Automatic generation of LL(1) parsers

Informatics 2A: Lecture 12

John Longley

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

14 October 2016

Recap of Lecture 11

- LL(1) predictive parsing reads the input string from left to right, and determines the correct production to apply purely on the basis of two pieces of information: (1) the current input symbol, and (2) the current predicted nonterminal symbol (which is kept on the head of a stack).

- The parsing algorithm is efficient and deterministic and uses a parse table to determine the next production.

- LL(1) parsing is suitable only for formal languages with unambiguous grammars. Even for such languages, finding an LL(1) grammar may require some thought.

  (Addendum: Some formal languages with unambiguous grammars cannot be given an LL(1) grammar at all.)

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.

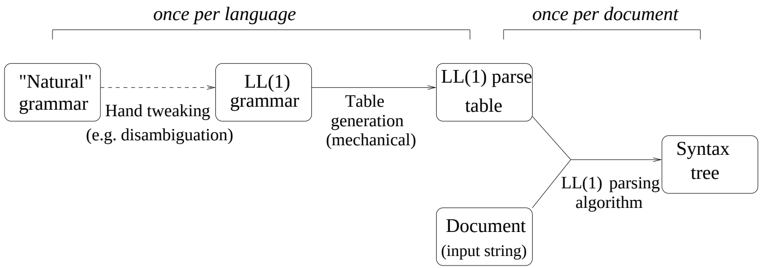
What's more, if $\mathcal{G}$ is LL(1), $\mathcal{G}$ is automatically unambiguous.

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'.)

Lecture 14: ways of getting from a grammar to an equivalent LL(1) grammar. (Not always possible, but work quite often.)

First and Follow sets

Two steps to construct a parse table for a given grammar:

1. For each nonterminal X, compute two sets called $\mathit{First}(X)$ and $\mathit{Follow}(X)$, defined as follows:
    - $\mathit{First}(X)$ 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 ϵ in $\mathit{First}(X)$.]
    - $\mathit{Follow}(X)$ 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 $\mathit{Follow}(X)$.]
2. Use these *First* and *Follow* sets to fill out the parse table.

The first step is somewhat tricky. The second is easier.

Exercises

- $First(X)$ 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 ϵ in $First(X)$.]

Recall our LL(1) grammar for well-matched bracket sequences:

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

Question. Work out each of the two sets below.

1. $First(T)$
2. $First(S)$

Exercises

- $First(X)$ 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 ϵ in $First(X)$.]

Recall our LL(1) grammar for well-matched bracket sequences:

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

Question. Work out each of the two sets below.

1. $First(T)$
2. $First(S)$

Answer: $First(T) = \{(\}$.

Exercises

- $First(X)$ 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 ϵ in $First(X)$.]

Recall our LL(1) grammar for well-matched bracket sequences:

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

Question. Work out each of the two sets below.

1. $First(T)$
2. $First(S)$

Answer: $First(T) = \{(\}$. $First(S) = \{(, \epsilon\}$.

More exercises

- $Follow(X)$ 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 \$ in $Follow(X)$.]

Again consider the same LL(1) grammar:

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

Question. Work out each of the two sets below.

1. $Follow(S)$
2. $Follow(T)$

More exercises

- $Follow(X)$ 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 \$ in $Follow(X)$.]

Again consider the same LL(1) grammar:

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

Question. Work out each of the two sets below.

1. $Follow(S)$
2. $Follow(T)$

Answer: $Follow(S) = \{), \$\}$.

More exercises

- $Follow(X)$ 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 \$ in $Follow(X)$.]

Again consider the same LL(1) grammar:

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

Question. Work out each of the two sets below.

1. $Follow(S)$
2. $Follow(T)$

Answer: $Follow(S) = \{), \$\}$. $Follow(T) = \{(,), \$\}$.

Those examples again

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

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

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 $X \to \alpha$ 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) then conclude that the grammar is not 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 every x_i.
- α can begin with a means that, for some i, $\epsilon \in \mathit{First}(x_1) \cap \ldots \cap \mathit{First}(x_{i-1})$, and $a \in \mathit{First}(x_i)$.

Comments on filling out the parse table

- The case $\alpha = \epsilon$ is counted as a case in which α can be empty.

  (This case is implicit in the last slide since $\alpha = \epsilon$ counts as an instance of $\alpha = x_1 \ldots x_n$ by taking $n = 0$, whence the condition "$\boldsymbol{\epsilon} \in First(x_i)$ for every x_i" is vacuously true since there are no x_i.)

- Similarly, we count $\alpha = x_1 \ldots x_n$ with $a \in First(x_1)$ as one case in which α can begin with a.

  (Again this is implicit in the last slide. The condition $\boldsymbol{\epsilon} \in First(x_1) \cap \ldots \cap First(x_{i-1})$ means that $\boldsymbol{\epsilon}$ is contained in *all* the sets $First(x_1)$, $First(x_2)$ up to $First(x_{i-1})$. In the case that $i = 1$, we consider the sequence $x_1, \ldots, x_{i-1}$ as being empty. Thus the condition "$\boldsymbol{\epsilon} \in First(x_1) \cap \ldots \cap First(x_{i-1})$" is again vacuously true.)

Filling out the parse table: example

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

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

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	$T \rightarrow (S)$		

Intermezzo: true or false?

1. Every LL(1) grammar is context free.
2. Every context-free language can be presented using an LL(1) grammar.
3. Every regular language can be presented using an LL(1) grammar.
4. Every LL(1) grammar is unambiguous.
5. Languages defined by LL(1) grammars can be efficiently parsed.

Intermezzo: true or false?

1. Every LL(1) grammar is context free.
2. Every context-free language can be presented using an LL(1) grammar.
3. Every regular language can be presented using an LL(1) grammar.
4. Every LL(1) grammar is unambiguous.
5. Languages defined by LL(1) grammars can be efficiently parsed.

Answer: 2 is false, the others are all true.

Calculating 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 all $Y_1, \ldots, Y_m$ 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 $X \to x_1 \dots x_n$ and each $i \leq n$, if $x_1, \dots, x_{i-1} \in E$ 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 $\mathit{Follow}(S) = \{\$\}$ for the start symbol S, and $\mathit{Follow}(X) = \emptyset$ 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 $\mathit{First}(x_i)$ (excluding ϵ) to $\mathit{Follow}(Y)$.
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 $\mathit{Follow}(X)$ to $\mathit{Follow}(Y)$.
4. Repeat step 3 until all *Follow* sets stabilize.

Example:

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

Parser generators

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

- Recommended: Some relevant lecture notes ("Note 12" in particular) and a tutorial sheet from previous years are available via the Course Schedule webpage.

- 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*.