Announcements

- HW1 due today
- HW2 will be out tonight
  - Due 2/18
  - LL and LR grammars and parsing

Last Class

- Top-down (LL) parsing
  - LL(1) parsing tables, FIRST, FOLLOW and PREDICT sets
  - Writing an LL(1) grammar
- Bottom-up (LR) parsing
  - Model the LR parser

Today’s Lecture Outline

- Bottom-up (LR) parsing
  - Model of the LR parser
- LR Items
  - Characteristic Finite State Machine (CFSM)
  - SLR(1) parsing table
  - LR parsing variants

Programming Language Syntax

Bottom-up Parsing

Read: Scott, Chapter 2.3.3

id + id*id

Stack | Input | Action
id+id*id | shift id
id | +id*id | reduce by term → id
term | +id*id | reduce by expr → term
expr | +id*id | shift +
expr+ | id*id | shift id
expr+id | *id | reduce by term → id

id + id*id

Stack | Input | Action
expr+term | *id | shift *
expr+term* | id | shift id
expr+term*id | reduce by term → term*id
expr+term | reduce by expr → expr+term
expr | accept, SUCCESS
id + id*id

Sequence of reductions performed by parser

id + id*id
term + id*id
expr + id*id
expr + term

- A rightmost derivation in reverse
- The stack (e.g., expr) concatenated with remaining input (e.g., +id*id) gives a sentential form (expr+id*id) in the rightmost derivation.
- I call valid sentential forms in rightmost derivations right sentential forms.

expr → expr + term | term
term → term * id | id

---

Handle

- A handle
  - If we have a rightmost derivation
    \[ S \Rightarrow \ldots \Rightarrow \alpha Aw \Rightarrow \alpha \beta w \]
    then we say that
    \[ A \rightarrow \beta \] at position \( \alpha \) is a handle of \( \alpha \beta w \)
  - Recall our example \( id + id*id \)

Stack  Input
expr*term  *id  Is expr → expr*term a handle of expr*term at position \( \varepsilon \)?
expr*term  id  Is term → id a handle of expr*term at position expr*term?

---

Question

Consider \( id*id*id \)
Stack  Input
term  *id*id  Is expr → term a handle of term*id*id at position \( \varepsilon \)?

- Answer: No! It brings sentential form term*id*id into expr*id*id which is not derivable in a rightmost derivation (You cannot derive sentential form expr*id*id from expr!)

---

Question

How about
Stack  Input
term*id  *id  Is term → term*id a handle of term*id*id at position \( \varepsilon \)?

- Answer: Yes! It brings sentential form term*id*id into term*id which is clearly derivable: expr → term ⇒ term*id

---

Model of an LR parser

Input: \( a_1 \ldots a_n \$ \)

LR Parser

Stack:

Grammar
Symbol
\( S_0 \)
\( X_0 \)
\( S_1 \)
\ldots
\( S_k \)

Parsing table:

action[\( a_i \)]: Do we shift or reduce?
goto[\( a_i, A \)]: After reduction to nonterminal \( A \), what state is pushed on top of the stack?

...
Model of an LR Parser

- Stack is \((s_0, X_1, s_1, \ldots, X_m, s_m)\), input pointer at \(a_i\)
- \(\text{action}[s_m, a_i] = \text{shift } s\)
  - Push \(a_i\) and state \(s\) on stack: \((s_0, X_1, s_1, \ldots, X_m, s_m, a_i, s)\)
  - Advance input pointer
- \(\text{action}[s_m, a_i] = \text{reduce by } A \rightarrow \beta\)
  - Pop \(\beta\) (i.e., pop \(2|\beta|\) things off the stack - all symbols in \(\beta\) plus all their corresponding states): \((s_0, X_1, s_1, \ldots, X_m-|\beta|, s_m-|\beta|)\)
  - Push \(A\) and goto \([s_m-|\beta|, A] = s\) on top of the stack: \((s_0, X_1, s_1, \ldots, X_m-|\beta|, s_m-|\beta|, A, s)\)

Lecture Outline

- Bottom-up (LR) parsing
  - Model of the LR parser
- LR Items
- Characteristic Finite State Machine
- SLR(1) parsing table
- LR Parsing variants

LR Items

- An LR item is a production with a dot at some position on the right-hand side
  - E.g., \(A \rightarrow \alpha \beta\)
  - We are trying to find an \(A\)
  - We already have seen \(\alpha\) (it is on top of the stack)
  - We are looking for \(\beta\)

<table>
<thead>
<tr>
<th>State 0</th>
<th>State 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{start} \rightarrow \text{expr})</td>
<td>(\text{start} \rightarrow \text{expr})</td>
</tr>
<tr>
<td>(\text{expr} \rightarrow \text{expr} + \text{term}</td>
<td>\text{term} \rightarrow \text{term} \ast \text{id}</td>
</tr>
<tr>
<td>Transition on expr</td>
<td></td>
</tr>
<tr>
<td>(\text{term} \rightarrow \text{term} \ast \text{id})</td>
<td></td>
</tr>
<tr>
<td>(\text{term} \rightarrow \text{id})</td>
<td></td>
</tr>
</tbody>
</table>

Closure of an LR Item

- The closure of an LR item \(A \rightarrow \alpha \beta\) is the set of LR items formed as follows:
  - \(A \rightarrow \alpha \beta\) is in the closure of \(A \rightarrow \alpha \beta\)
  - If the dot is in front of a nonterminal \(B\) for some item in the closure, then all of \(B \rightarrow \gamma_1, B \rightarrow \gamma_2, \ldots\) \(B \rightarrow \gamma_n\) are in the closure (\(B \rightarrow \gamma_1, B \rightarrow \gamma_2, \ldots\) \(B \rightarrow \gamma_n\) are all productions for \(B\))

Example

- Compute closure of \(\text{start} \rightarrow \ast \text{expr}\)
  
  Answer:
  
  \(\text{start} \rightarrow \ast \text{expr}\)
  \(\text{expr} \rightarrow \ast \text{expr} + \text{term}\)
  \(\text{expr} \rightarrow \ast \text{term}\)
  \(\text{term} \rightarrow \ast \text{term} \ast \text{id}\)
  \(\text{term} \rightarrow \ast \text{id}\)

Question

- Compute closure of \(\text{expr} \rightarrow \text{expr} + \ast \text{term}\)
  
  Answer:
  
  \(\text{expr} \rightarrow \ast \text{expr} + \ast \text{term}\)
  \(\text{term} \rightarrow \ast \text{term} \ast \text{id}\)
  \(\text{term} \rightarrow \ast \text{id}\)
Question

Compute closure of \( \text{start} \rightarrow \bullet \text{list} \)

- Answer:
  \[
  \text{start} \rightarrow \bullet \text{list} \\
  \text{list} \rightarrow \bullet \text{prefix} ; \\
  \text{prefix} \rightarrow \bullet \text{prefix} , \text{id} \\
  \text{prefix} \rightarrow \bullet \text{id}
  \]

Example

Construct the collection of sets of LR items with transitions for the above grammar

Lecture Outline

- Bottom-up (LR) parsing
  - Model of the LR parser
    - LR Items
    - Characteristic Finite State Machine
    - SLR(1) parsing table
    - LR parsing variants

Characteristic Finite State Machine (CFSM)

The collection of sets of items with transitions is a DFA. This DFA is one part of the CFSM (we will see the other part shortly). CFSM states are parsing states. Transitions on terminals represent shifts. Transitions on nonterminals represent gotos.

CFSM

- 3, 7 contain only items of kind \( A \rightarrow \alpha \), i.e., reduce items
- 0, 4, 5 contain items of kind \( A \rightarrow \alpha \bullet \beta \), i.e., shift items
- 1, 2, 6 contains both reduce and shift items
When the parser is in state 2:

\[
\begin{align*}
expr & \rightarrow \text{term}^* \\
term & \rightarrow \text{term} \ast \text{id}
\end{align*}
\]

should it reduce by \( expr \rightarrow \text{term} \), or should it shift \( \ast \) continuing to look for \( \ast \text{id} \) ?

Answer: It depends on the lookahead! If what comes next is \(+\) or \(\$$\), then reduce. If it is a \(\ast\), then shift.

---

For every state that contains a reduce item \( A \rightarrow \alpha \ast \), add label “reduce by \( A \rightarrow \alpha \) on FOLLOW(A)”

For example, we add label on state 2:

\[
\begin{align*}
expr & \rightarrow \text{term}^* \\
term & \rightarrow \text{term} \ast \text{id}
\end{align*}
\]

reduce by \( expr \rightarrow \text{term} \) on \( \$$,\ast \).

---

The CFSM has 2 parts
- The collection of sets of LR items with transitions
- The “reduce by” labels

To construct the CFSM for a grammar \( G \)
- First, construct the collection of sets of LR items with transitions
- Second, add the “reduce by” labels

---

Construct the CFSM for above grammar
- First, construct collection of sets of LR items with transitions
- Second, add “reduce by” labels
LR Parsing Variants: LR(0), SLR(1)
- An LR(0) parser does not look at input, 0 lookahead
  - No state in the CFSM can contain both reduce and shift items
- An SLR(1) parser looks at 1 token of lookahead
  - This is the variant we studied in class
  - Resolves certain shift-reduce conflicts by looking ahead at input \( a \) and allowing reduction \( A \to \beta \) only if \( a \) is in the FOLLOW set of \( A \)
  - Cannot resolve shift-reduce conflicts such as \( A \to \beta \) where terminal \( a \) is in FOLLOW(A) and \( A' \to \alpha \beta \) where \( a \) is in FIRST(\( \beta' \))

LR Parsing Variants: LALR(1), LR(1)
- LALR(1)
  - Constructs local, context-sensitive FOLLOW sets and avoids more conflicts than SLR(1)
  - An efficiency hack
  - Most common parsers in practice
- LR(1)
  - Uses a different set of LR items
  - More states in CFSM automaton allows LR(1) to keep paths disjoint

From CSMR to SLR(1) Parsing Table
<table>
<thead>
<tr>
<th>State</th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>$$</th>
<th>expr</th>
<th>term</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>shift 3</td>
<td>1</td>
<td>2</td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>1</td>
<td>shift 4</td>
<td>accept</td>
<td></td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>2</td>
<td>reduce 2</td>
<td>shift 5</td>
<td>reduce 2</td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>3</td>
<td>reduce 4</td>
<td>reduce 4</td>
<td>reduce 4</td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>4</td>
<td>shift 3</td>
<td></td>
<td></td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>5</td>
<td>shift 7</td>
<td></td>
<td></td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>6</td>
<td>reduce 1</td>
<td>shift 5</td>
<td>reduce 1</td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
<tr>
<td>7</td>
<td>reduce 3</td>
<td>reduce 3</td>
<td>reduce 3</td>
<td></td>
<td>expr</td>
<td>term</td>
</tr>
</tbody>
</table>

SLR(1) Parsing Table
Input: An augmented grammar \( G' \) (\( G \) with starting production \( start \to \cdots \))
Output: Functions action and goto for \( G' \)
Construct \( C = (l_i,l_i,\ldots) \) the collection of sets of LR items with transitions
State \( i \) is constructed from \( l_i \). The parsing actions for state \( i \) are
- a) If item \( A \to \alpha \beta \) is in \( i \) and there is a transition from \( i \) to \( j \) on \( a \), then set action[\( a \)] to “shift”
- b) If item \( A \to \alpha \) is in \( i \), then set action[\( a \)] to “reduce by \( A \to \alpha \)” for all terminals \( a \) in FOLLOW(A)
- c) If start \( \to \cdots \cdot \cdot \cdot \cdot \cdot \cdot \cdot \) is in \( i \), then set action[\( \$$\)] to “accept”

The goto transition for state \( i \) are constructed for all nonterminals \( A \) using this rule: if there is transition from \( i \) to \( j \) on \( A \), set goto[\( A \to i \)]

If the table contains no multiply-defined entries, the grammar is said to be SLR(1)
A Hierarchy of Grammars

- LL(0) < LL(1) < LL(k)
- LR(0) < SLR(1) < LALR(1) < LR(1) < LR(k)
- Also, LL(k) < LR(k)

Next class

- Conclude with parsing!
- Logic programming and Prolog