

# Automatic generation of LL(1) parsers

## Informatics 2A: Lecture 11

Alex Simpson

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

11 October, 2012

# 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.

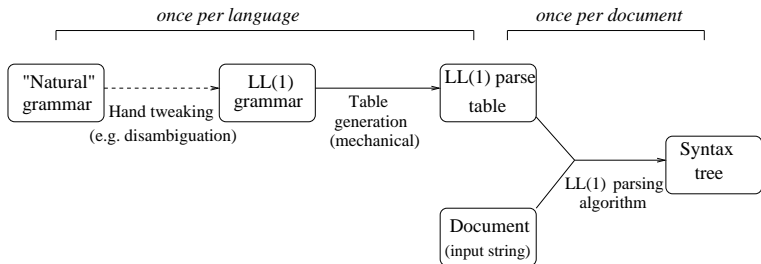
What's more, if  $\mathcal{G}$  is LL(1),  $\mathcal{G}$  is automatically **unambiguous**.

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

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

- 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.

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

## Two clicker questions

- $First(X)$  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  $First(X)$ .]

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

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

Try to work out each of the two sets below.

$First(T)$

$First(S)$

- 1  $\{\epsilon\}$
- 2  $\{( \}$
- 3  $\{(, \epsilon\}$
- 4  $\{(, ), \epsilon\}$

## Two more clicker questions

- $Follow(X)$  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:

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

Try to work out each of the two sets below.

$Follow(S)$

$Follow(T)$

- 1 { \$ }
- 2 { ) }
- 3 { ), \$ }
- 4 { (, ), \$ }

# 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

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

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

**Further convention:** take  $\text{First}(a) = \{a\}$  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  $\mathcal{G}$  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) then **conclude that the grammar is not LL(1)**.

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

- $\alpha$  **can be empty** means  $\epsilon \in \text{First}(x_i)$  for every  $x_i$ .
- $\alpha$  **can begin with**  $a$  means that, for some  $i$ ,  $\epsilon \in \text{First}(x_1) \cap \dots \cap \text{First}(x_{i-1})$ , and  $a \in \text{First}(x_i)$ .



# Comments on filling out the parse table

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

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

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

(Again this is implicit in the last slide. The condition  $\epsilon \in First(x_1) \cap \dots \cap First(x_{i-1})$  means that  $\epsilon$  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  $x_1, \dots, x_{i-1}$  as being empty. Thus the condition “ $\epsilon \in First(x_1) \cap \dots \cap First(x_{i-1})$ ” is again vacuously true. )

## Filling out the parse table: example

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

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

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  $(, S$ .
- $\epsilon$  can be empty, and  $\text{Follow}(S) = \{), \$\}$ , 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: clicker questions

Consider the following statements.

- 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.

How many of the above statements are true?

## Intermezzo: clicker questions

Consider the following statements.

- 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.

How many of the above statements are true?

Which is the unique false statement?

# 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  $\epsilon$ ':

- 1 Start by adding  $X$  to  $E$  whenever  $X \rightarrow \epsilon$  is a production of  $\mathcal{G}$ .
- 2 If  $X \rightarrow Y_1 \dots Y_m$  is a production and  $Y_1, \dots, Y_m$  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  $X \rightarrow x_1 \dots x_n$  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  $T \rightarrow (S)$  with  $i = 1$ : add ( to  $First(T)$ .
- Now consider  $S \rightarrow TS$  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  $X \rightarrow \alpha$ , each splitting of  $\alpha$  as  $\beta Y x_1 \dots x_n$  where  $n \geq 1$ , and each  $i$  with  $x_1, \dots, x_{i-1} \in E$ , add all of  $First(x_i)$  (excluding  $\epsilon$ ) to  $Follow(Y)$ .
- 3 For each production  $X \rightarrow \alpha$  and each splitting of  $\alpha$  as  $\beta Y$  or  $\beta Y x_1 \dots x_n$  with  $x_1, \dots, x_n \in E$ , 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  $T \rightarrow (S)$  with  $i = 1$ : add  $)$  to  $Follow(S)$ .
- Apply step 2 to  $S \rightarrow TS$  with  $i = 1$ : add  $($  to  $Follow(T)$ .
- Apply step 3 to  $S \rightarrow TS$  with  $n = 1$ : 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.  $LL(1) \subset LALR \subset LR(1)$ . 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

- **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.