

# Inf2A: Deterministic Parsing

Stuart Anderson

School of Informatics  
University of Edinburgh

October 29, 2009

# Outline

- 1 Parsing
- 2 Getting rid of non-determinism
  - Constructing a PDA from a CFG
  - Example Grammars
  - Grammars for Expressions
- 3 Lookaheads
- 4 Building a Parse Table

# Parsing

- Recall that *parsing* is the task of checking, given a grammar  $G = (N, \Sigma, P, S)$  and a string  $w \in \Sigma^*$ , whether  $w \in L(G)$ .
- There are general techniques that check this provided  $G$  is a CFG. Today we look at efficient parsing techniques that only work for some CFGs that have a particular structure.
- These techniques are “efficient” in the sense that they take time linear in the length of the input.

## Derivations

- Consider the CFG with productions:

$$A \rightarrow [B] \mid a \quad B \rightarrow CB \mid C \quad C \rightarrow \{A\}$$

- We know that if a string can be derived from  $A$  it is derivable by a *leftmost* derivation or by a *rightmost* derivation.
- Leftmost:  $A \Rightarrow [B] \Rightarrow [CB] \Rightarrow [\{A\}B] \Rightarrow [\{a\}B] \Rightarrow [\{a\}C] \Rightarrow [\{a\}\{A\}] \Rightarrow [\{a\}\{a\}]$
- Rightmost:  $A \Rightarrow [B] \Rightarrow [CB] \Rightarrow [CC] \Rightarrow [C\{A\}] \Rightarrow [C\{a\}] \Rightarrow [\{A\}\{a\}] \Rightarrow [\{a\}\{a\}]$
- Top-down parsers construct a leftmost derivation.
- Bottom-up parsers construct a leftmost parse which is a rightmost derivation in reverse.

## Constructing a PDA from a CFG

- Given a CFG  $G = (N, \Sigma, P, S)$  in Griebach normal form
- Construct the PDA  $M = (\{q_0\}, \Sigma, N, \delta, q_0, S, \emptyset)$  which accepts on empty stack. The PDA has transitions:
  - For each production  $A \rightarrow cB_1 \dots B_k$  add the transition  $((q_0, c, A), (q_0, B_1 \dots B_k))$
- As it stands this generates a nondeterministic PDA if any non-terminal has more than one production rule beginning with the same terminal symbol.
- Any grammar for an infinite language must have at least one non-terminal with two rules - why?

## Example Grammars

- Consider the grammars with the following production rules:
  - 1  $S \rightarrow A$,  $A \rightarrow AA \mid a \mid \varepsilon$$
  - 2  $S \rightarrow A$,  $A \rightarrow aAa \mid a \mid \varepsilon$$
  - 3  $S \rightarrow A$,  $A \rightarrow aA \mid \varepsilon$$
- All grammars generate the same language.
- How many parses for  $a^n$ ?
- Why can't we just make the PDA deterministic?
- How do the constructed PDAs behave on  $aaaa$ ?
- What information do we have to predict the correct choice of rule?

# Grammars for Expressions

- $S \rightarrow E \$, E \rightarrow E + E \mid T, T \rightarrow T * T \mid F, F \rightarrow x \mid (E)$
- Not easy to parse but captures the precedence structure on the operators (and is ambiguous on how to parse expressions like  $(x + x + x)$ ).
- Consider the grammar with rules:  $S \rightarrow E \$, E \rightarrow TT_S, T_S \rightarrow +TT_S \mid \varepsilon, T \rightarrow FF_S, F_S \rightarrow *FF_S \mid \varepsilon, F \rightarrow x \mid (E)$
- Do these describe the same language? Why?
- Try parsing  $((x + x) + x * x) \$$  with both grammars
- What information are we using to decide on the choice of production?

# Lookaheads

- Looking at grammars in GNF that generate DPDAs we can see that for any pair of productions  $A \rightarrow a_1\alpha_1$  and  $A \rightarrow a_2\alpha_2$  with the same left-hand side we have  $a_1 \neq a_2$ .
- We say that we are deciding on the choice of production with one symbol of *lookahead* because we look at the next input symbol to choose between productions for the non-terminal.
- The natural generalisation of this is to consider longer lookaheads. We can construct grammars that require  $k$  symbols of lookahead to make the choice of production (consider  $A \rightarrow a^{k-1}b$  and  $A \rightarrow a^k$  for example.)

## Lookaheads ctd

- So far we have only used single symbol lookaheads, but some grammars need more and for some the choice of rule can't be made on the basis of any finite lookahead. What lookahead is required, deterministically to parse the grammar:  $S \rightarrow bbCd \mid Bcc, B \rightarrow bB \mid b, C \rightarrow cC \mid c$

## Constructing Left Derivations

- Consider the CFG with productions:

$$A \rightarrow [B] \mid a \quad B \rightarrow CB \mid C \quad C \rightarrow \{A\}$$

- Consider constructing a leftmost derivation for:  $[\{a\}\{a\}]$
- Is there anywhere we need judgement to decide on the choice of production?

## First<sub>k</sub>

- For a grammar  $G = (N, \Sigma, S, P)$ , we define First<sub>k</sub> as follows:

$$\text{First}_k(\alpha) = \{t \mid \alpha \rightarrow_G^* t \text{ and } \|t\| \leq k \text{ or } \alpha \rightarrow_G^* t\beta \text{ and } \|t\| = k\}$$

So  $t \in \text{First}_k(\alpha)$  if  $t$  can begin a string derived from  $\alpha$  in  $G$ .  
 If  $t$  is shorter than  $k$  symbols then  $\alpha \rightarrow_G^* t$ .

- This definition is not particularly helpful for calculating First<sub>k</sub>( $\alpha$ ). If we define  $\text{First}_k(X) = \bigcup_{\alpha \in X} \text{First}_k(\alpha)$ , we can define First<sub>k</sub>( $A$ ) for each non-terminal  $A$  in  $G$  as follows, for each production  $A \rightarrow A_1 \dots A_m$  where  $A_i \in N \cup \Sigma$ :

$$\text{First}_k(A) = \text{First}_k(\text{First}_k(A_1) \dots \text{First}_k(A_m))$$

then we solve for each non-terminal.

# Lookaheads

- We are interested in finding a criterion for deciding which production to use to expand a non-terminal  $A$  given that we know the next  $k$  symbols of the input.
- One criterion might be to require that for every non-terminal  $A$  with rules  $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$  we require no string in  $\text{First}_k(\alpha_i)$  is a prefix of any string in  $\text{First}_k(\alpha_j)$  if  $i \neq j$  and vice versa.
- Are there problems with this?

## Empty Productions

- Consider the CFG with productions:

$$A \rightarrow [B] \mid a \quad B \rightarrow CB' \quad B' \rightarrow B \mid \varepsilon \quad C \rightarrow \{A\}$$

- Is this an improvement over our earlier grammar? Why?
- What do we do about empty productions?
- For this grammar when would we choose the empty production?

## Follow Sets

- To ease the definition we suppose any grammar is augmented with a new non-terminal  $S'$  with a single production  $S' \rightarrow S\$^k$ , we also assume that  $\$$  is not a terminal symbol of the grammar.
- For a particular grammar  $G = (N, \Sigma, S, P)$  we define  $\text{Follow}_k(A)$ , where  $A$  is a non-terminal as follows:

$$\text{Follow}_k(A) = \{t \in \Sigma^* \mid S' \xrightarrow{*}_G \alpha A \beta, t \in \text{First}_k(\beta)\}$$

- How does this help with the problem we identified?
- We call a grammar LL( $k$ ) if for all non-terminals  $A$  with productions  $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$  there is no string  $t$ ,  $|t| = k$  such that  $t$  is a prefix of a string in  $\text{First}_k(\alpha_i)\text{Follow}_k(A)$  and  $\text{First}_k(\alpha_j)\text{Follow}_k(A)$  where  $i \neq j$ .

## Building a Parse Table for our Grammar

- Consider a revised version of our grammar:

$$A \rightarrow [B] \mid a \quad B \rightarrow CB' \quad B' \rightarrow B \mid \varepsilon \quad C \rightarrow \{A\}$$

- Calculate  $\text{First}_1$  for each nonterminal, and calculate  $\text{Follow}_1$  for each production
- Build the parse table that for each combination of nonterminal and terminal symbol specifies *at most one* production to use to expand the non-terminal
- Try parsing  $[[\{a\}]\{a}]$  using your table.

	[	]	{	}	a
A	[B]				a
B			CB'		
B'		$\varepsilon$	B		
C			{A}		

## First Sets

- Define  $\text{First}_k$  on strings  $t \in \Sigma^*$  first, in the definition assume  $u$  is some string of length  $k$ . Then define for sets of strings  $T \subseteq \Sigma^*$ . Then define  $\text{First}_k(A)$  where  $A$  is a non-terminal symbol with productions  $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$ . Finally we define  $\text{First}_k$  for strings over  $(N \cup \Sigma)^*$

$$\text{First}_k(t) = t, \text{ if } \|t\| \leq k$$

$$\text{First}_k(uw) = u$$

$$\text{First}_k(T) = \{\text{First}_k(t) \mid t \in T\}$$

$$\text{First}_k(A) = \text{First}_k(\alpha_1) \cup \dots \cup \text{First}_k(\alpha_n)$$

$$\text{First}_k(A\alpha) = \text{First}_k(\text{First}_k(A)\text{First}_k(\alpha)), A \in N \cup \Sigma$$

## Calculating $\text{First}_k$



$$S \rightarrow A\$ \quad A \rightarrow [B] \mid a \quad B \rightarrow CB' \quad B' \rightarrow B \mid \varepsilon \quad C \rightarrow \{A\}$$

- Now calculate  $\text{First}_1(X)$  for each nonterminal  $X$ :

$$\begin{aligned} \text{First}_1(S) &= \text{First}_1(A\$) = \text{First}_1(\text{First}_1(A)\text{First}_1(\$)) \\ &= \text{First}_1(\text{First}_1(A)\$) = \{[, a\} \end{aligned}$$

$$\text{First}_1(A) = \text{First}_1([B]) \cup \text{First}_1(a) = \{[, a\}$$

$$\text{First}_1(B) = \text{First}_1(CB') = \text{First}_1(\text{First}_1(C)\text{First}_1(B')) = \{\{\}$$

$$\text{First}_1(B') = \text{First}_1(B) \cup \text{First}_1(\varepsilon) = \{\{\, \varepsilon\}$$

$$\text{First}_1(C) = \text{First}_k(\{A\}) = \{\{\}$$