ accept

For grammar \( E \rightarrow E + (E) \mid \text{int} \)
How is the DFA Constructed?

- The stack describes the “context” of the parse
  - What non-terminal we are looking for
  - What production rhs we are looking for
  - What we have seen so far from the rhs
- Each DFA state describes several such contexts
  - E.g., when we are looking for non-terminal E, we might be looking either for an int or an E+(E) rhs
LR(1) Items

- An LR(1) item is a pair, e.g.,:
  \[ X \rightarrow \alpha \cdot \beta, \ a \]
- \( X \rightarrow \alpha \beta \) is a production
- \( a \) is a terminal (the lookahead terminal)
- LR(1) means 1 lookahead terminal
- \([X \rightarrow \alpha \cdot \beta, a]\) describes a context of the parser
  - We are trying to find an \( X \) followed by an \( a \), and
  - We already have \( \alpha \) on the top of the stack
  - Thus we need to see a prefix derived from \( \beta a \)
Note

• Symbol “•” separates stack from rest of input: $\alpha \cdot \gamma$
  where $\alpha$ is the stack and $\gamma$ is the remaining string of terminals.

• In LR(1) items, • is used to mark a prefix of a production RHS:
  $$X \rightarrow \alpha \cdot \beta, a$$
  • Here, $\beta$ might contain non-terminals as well.

• In both cases, the stack is on the left.
Convention

- We add to our grammar a fresh new start symbol and a production $S \rightarrow E$
  - Where $E$ is the old start symbol
- The initial parsing context contains:
  - $S \rightarrow \cdot E, \$
- Trying to find an $S$ as a string derived from $E\$
- The stack is empty
LR(1) Items

- In context containing
  \[ E \rightarrow E + \cdot (E), + \]
  - If '(' follows then we can perform a shift to context containing
  \[ E \rightarrow E + (\cdot E), + \]
- In context containing
  \[ E \rightarrow E + (E)\cdot, + \]
  - We can perform a reduction with \[ E \rightarrow E + (E) \]
  - But only if a + follows
LR(1) Items

- Consider a context with the item
  \[ E \rightarrow E + (\cdot E), + \]
- We expect a string derived from \( E \) \( ) + \)
- There are two productions for \( E \)
  \[ E \rightarrow E + (E) \] and \( E \rightarrow \text{int} \)
- We describe this by extending the context with two more items:
  \[ E \rightarrow \cdot \text{int}, ) \]
  \[ E \rightarrow \cdot E + (E), ) \]
First and Follows

- Consider the state $S \rightarrow \beta \cdot A \gamma$
- We’re trying to match the string $\beta \beta \delta$
- Suppose that $b$ is the next token.
- Either:
  - $A \rightarrow \alpha$, if $b$ can start a string derived from $\alpha$
    - We say that $b \in \text{First}(\alpha)$
  - Or, the expansion of $A$ is empty and $b$ belongs to an expansion of $\gamma$ (e.g., $\gamma \rightarrow b \omega$).
  - $b$ can appear after $A$ in a derivation of the form $S \rightarrow \beta A b \omega$
    - We say that $b \in \text{Follow}(A)$ in this case.
What productions can we use?

• Consider the state $S \rightarrow \beta \cdot A \gamma$
• We’re trying to match the string $\beta b \delta$
  • The expansion of $A$ is empty and $b$ belongs to an expansion of $\gamma$ (e.g., $\gamma \rightarrow b \omega$).
  • $b$ can appear after $A$ in a derivation of the form $S \rightarrow \cdot \beta Ab \omega$
• We say that $b \in \text{Follow}(A)$ in this case.

• Can use as a production:
  • $A \rightarrow \alpha$ can be used if $\alpha$ can expand to $\varepsilon$
  • We say that $\varepsilon \rightarrow \text{First}(A)$ in this case
Computing First Sets

- Definition: First(X) = \{b | X \to b\alpha\} \cup \{\varepsilon | X \to \varepsilon\}
- First (b) = \{b\}
- For all productions X \to A_1 \ldots A_n
  - Add First (A_1) - \{\varepsilon\} to First(X). Stop if \varepsilon \notin First(A_1)
  - ...
  - Add First (A_n) - \{\varepsilon\} to First(X). Stop if \varepsilon \notin First(A_n)
  - Add \varepsilon to First(X) (ignore A_i if it is X)
First sets Example

• Grammar:
  \[ E \rightarrow TX \]
  \[ T \rightarrow (E) \mid \text{int}Y \]
  \[ X \rightarrow + E \mid \varepsilon \]
  \[ Y \rightarrow *T \mid \varepsilon \]

• First sets
  \[ \text{First} \ ( ( ) ) = \{ ( ) \} \]
  \[ \text{First} \ ( ) ) = \{ ( ) \} \]
  \[ \text{First} \ ( \text{int} ) = \{ \text{int} \} \]
  \[ \text{First} \ ( + ) = \{ + \} \]
  \[ \text{First} \ ( * ) = \{ * \} \]
  \[ \text{First} \ ( \text{T} ) = \{ \text{int}, ( ) \} \]
  \[ \text{First} \ ( \text{E} ) = \{ \text{int}, ( ) \} \]
  \[ \text{First} \ ( \text{X} ) = \{ +, \varepsilon \} \]
  \[ \text{First} \ ( \text{Y} ) = \{ *, \varepsilon \} \]
Computing *Follow* Sets

- Definition: $\text{Follow}(X) = \{ b \mid S \to \beta X b \omega \}$
- Compute the First sets for all non-terminals first
- Add $\$ \text{ to } \text{Follow}(S)$ (if $S$ is the start non-terminal)
- For all productions $Y \to X A_1 \ldots A_n$
  - Add $\text{First}(A_1) - \{ \varepsilon \}$ to $\text{Follow}(X)$. Stop if $\varepsilon \notin \text{First}(A_1)$
  - ...
  - Add $\text{First}(A_n) - \{ \varepsilon \}$ to $\text{Follow}(X)$. Stop if $\varepsilon \notin \text{First}(A_n)$
  - Add $\text{Follow}(Y)$ to $\text{Follow}(X)$
Follow sets Example

• Grammar:

\[
E \rightarrow TX \\
T \rightarrow (E) \mid \text{int} \ Y
\]

\[
X \rightarrow + E \mid \epsilon \\
Y \rightarrow *T \mid \epsilon
\]

• First sets

Follow ( + ) = \{ \text{int}, ( \} \\
Follow ( ( ) = \{ \text{int}, ( \} \\
Follow ( X ) = \{ \$, ) \} \\
Follow ( ) ) = \{ +, ), $ \} \\
Follow ( \text{int} ) = \{ *, +, ), $ \}

Follow ( * ) = \{ \text{int}, ( \} \\
Follow ( E ) = \{ ), $ \} \\
Follow ( T ) = \{ +, ), $ \} \\
Follow ( Y ) = \{ +, ), $ \}
LR(1) Items

- Consider a context with the item
  \[ E \to E + (\cdot E), + \]
- We expect a string derived from \( E \) \( + \)
- There are two productions for \( E \)
  \[ E \to E + (E) \quad \text{and} \quad E \to \text{int} \]
- We describe this by extending the context with two more items:
  \[ E \to \cdot \text{int}, ) \]
  \[ E \to \cdot E + (E), ) \]
The Closure Operation

- The operation of extending the context with items is called the closure operation.

\[
\text{Closure(Items) = }
\begin{align*}
\text{repeat} \\
\text{for each } [X \rightarrow \alpha \cdot \beta, a] \text{ in Items} \\
\text{for each production } Y \rightarrow \gamma \\
\text{for each } b \in \text{First(\beta a)} \\
\text{add } [Y \rightarrow \cdot \gamma, b] \text{ to Items} \\
\text{until Items is unchanged}
\end{align*}
\]
Construct the Parsing DFA

- Construct the start context: Closure({S → •E, $})
  
  S → •E, $
  E → •E+(E), $
  E → •int, $
  E → •E+(E), +
  E → •int, +

- We abbreviate as:
  
  S → •E, $
  E → •E+(E), $/+ 
  E → •int, $, $/+
Construct the Parsing DFA

• A DFA state is a closed set of LR(1) items
  • This means that we performed Closure
• The start state contains \([S \rightarrow \epsilon, \$$]
• A state that contains \([X \rightarrow \alpha\epsilon, b]\) is labeled with “reduce with \(X \rightarrow \alpha\) on \(b\)”
• And now the transitions...
DFA Transitions

- A state “State” that contains $[X \rightarrow \alpha \cdot \gamma \beta, b]$ has a transition labeled $y$ to a state that contains the items “Transition(State, $\gamma$)”
  - $\gamma$ can be a terminal or a non-terminal

**Transition(State, $\gamma$)**

\[
\text{Items} \leftarrow \emptyset \\
\text{for each } [X \rightarrow \alpha \cdot \gamma \beta, b] \in \text{State} \\
\text{add } [X \rightarrow \alpha \gamma \cdot b, b] \text{ to Items} \\
\text{return } \text{Closure(Items)}
\]
Example Diagram
LR Parsing Tables

• Parsing tables (i.e., the DFA) can be constructed automatically for a CFG.
• But we still need to understand the construction to work with parser generators
  • E.g., they report errors in terms of sets of items
• What kind of errors can we expect?
Shift/Reduce Conflicts

- If a DFA state contains both:
  \[ X \rightarrow \alpha \cdot a\beta, b \] and \[ Y \rightarrow \gamma \cdot, a \]

- Then on input “a” we could either:
  - Shift into state \[ X \rightarrow \alpha a \cdot \beta, b \], or
  - Reduce with \[ Y \rightarrow \gamma \]

- This is called a shift-reduce conflict.
Shift/Reduce Conflicts

• Typically due to ambiguities in the grammar
• Classic example: the dangling else:
  \[ S \to \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid \text{OTHER} \]
• Will have a DFA state containing
  \[
  \begin{align*}
  [S &\to \text{if } E \text{ then } S \text{•}, \quad \text{ else}] \\
  [S &\to \text{if } E \text{ then } S \text{• else } S, \ \times ]
  \end{align*}
  \]
• If “else” follows then we can shift or reduce
• Default (bison, CUP, etc..) is to shift.
  • Default behavior is as needed in this case.
Shift/Reduce Conflicts

• Consider the ambiguous grammar:
  \[ E \rightarrow E + E | E * E | \text{int} \]

• Will have the DFA states containing
  \[ [E \rightarrow E * \bullet E, +] \quad [E \rightarrow E * E\bullet, +] \]
  \[ [E \rightarrow \bullet E + E, +] \quad \Rightarrow^E \quad [E \rightarrow E\bullet + E, +] \]

• Again we have a shift/reduce input on +
• We need to reduce (\* binds more tightly than +)
• Recall solution: declare precedence of + and *
Associativity

- Example: 1 - 2 - 5
- Stack contains “1-2” and the look-ahead is “-”

- Left associative: (1 - 2) - 5
- Right associative: 1 - (2 - 5)
Shift/Reduce Conflicts

- In bison declare precedence and associativity:
  - `%left +` (left associative)
  - `%left *` (left associative, higher precedence)
- Precedence of a rule = that of its last terminal
  - See bison manual (“Conflict-Dependent Precedence”) for ways to override this default
- Bison resolves shift/reduce conflicts with a `shift` if:
  - no precedence declared for either rule or terminal
  - input terminal has higher precedence than the rule
  - the precedences are the same and right associative