Language Processors (E2.15)

Lecture 11: Parsing V
Bottom up parsing II

Objectives
- To describe the algorithms that construct parsing tables for use in LR parsing
- Introduce LR(0) items
- Define functions closure/goto
- Describe algorithm for set-of-items construction
- Derive the parsing table we used in previous lecture

Parsing table construction (1)

Definitions
- How is the parsing table constructed?
  - SLR, LR, LALR
  - SLR (for Simple LR) is the weakest of the three in terms of grammars for which it succeeds but is the easiest to implement.
- We will need some definitions:
  1. An LR(0) item (or simply item) of a grammar is a production rule augmented with a position marker (a dot) somewhere within its right hand side.
     - For example, the production A->XYZ yields the following four items:
       - A -> .XYZ
       - A -> X.YZ
       - A -> XY.Z
       - A -> XYZ.
  - Intuitively an item indicates how much of a production we have seen at a given point in the parsing process.

Definitions (2)
- A viable prefix is a prefix of a right sentential form which can appear as the stack contents during a shift-reduce parse.
- Observation: the viable prefixes of a grammar form a regular language => they can be recognized by a DFA. Constructing the table is essentially trying to construct a DFA from the grammar.
- Items can be viewed as the states of an NFA recognizing viable prefixes. The states of the SLR parser are grouped sets of items, and the grouping of items into sets is the subset construction algorithm we have seen before (NFA->DFA)
- The augmented grammar G' of a grammar G with a start symbol S, is the grammar with a new start symbol S' and an added production rule S' -> S
  - The purpose of this is to indicate to the parser when it should stop and announce acceptance of input (i.e. acceptance occurs when and only when the parser is about to reduce by S' -> S)

Parsing table construction (2)

Functions (1) – the closure function (1)
- In order to construct the parsing table we will need the following functions:
  1. The closure(I) of a set of items I for a grammar G is the set of items constructed from I using two rules:
     - Initially, every item in I is added in closure(I)
     - If A->α.Bβ in closure(I) and B->γ is a production, then add item B->.γ to I if its not already there. We apply this rule until no more new items can be added to closure(I)
  - Intuitively, A->α.Bβ in closure(I) indicates that, at some points in the parsing process we think we might next see as input a substring derivable from Bβ. If B->γ is a production we also expect we might see a substring derivable from γ at this point. For this reason, we include B->.γ in closure(I)
Parsing table construction (2)
Functions (1) – the closure function (2) - example

For example, in the augmented grammar:

- $E' \rightarrow E$
- $E \rightarrow E + T \mid T$
- $T \rightarrow T \ast F \mid F$
- $F \rightarrow (E) \mid id$

If $I$ is the set of one item $\{E' \rightarrow .E\}$, then closure($I$) contains the items:

- $E' \rightarrow .E$ [starting point]
- $E \rightarrow E + .T$ [since there is an E immediately to the right of a dot, by rule (2) we add the E productions with . at the left end]
- $T \rightarrow T \ast .F$ [similarly now for T since $E \rightarrow .T$]
- $F \rightarrow .(E)$
- $F \rightarrow .id$ [similarly now for F since $T \rightarrow .F$]

 Parsing table construction (2)
Functions (2) – the goto function

5. We will also need the goto function – this is defined as follows:

- For a set of items $I$ and a grammar symbol $X$, the function goto$(I, X)$ is defined as the closure of the set of all items $\{A \rightarrow \alpha X \beta\}$ such that $\{A \rightarrow \alpha X \beta\}$ is in $I$

Example: in the augmented grammar

- $E \rightarrow E$
- $E \rightarrow E + T \mid T$
- $T \rightarrow T \ast F \mid F$
- $F \rightarrow (E) \mid id$

If $I$ is $\{E \rightarrow .E \mid E \rightarrow .E + T\}$, then goto$(I, +)$ is $\{E \rightarrow .E + .T\}$ – we examined the items that have + after the dot ($E \rightarrow .E + T$ is the only one) so we moved the . over the + thus getting $\{E \rightarrow .E + .T\}$ and then took the closure of this.

Parsing table construction (3)
The set-of-items construction (1)

- We also need to compute a special collection of sets of items, $C$, which we will call the canonical LR(0) collection:
  - We start with $C = \{\text{closure}(\{S' \rightarrow .S\})\}$
  - Repeat
    - For each set of items $I$ in $C$ and each grammar symbol $X$ such that goto$(I, X)$ is not empty and not in $C$
    - Do: add goto$(I, X)$ to $C$
  - Until no more sets of items can be added to $C$

Example: for the augmented grammar

$E' \rightarrow E$
$E \rightarrow E + T \mid T$
$T \rightarrow T \ast F \mid F$
$F \rightarrow (E) \mid id$

the canonical set of LR(0) items is:

I0: $E' \rightarrow .E$
I1: $E' \rightarrow E.$
I2: $E \rightarrow T.$
I3: $T \rightarrow F.$
I4: $F \rightarrow (.E)$
I5: $F \rightarrow .id$
I6: $E \rightarrow E+.T$
I7: $T \rightarrow T*.F$
I8: $F \rightarrow (E.)$
I9: $E \rightarrow E+T.$
I10: $T \rightarrow T * F.$
I11: $F \rightarrow (E)$.
The set-of-items construction (2) – example (2)

The set of items we computed gives rise to a DFA recognizing viable prefixes.

Algorithm for constructing the SLR table

1. Construct \( C = \{I_0, I_1, \ldots, I_n\} \), the collection of sets of LR(0) items for the augmented grammar.
2. Each set \( I_i \) constructs a state \( i \). The parsing entries for state \( i \) are determined as follows:
   a) For a terminal \( a \), if \([A \rightarrow \alpha, a] \) is in \( I_i \) and goto\((I_i, a) = I_j\), then set \( \text{action}[i, a] = \text{shift} j \).
   b) If \([A \rightarrow \alpha] \) is in \( I_i \), then set \( \text{action}[i, a] = \text{reduce} A \rightarrow \alpha \) for all \( a \) in \( \text{FOLLOW}(A) \) – here \( A \) may not be \( S' \).
   c) If \([S' \rightarrow S.] \) is in \( I_i \), then set \( \text{action}[i, \$] = \text{accept} \).
3. The goto transitions for state \( i \) are constructed for all nonterminals \( A \) using the rule: if goto\((I_i, A) = I_j\), then goto\([i, A] = j \).
4. All entries not defined by (2) and (3) are made "error".
5. The initial state of the parser is the one constructed from the set of items containing \([S' \rightarrow .] \).

Example (1)
For the augmented grammar \( G' \) (previous slides), and the canonical collection of sets of LR(0) items we have computed:

- First consider the set of items \( I_0 \):
  - Item \( F \rightarrow .(E) \) gives rise to the entry \( \text{action}[0, \$] = \text{shift} 4 \) [algorithm rule 3a]
  - Item \( F \rightarrow .id \) to the entry \( \text{action}[0, \text{id}] = \text{shift} 5 \) [alg. rule 3a]
  - The rest of the items yield no actions.

- Moving on to items in \( I_1 \):
  - The first item yields action\([1, \$] = \text{accept} \) [alg. rule 3d]
  - The second item yields action\([1, \text{\texttt{+}}] = \text{shift} 6 \) [alg. rule 3a]
### Parsing table construction (5)

**Example (3)**

The completed table; all blank entries indicate parsing error

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r6</td>
<td>1s</td>
</tr>
<tr>
<td>2</td>
<td>r7</td>
<td>2s</td>
</tr>
<tr>
<td>3</td>
<td>r8</td>
<td>3s</td>
</tr>
<tr>
<td>4</td>
<td>r9</td>
<td>4s</td>
</tr>
<tr>
<td>5</td>
<td>r10</td>
<td>5s</td>
</tr>
<tr>
<td>6</td>
<td>r11</td>
<td>6s</td>
</tr>
<tr>
<td>7</td>
<td>r12</td>
<td>7s</td>
</tr>
<tr>
<td>8</td>
<td>r13</td>
<td>8s</td>
</tr>
<tr>
<td>9</td>
<td>r14</td>
<td>9s</td>
</tr>
<tr>
<td>10</td>
<td>r15</td>
<td>10s</td>
</tr>
<tr>
<td>11</td>
<td>r16</td>
<td>11s</td>
</tr>
</tbody>
</table>

### Summary

- LR parsing algorithm consults the top of the stack, the input symbol and the corresponding parsing table entry to decide what to do next.
- LR, SLR, Canonical LR, LALR are different methods for constructing the parsing table from a grammar.
- We have seen the basic functions and algorithms that are used in the construction of the parsing tables for SLR.

### Next lecture:
Context Analysis

### Recommended Reading:
- Material in this lecture can be found in sections 4.5-4.7 of Aho et al.
- "Parsing Techniques: A Practical Guide" by Grune and Jacobs, on-line book (see LP support page for links)