Monash University
Faculty of Information Technology

Lecture 10
Context Free Grammars

Slides by David Albrecht (2011), modified by Graham Farr (2013).

FIT2014 Theory of Computation

Overview

- Inductive Definitions
- Context Free Grammars
- Parse Trees
- Derivations

Arithmetic Expressions

1. All integers are Arithmetic Expressions
2. If A and B are Arithmetic Expressions, so are:
   (i) A + B
   (ii) A - B
   (iii) A * B
   (iv) A / B
   (v) (A)

Production Rules

	$S \to A$
$AE \to$ integer	$A \to$ integer
$AE \to AE + AE$	$A \to A + A$
$AE \to AE - AE$	$A \to A - A$
$AE \to AE * AE$	$A \to A * A$
$AE \to AE / AE$	$A \to A / A$
$AE \to (AE)$	$A \to (A)$

Backus-Naur Form

(a.k.a. Backus Normal Form)

John Backus (1924-2007)
http://www-history.mcs.st-and.ac.uk/Biographies/Backus.html

$S \to A$

$A \to$ integer | A + A | A - A | A * A | A / A | (A)

Peter Naur (b. 1928)
http://datamuseum.dk/

Historical example: fragment of the BNF of ALGOL 60

4.1. Compound Statements and Blocks

4.1.1. Syntax

⟨unlabelled basic statement⟩ ::= ⟨assignment statement⟩|
  ⟨go to statement⟩|⟨dummy statement⟩|⟨procedure statement⟩
⟨basic statement⟩ ::= ⟨unlabelled basic statement⟩|⟨label⟩:
  ⟨basic statement⟩
⟨unconditional statement⟩ ::= ⟨basic statement⟩|⟨for statement⟩|
  ⟨compound statement⟩|⟨block⟩
⟨statement⟩ ::= ⟨unconditional statement⟩|
  ⟨conditional statement⟩
⟨compound tail⟩ ::= ⟨statement⟩ **end** |⟨statement⟩ ;
  ⟨compound tail⟩
⟨block head⟩ ::= **begin**⟨declaration⟩|⟨block head⟩ ;
  ⟨declaration⟩
⟨unlabelled compound⟩ ::= **begin** ⟨compound tail⟩
⟨unlabelled block⟩ ::= ⟨block head⟩ ; ⟨compound tail⟩
⟨compound statement⟩ ::= ⟨unlabelled compound⟩|
  ⟨label⟩:⟨compound statement⟩
⟨block⟩::=⟨unlabelled block⟩|⟨label⟩:⟨block⟩

from: J. W. Backus *et al.*, *Comm. ACM* **3** (5) (May 1960) 299-314.

Definitions

- A string is in **EQUAL** if it has an equal number of **a**' s and **b**' s.
- E.g.

  {ε, ab, ba, aabb, abab, abba, baba, …}

- An **a-type** string has one more **a** than **b**.
- A **b-type** string has one more **b** than **a**.

Definition of EQUAL

- A string is in **EQUAL** if it is
  - ε, or
  - an, **a** followed by a string of **b-type**, or
  - it is a **b** followed by a string of **a-type**.
- A string is of **a-type** if it is
  - an **a**, or
  - an **a** followed by a string in **EQUAL**, or
  - a **b** followed by two strings of **a-type**.
- A string is of **b-type** if it is
  - an **b**, or
  - a **b** followed by a string in **EQUAL**, or
  - an **a** followed by two strings of **b-type**.

$S \rightarrow \varepsilon$
$S \rightarrow aB$
$S \rightarrow bA$
$A \rightarrow a$
$A \rightarrow aS$
$A \rightarrow bAA$
$B \rightarrow b$
$B \rightarrow bS$
$B \rightarrow aBB$

Production Rules

$S \rightarrow \varepsilon$
$S \rightarrow bA$
$S \rightarrow aB$
$A \rightarrow a$
$A \rightarrow aS$
$A \rightarrow bAA$
$B \rightarrow b$
$B \rightarrow bS$
$B \rightarrow aBB$

Non-terminals

Terminals

A Context Free Grammar consists of:

1. An alphabet
   - The letters are called **terminals**
2. A set of symbols
   - We call these symbols **nonterminals**
   - One of these symbols is the **Start symbol**
   - **S** is often used as the start symbol.
3. A finite set of production rules of the form:

   One nonterminal → finite string of terminals and/or nonterminals

Definition

- The ***language generated*** by a Context Free Grammar (CFG)
  - consists of those strings which can be produced from the start symbol using the production rules.
- A language generated by a CFG is called a **Context Free Language** (CFL).

History

http://www-groups.dcs.st-and.ac.uk/history/Biographies/Panini.html

- Pāṇini (c520BC-c460BC)
  - studied *Sanskrit*
- Noam Chomsky (b. 1928)
  - studied *natural languages*
- John Backus
  - studied *programming languages*

Noam Chomsky, during visit to Australia in 2011 to accept Sydney Peace Prize.

http://www.abc.net.au/news/2011-06-02/noam-chomsky/2741826

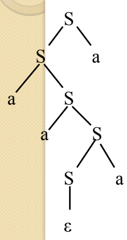
S → aS | Sa | ε

1. S → Sa
2. S → aS
3. S → ε

Derivation of aaaa

S ⇒ Sa (Rule 1)
⇒ aSa (Rule 2)
⇒ aaSa (Rule 2)
⇒ aaSaa (Rule 1)
⇒ aaεaa (Rule 3)
= aaaa

Parse Tree



Derivation of aaaa

S ⇒ Sa (Rule 1)
⇒ aSa (Rule 2)
⇒ aaSa (Rule 2)
⇒ aaSaa (Rule 1)
⇒ aaεaa (Rule 3)
= aaaa

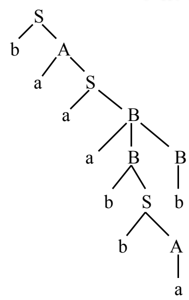
EQUAL

1. S → ε
2. S → bA
3. S → aB
4. A → a
5. A → aS
6. A → bAA
7. B → b
8. B → bS
9. B → aBB

Derivation of baaabbab

S ⇒ bA (Rule 2)
⇒ baS (Rule 5)
⇒ baaB (Rule 3)
⇒ baaaBB (Rule 9)
⇒ baaaBb (Rule 7)
⇒ baaabSb (Rule 8)
⇒ baaabbAb (Rule 2)
⇒ baaabbab (Rule 4)

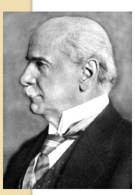
Parse Tree



S ⇒ bA
⇒ baS
⇒ baaB
⇒ baaaBB
⇒ baaaBb
⇒ baaabSb
⇒ baaabbAb
⇒ baaabbab

PARENTHESES (a.k.a. the Dyck Language)

1. S → ε
2. S → (S)
3. S → SS

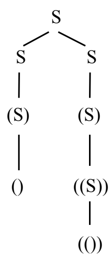


Walther von Dyck (1856-1934)
http://www-history.mcs.st-and.ac.uk/Biographies/Von_Dyck.html

Derivation of ()(())

S ⇒ SS (Rule 3)
⇒ (S)S (Rule 2)
⇒ (S)(S) (Rule 2)
⇒ ()(S) (Rule 1)
⇒ ()((S)) (Rule 2)
⇒ ()(()) (Rule 1)

Parse Tree



S ⇒ SS
⇒ (S)S
⇒ (S)(S)
⇒ ()(S)
⇒ ()((S))
⇒ ()(())

Exercises

- Suppose we have two types of brackets, such as round and square: ( ) and [ ] . Find a context-free language for the set of all valid strings of such brackets.
- Find a context-free grammar for PALINDROMES
- For other languages we have met:
  - find context-free grammars for them, OR
  - if you think they don't have one, think about *why*.

A simple property of derivations

At any stage, the string to the left of the first nonterminal must be a prefix of the final (derived) string.

…….

$x_1 \ldots x_k \mathbf{A}\mathrm{B}\ldots$

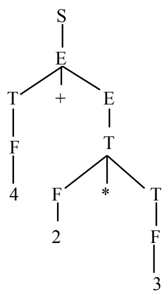
$\Rightarrow x_1 \ldots x_k \mathbf{aXY}\mathrm{B}\ldots$ (using $\mathbf{A} \rightarrow \mathbf{aXY}$)

……

$\Rightarrow x_1 \ldots x_k \mathrm{a}\ldots\ldots$ (derived string)

4 + 2*3

S → E
E → T + E | T - E | T
T → F * T | F / T | F
F → integer | (E)



4 + 2*3

S → E
E → T + E | T - E | T
T → F * T | F / T | F
F → integer | (E)

Leftmost Derivation

$\mathbf{S} \Rightarrow \mathbf{E}$
$\Rightarrow \mathbf{T} + \mathrm{E}$
$\Rightarrow \mathbf{F} + \mathrm{E}$
$\Rightarrow 4 + \mathbf{E}$
$\Rightarrow 4 + \mathbf{T}$
$\Rightarrow 4 + \mathbf{F}*\mathrm{T}$
$\Rightarrow 4 + 2*\mathbf{T}$
$\Rightarrow 4 + 2*\mathbf{F}$
$\Rightarrow 4 + 2*3$

4 + 2*3

S → E
E → T + E | T - E | T
T → F * T | F / T | F
F → integer | (E)

Rightmost Derivation

$\mathbf{S} \Rightarrow \mathbf{E}$
$\Rightarrow \mathrm{T} + \mathbf{E}$
$\Rightarrow \mathrm{T} + \mathbf{T}$
$\Rightarrow \mathrm{T} + \mathrm{F}*\mathbf{T}$
$\Rightarrow \mathrm{T} + \mathrm{F}*\mathbf{F}$
$\Rightarrow \mathrm{T} + \mathbf{F}*3$
$\Rightarrow \mathbf{T} + 2*3$
$\Rightarrow \mathbf{F} + 2*3$
$\Rightarrow 4 + 2*3$

Leftmost and rightmost derivations

- In a **Leftmost derivation**, the leftmost non-terminal is always replaced first.
- In a **Rightmost derivation**, the rightmost non-terminal is always replaced first.

Theorem
Whenever a string has a derivation, it also has a leftmost derivation of the same length.

Proof: see Tute 3.

Does the same hold for rightmost derivations?

A simple property of leftmost derivations

Whenever a production

X → terminals Non-terminal theRest

is applied, the terminal letters on the left are appended to the current prefix to give a larger prefix of the derived string

…….

$x_1 \ldots x_k$**A**B…

⇒ $x_1 \ldots x_k$**aXY**B… (using **A** → **aXY**)

……

⇒ $x_1 \ldots x_k$**a**…… (derived string)

Revision

- Context Free Grammars
  - Definition. How to use them.
- Parse Trees
  - Definition. How to make them.
- Be able to construct leftmost and rightmost derivations.
- Read
  - M. Sipser, , "***Introduction to the Theory of Computation***", Chapter 2.