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

Generating parse tables

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.

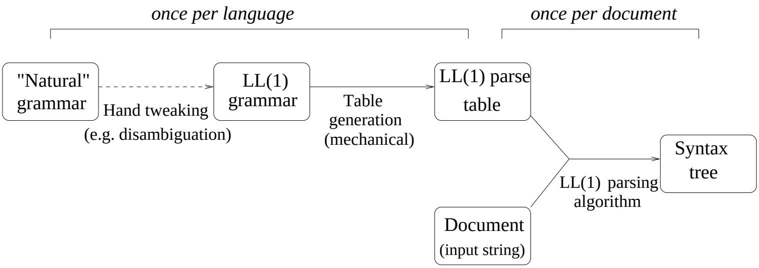
What's more, if $\mathcal{G}$ is LL(1), $\mathcal{G}$ is automatically unambiguous.

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.

The overall picture



Previous lecture: the 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 ϵ can be derived from X, also include the special symbol $\boldsymbol{\epsilon}$ 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:

$$S \rightarrow \epsilon \mid TS \qquad\qquad T \rightarrow (S)$$

By inspection, we can see that

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

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 $X \to \alpha$ of $\mathcal{G}$ in turn:

- For each terminal a, if α 'can begin with' a, insert $X \to \alpha$ in row X, column a.
- If α 'can be empty', then for each $b \in \mathit{Follow}(X)$ (where b may be \$), insert $X \to \alpha$ 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.

- α can be empty means $\epsilon \in \mathit{First}(x_i)$ for each i.
- α can begin with a means that for some i, $\epsilon \in \mathit{First}(x_1), \ldots, \mathit{First}(x_{i-1})$, and $a \in \mathit{First}(x_i)$.

Filling out the parse table: example

$$S \rightarrow \epsilon \mid TS \qquad T \rightarrow (S)$$

$$\begin{aligned} First(S) &= \{(, \epsilon\} & Follow(S) &= \{), \$\} \\ First(T) &= \{(\} & Follow(T) &= \{(,), \$\} \end{aligned}$$

Use this information to fill out the parse table:

- (S) can begin with (, so insert $T \rightarrow (S)$ in entry for $(, T$.
- TS can begin with (, so insert $S \rightarrow TS$ in entry for $(, S$.
- ϵ 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:

	(	)	$
S	$S \rightarrow TS$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$
T	$T \rightarrow (S)$		

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 ϵ':

1. Start by adding X to E whenever $X \to \epsilon$ is a production of $\mathcal{G}$.
2. If $X \to Y_1 \ldots Y_m$ 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 $First(a) = \{a\}$ for each $a \in \Sigma$. For each nonterminal X, initially set $First(X)$ to $\{\epsilon\}$ if $X \in E$, or $\emptyset$ otherwise.
2. For each production $X \to x_1 \ldots x_n$ and each $i \leq n$, if $x_1, \ldots, x_{i-1} \in E$ and $a \in First(x_i)$, add a to $First(X)$.
3. Repeat step 2 until all *First* sets stabilize.

Example:

- Start with $First(S) = \{\epsilon\}$, $First(T) = \emptyset$, etc.
- Consider $T \to (S)$ with $i = 1$: add (to $First(T)$.
- Now consider $S \to TS$ with $i = 1$: add (to $First(S)$.
- That's all.

Calculating Follow sets: the details

1. Initially set $Follow(S) = \{\$\}$ for the start symbol S, and $Follow(X) = \emptyset$ for all other nonterminals X.
2. For each production $X \to \alpha$, each splitting of α as $\beta Y x_1 \ldots x_n$ where $n \geq 1$, and each i with $x_1, \ldots, x_{i-1} \in E$, add all of $First(x_i)$ (excluding ϵ) to $Follow(Y)$.
3. For each production $X \to \alpha$ and each splitting of α as βY or $\beta Y x_1 \ldots x_n$ with $x_1, \ldots, x_n \in E$, add all of $Follow(X)$ to $Follow(Y)$.
4. Repeat step 3 until all *Follow* sets stabilize.

Example:

- Start with $Follow(S) = \{\$\}$, $Follow(T) = \emptyset$.
- Apply step 2 to $T \to (S)$ with $i = 1$: add) to $Follow(S)$.
- Apply step 2 to $S \to TS$ with $i = 1$: add (to $Follow(T)$.
- Apply step 3 to $S \to TS$: add) and \$ to $Follow(T)$.
- That's all.

Parser generators

LL(1) is representative of a bunch of classes of CFGs that are efficiently parseable. E.g. LL1 $\subset$ LALR $\subset$ 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`.

Reading

- Dragon book: Aho, Sethi and Ullman, *Compilers: Principles, Techniques and Tools*, Section 4.4.
- Tiger book: Andrew Appel, *Modern Compiler Implementation in (C | Java | ML)*.
- Turtle book: Aho and Ullman, *Foundations of Computer Science*.
- Some relevant lecture notes and a tutorial sheet from previous years are available via the Course Schedule webpage.