Automatic generation of LL(1) parsers
Informatics 2A: Lecture 12

John Longley

School of Informatics
University of Edinburgh
jrl@staffmail.ed.ac.uk

14 October, 2011
We’ve seen that if a grammar $\mathcal{G}$ happens to be LL(1) — i.e. if it admits a parse table — then efficient, deterministic, predictive parsing is possible with the help of a stack.

What’s more, if $\mathcal{G}$ is LL(1), $\mathcal{G}$ is automatically unambiguous.

But how do we tell whether a grammar is LL(1)? And if it is, how can we construct a parse table for it?

For very small grammars, might be able to answer these questions by eye inspection. But for realistic grammars, a systematic method is needed.

In this lecture, we give an algorithmic procedure for answering both questions.
Previous lecture: the \textbf{LL(1) parsing algorithm}, which works on a parse table and a particular input string.

This lecture: algorithm for getting from a grammar $\mathcal{G}$ to a parse table. The algorithm will succeed if $\mathcal{G}$ is LL(1), or fail if it isn’t. As in previous lecture, assume $\mathcal{G}$ has no ‘useless nonterminals’.

Next lecture: ways of getting from a grammar to an equivalent LL(1) grammar. (Not always possible, but work quite often.)
First and Follow sets

The construction of a parse table for a given grammar falls into two parts.

1. For each nonterminal $X$, compute two sets called $First(X)$ and $Follow(X)$, defined as follows:
   - $First(X)$ is the set of all terminals that can appear at the start of a phrase derived from $X$.
     [Convention: if $\epsilon$ can be derived from $X$, also include the special symbol $\epsilon$ in $First(X)$.
   - $Follow(X)$ is the set of all terminals that can appear immediately after $X$ in some sentential form derived from the start symbol $S$.
     [Convention: if $X$ can appear at the end of some such sentential form, also include the special symbol $\$ in $Follow(X)$.

2. Use these $First$ and $Follow$ sets to fill out the parse table.
   We’ll do the latter (easier) stage first.
First and Follow sets: an example

Look again at our grammar for well-matched bracket sequences:

\[ S \rightarrow \epsilon \mid TS \quad T \rightarrow (S) \]

By inspection, we can see that

- \( First(S) = \{ (, \epsilon \} \) because an \( S \) can begin with ( or be empty
- \( First(T) = \{ () \} \) because a \( T \) must begin with (  
- \( Follow(S) = \{ ), $ \} \) because within a complete phrase, an \( S \) can be followed by ) or appear at the end
- \( Follow(T) = \{ (, ), $ \} \) because a \( T \) can be followed by ( or ) or appear at the end

Later we’ll give a systematic method for computing these sets.

Further convention: take \( First(a) = \{ a \} \) for each terminal \( a \).
Filling out the parse table

Once we’ve got these *First* and *Follow* sets, we can fill out the
parse table as follows.

For each production $X \rightarrow \alpha$ of $G$ in turn:

- For each terminal $a$, if $\alpha$ ‘can begin with’ $a$, insert $X \rightarrow \alpha$ in
row $X$, column $a$.

- If $\alpha$ ‘can be empty’, then for each $b \in \text{Follow}(X)$ (where $b$
may be $\$), insert $X \rightarrow \alpha$ in row $X$, column $b$.

If doing this leads to clashes (i.e. two productions fighting for the
same table entry), conclude that the grammar isn’t LL(1).

To explain the phrases in blue, suppose $\alpha = x_1 \ldots x_n$, where the $x_i$
may be terminals or nonterminals.

- $\alpha$ can be empty means $\epsilon \in First(x_i)$ for each $i$.

- $\alpha$ can begin with $a$ means that for some $i$,$\epsilon \in First(x_1), \ldots, First(x_{i-1})$, and $a \in First(x_i)$.
Generating parse tables
Calculating First and Follow sets

Filling out the parse table: example

\[
S \rightarrow \epsilon \mid TS \\
T \rightarrow (S)
\]

First\((S)\) = \{(), \epsilon\} \quad Follow\((S)\) = \{\}, $\}$

First\((T)\) = \{()\} \quad Follow\((T)\) = \{(), ), $\}$

Use this information to fill out the parse table:

- \((S)\) can begin with (, so insert \(T \rightarrow (S)\) in entry for (, T).
- \(T\) can begin with (, so insert \(S \rightarrow TS\) in entry for (, S.
- \(\epsilon\) can be empty, and Follow\((S)\) = \{\}, $\}$, so insert \(S \rightarrow \epsilon\) in entries for ), S and $, S.

This gives the parse table we had in the previous lecture:

<table>
<thead>
<tr>
<th></th>
<th>(</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S)</td>
<td>(S \rightarrow TS)</td>
<td>(S \rightarrow \epsilon)</td>
</tr>
<tr>
<td>(T)</td>
<td>(T \rightarrow (S))</td>
<td>(S \rightarrow \epsilon)</td>
</tr>
</tbody>
</table>
First and Follow sets: preliminary stage

To complete the story, we’d like an algorithm for calculating *First* and *Follow* sets.

Easy first step: compute the set $E$ of nonterminals that ‘can be $\epsilon$’:

1. Start by adding $X$ to $E$ whenever $X \rightarrow \epsilon$ is a production of $G$.
2. If $X \rightarrow Y_1 \ldots Y_m$ is a production and $Y_1, \ldots, Y_m$ are already in $E$, add $X$ to $E$.
3. Repeat step 2 until $E$ stabilizes.

Example: for our grammar of well-matched bracket sequences, we have $E = \{S\}$. 
Calculating First sets: the details

1. Set $\text{First}(a) = \{a\}$ for each $a \in \Sigma$. For each nonterminal $X$, initially set $\text{First}(X)$ to $\{\epsilon\}$ if $X \in E$, or $\emptyset$ otherwise.

2. For each production $X \rightarrow x_1 \ldots x_n$ and each $i \leq n$, if $x_1, \ldots, x_{i-1} \in E$ and $a \in \text{First}(x_i)$, add $a$ to $\text{First}(X)$.

3. Repeat step 2 until all $\text{First}$ sets stabilize.

Example:

- Start with $\text{First}(S) = \{\epsilon\}$, $\text{First}(T) = \emptyset$, etc.
- Consider $T \rightarrow (S)$ with $i = 1$: add ( to $\text{First}(T)$.
- Now consider $S \rightarrow TS$ with $i = 1$: add ( to $\text{First}(S)$.
- That’s all.
Calculating Follow sets: the details

1. Initially set \( \text{Follow}(S) = \{\$\} \) for the start symbol \( S \), and \( \text{Follow}(X) = \emptyset \) for all other nonterminals \( X \).

2. For each production \( X \rightarrow \alpha \), each splitting of \( \alpha \) as \( \beta Yx_1 \ldots x_n \) where \( n \geq 1 \), and each \( i \) with \( x_1, \ldots, x_{i-1} \in E \), add all of \( \text{First}(x_i) \) (excluding \( \epsilon \)) to \( \text{Follow}(Y) \).

3. For each production \( X \rightarrow \alpha \) and each splitting of \( \alpha \) as \( \beta Y \) or \( \beta Yx_1 \ldots x_n \) with \( x_1, \ldots, x_n \in E \), add all of \( \text{Follow}(X) \) to \( \text{Follow}(Y) \).

4. Repeat step 3 until all \( \text{Follow} \) sets stabilize.

Example:

- Start with \( \text{Follow}(S) = \{\$\} \), \( \text{Follow}(T) = \emptyset \).
- Apply step 2 to \( T \rightarrow (S) \) with \( i = 1 \): add \( ) \) to \( \text{Follow}(S) \).
- Apply step 2 to \( S \rightarrow TS \) with \( i = 1 \): add \( ( \) to \( \text{Follow}(T) \).
- Apply step 3 to \( S \rightarrow TS \): add \( ) \) and \( \$ \) to \( \text{Follow}(T) \).
- That’s all.
LL(1) is representative of a bunch of classes of CFGs that are efficiently parseable. E.g. \( \text{LL}1 \subset \text{LALR} \subset \text{LR}(1) \). These involve various tradeoffs of expressive power vs. efficiency/simplicity.

For such languages, a parser can be generated automatically from a suitable grammar. (E.g. for LL(1), just need parse table plus fixed ‘driver’ for the parsing algorithm.)

So we don’t need to write parsers ourselves — just the grammar! (E.g. one can basically define the syntax of Java in about 7 pages of context-free rules.)

This is the principle behind parser generators like yacc (‘yet another compiler compiler’) and java-cup.

Tiger book: Andrew Appel, *Modern Compiler Implementation in (C | Java | ML)*.


Some relevant lecture notes and a tutorial sheet from previous years are available via the Course Schedule webpage.