

# Inf2A: LL(1) Parsing

Stuart Anderson

School of Informatics  
University of Edinburgh

October 29, 2009

# Outline

- 1 Parsing with LL(1) Tables
  - Example
- 2 Calculating First and Follow Sets
  - Definitions
  - Calculating First and Follow Sets
  - Calculating Follow Sets
- 3 Building Parse Tables
  - Example

## The Simplest Case: LL(1) Grammars

- Most of the time if we design our grammar carefully then we can ensure we only require one lookahead symbol to determine which production to choose.
- Care is required in the design of recursive rules.
- No left recursive grammar is LL(k) for any k. (A left recursive rule has the form  $A \rightarrow A\beta$ .)
- Often we design programming languages to ensure one symbol of lookahead is sufficient to determine the construct in use – often this helps ensure a grammar is LL(k)

# Table Driven Parsing

- The grammar  $G_2$  describes a language of nested lists like  $\{x, (x,x), x\}$  and  $((x))$ . Every list must have at least one element, and can use both parentheses  $()$  and braces  $\{\}$  provided they always match.

$$\begin{aligned}
 \textit{term} &\rightarrow \textit{braces} \mid \textit{parens} & \textit{braces} &\rightarrow \{\textit{list}\} & \textit{list} &\rightarrow \textit{term rest} \\
 & & & & & \textit{parens} \rightarrow (\textit{list}) & \textit{list} &\rightarrow x \textit{rest} \\
 \textit{rest} &\rightarrow, \textit{list} \mid \varepsilon
 \end{aligned}$$

# The Parse Table

The parse table for this grammar is:

$G_2$	{	}	(	)	x	,
<i>term</i>	<i>braces</i>		<i>parens</i>			
<i>braces</i>	{ <i>list</i> }					
<i>parens</i>			( <i>list</i> )			
<i>list</i>	<i>term rest</i>		<i>term rest</i>		<i>x rest</i>	
<i>rest</i>		$\epsilon$		$\epsilon$		, <i>list</i>

## Parsing Using the Table

- Suppose we wish to parse the sentence (x) using the grammar  $G_2$ .
- The initial state of the parser is:.

Input	Stack
( x ) \$	<i>term</i> \$

## The procedure for parsing

To parse we repeatedly do the following:

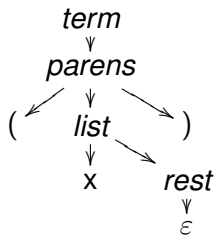
- If top of the stack is a terminal, then it should match the current input symbol.
  - If so, then discard both tokens and proceed.
  - If not, report an error: “Expected ‘a’ but found ‘b’ instead”.
- If the top of the stack is a nonterminal  $A$ , and the current input symbol is  $a$ , then look up the entry  $(A,a)$  in the parse table.
  - This should give an appropriate production  $A \rightarrow t$ : replace  $A$  by  $t$  on top of the stack and proceed.
  - If the entry  $(A,a)$  is empty, signal an error of the form “When trying to process an ‘A’, found ‘a’ and no production applies”.

This process continues for valid input until the \$ token on the input is matched with the \$ symbol on the stack.

# A sample parse

Input	Stack	Productions	Derivation
( x ) \$	<i>term</i> \$	<i>term</i> $\rightarrow$ <i>parens</i>	<i>term</i> $\Rightarrow$ <i>parens</i>
( x ) \$	<i>parens</i> \$	<i>parens</i> $\rightarrow$ ( <i>list</i> )	$\Rightarrow$ ( <i>list</i> )
( x ) \$	( <i>list</i> ) \$		
x ) \$	<i>list</i> ) \$	<i>list</i> $\rightarrow$ x <i>rest</i>	$\Rightarrow$ (x <i>rest</i> )
x ) \$	x <i>rest</i> ) \$		
) \$	<i>rest</i> ) \$	<i>rest</i> $\rightarrow$ $\epsilon$	$\Rightarrow$ (x)
) \$	) \$		
\$	\$		

# The Parse Tree



## Definitions of *First* and *Follow*

- Recall the definitions of  $First_k$  and  $Follow_k$ , here we use  $First$  to stand for  $First_1$  and  $Follow$  to stand for  $Follow_1$ .

### *First*

For any sentential form  $t$ , the set  $First(t)$  is all terminals that appear at the start of some sentential form derived from  $t$ . In addition, if  $t$  derives the empty string, then  $First(t)$  includes  $\epsilon$ .

### *Follow*

For any nonterminal  $A$ , the set  $Follow(A)$  is all the terminals that can appear after  $A$  in any sentential form derived from some nonterminal. Additionally,  $Follow(A)$  contains  $\$$  if  $A$  appears at the end of some sentential form derived from the start symbol.

# First and Follow for $G_2$

$G_2$	<i>First</i>	<i>Follow</i>
<i>term</i>	{ (	), \$
<i>braces</i>	{	), \$
<i>parens</i>	(	), \$
<i>list</i>	{ ( x	)
<i>rest</i>	, $\epsilon$	)

## Calculating *First* sets

To calculate the *First* sets for individual nonterminals we apply the following progressive algorithm.

- Set  $First(A) = \emptyset$ , the empty set, for all nonterminals  $A$ .
- For every rule  $A \rightarrow t$  in the grammar, add  $First(t)$  to  $First(A)$ .
- Repeat until all *First* sets stabilise.

This computation is much easier to carry out by hand if you keep in mind the intuition behind *First* sets: for  $First(A)$ , what might  $A$  begin with?

## Hints on Calculating *First*

We can always work out the *First* set for a sentential form from the *First* sets for its nonterminals.

- The empty sentential form  $\varepsilon$  has a *First* set containing only the empty string.
- A sentential form  $at$  that begins with some terminal  $a$  has just that terminal in its *First* set.
- For a sentential form  $At$  that begins with a nonterminal, look to see if  $First(A)$  contains  $\varepsilon$ ; that is, whether  $A$  can derive the empty string. If not, then  $First(At)$  is simply  $First(A)$  and we are done. If  $A$  may derive the empty string, then for  $First(At)$  take all of  $First(A)$ , remove  $\varepsilon$ , and add in all of  $First(t)$ . If  $t$  is complex, this can mean repeating the whole process; though as  $t$  is shorter than  $At$  we have made progress.

## Calculating *Follow*

*Follow* sets are also calculated progressively, building on the information from *First* sets.

- Begin with  $Follow(S) = \{\$ \}$  for the start symbol and  $Follow(A) = \emptyset$  for all other nonterminals.
- For each rule that can be presented as  $A \rightarrow tBu$  for nonempty  $u$ , add all of  $First(u)$ , except  $\epsilon$ , to  $Follow(B)$ .
- For each rule  $A \rightarrow tB$ , or  $A \rightarrow tBu$  with  $\epsilon \in First(u)$ , add  $Follow(A)$  to  $Follow(B)$ .
- Repeat the last step until all *Follow* sets stabilise.

This is generally rather trickier to carry out than the calculation of *First* sets. Again, bear in mind the intuition behind *Follow* sets: for any nonterminal  $A$ , what might sensibly follow it? Knowing the purpose of a grammar can give a clearer guide than the just studying the details of every production.

# Building Parse Tables

Given the *First* and *Follow* sets for a grammar, constructing a parse table is straightforward. Initially, every entry is empty; then for every production  $A \rightarrow t$  in the grammar:

- for each terminal  $a$  in  $First(t)$ , put  $A \rightarrow t$  in the table at row  $A$ , column  $a$ ;
- if  $\varepsilon \in First(t)$ , then for each  $b \in Follow(A)$  enter  $A \rightarrow t$  in row  $A$ , column  $b$ .

Constructing this table is the final test of suitability for predictive parsing: at the end, there can be at most one production in each cell of the table. If any has two or more, then the grammar is too complex for a predictive parser.

# Final Example

*shell* → command *args*

*args* → *opts files*

*opts* → option *opts* |  $\epsilon$

*files* → file *files* |  $\epsilon$

$G_3$	<i>First</i>	<i>Follow</i>
<i>shell</i>	command	\$
<i>args</i>	option file $\epsilon$	\$
<i>opts</i>	option $\epsilon$	\$ file
<i>files</i>	file $\epsilon$	\$

$G_3$	command	option	file	\$
<i>shell</i>	command <i>args</i>			
<i>args</i>		<i>opts files</i>	<i>opts files</i>	<i>opts files</i>
<i>opts</i>		option <i>opts</i>	$\epsilon$	$\epsilon$
<i>files</i>			file <i>files</i>	$\epsilon$

# Summary

- We have seen how to use a parse table to parse a terminal string derived from some CFG
- We have considered the simple case of calculating the First and Follow sets for an LL(1) parser.
- We have seen how to build a parse table from the grammar and the First and Follow sets.
- Thanks to Ian Stark for earlier versions of this note.