Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 12

Context-Free Grammars

slides by Graham Farr
based in part on previous slides by David Albrecht

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

$$
\begin{array}{rcl}
AE & \rightarrow & \text{integer} \\
AE & \rightarrow & AE + AE \\
AE & \rightarrow & AE - AE \\
AE & \rightarrow & AE * AE \\
AE & \rightarrow & AE / AE \\
AE & \rightarrow & (AE)
\end{array}
$$

$$
\begin{array}{rcl}
S & \rightarrow & A \\
A & \rightarrow & \text{integer} \\
A & \rightarrow & A + A \\
A & \rightarrow & A - A \\
A & \rightarrow & A * A \\
A & \rightarrow & A / A \\
A & \rightarrow & (A)
\end{array}
$$

Backus-Naur Form (a.k.a. Backus Normal Form)

$$
\begin{aligned}
S &\rightarrow A \\
A &\rightarrow \text{integer} \mid A + A \mid A - A \mid A * A \mid A/A \mid (A)
\end{aligned}
$$

John Backus (1924–2007)
https://mathshistory.st-andrews.ac.uk/Biographies/Backus/

Peter Naur (1928–2016)
https://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.

EQUAL

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

$$\{\varepsilon, \mathtt{ab}, \mathtt{ba}, \mathtt{aabb}, \mathtt{abab}, \mathtt{abba}, \mathtt{baba}, \ldots\}$$

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

EQUAL

A string is in EQUAL if it is

- ε, *or* — $S \longrightarrow \varepsilon$
- a followed by a string of b-type, *or* — $S \longrightarrow \mathtt{a}B$
- b followed by a string of a-type. — $S \longrightarrow \mathtt{b}A$

A string is of a-type if it is

- just a, *or* — $A \longrightarrow \mathtt{a}$
- a followed by a string in EQUAL, *or* — $A \longrightarrow \mathtt{a}S$
- b followed by two strings of a-type. — $A \longrightarrow \mathtt{b}AA$

A string is of b-type if it is

- just b, *or* — $B \longrightarrow \mathtt{b}$
- b followed by a string in EQUAL, *or* — $B \longrightarrow \mathtt{b}S$
- a followed by two strings of b-type. — $B \longrightarrow \mathtt{a}BB$

Context Free Grammar (CFG)

A Context Free Grammar consists of:

1. An alphabet
   - The letters are called **terminals**.
2. Another set of symbols
   - We call these symbols **nonterminals**.
   - often represented by upper-case letters.
   - 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:

$$\text{One nonterminal} \longrightarrow \text{finite string of terminals and/or nonterminals}$$

Context Free Grammar (CFG)

Definition

The **language generated** by a Context Free Grammar (CFG) consists of those *strings of terminals* 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)**.

EQUAL

Production rules:

$$
\begin{aligned}
S &\longrightarrow \varepsilon \\
S &\longrightarrow \mathtt{b}A \\
S &\longrightarrow \mathtt{a}B \\
A &\longrightarrow \mathtt{a} \\
A &\longrightarrow \mathtt{a}S \\
A &\longrightarrow \mathtt{b}AA \\
B &\longrightarrow \mathtt{b} \\
B &\longrightarrow \mathtt{b}S \\
B &\longrightarrow \mathtt{a}BB
\end{aligned}
$$

Terminals: $\mathtt{a}$, $\mathtt{b}$

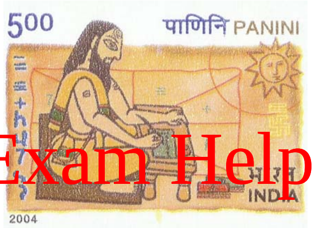
Nonterminals: S, A, B

This CFG generates the language EQUAL.

History

Pāṇini (c520BC–c460BC)

- studied *Sanskrit*



https://mathshistory.st-andrews.ac.uk/Biographies/Panini/

Noam Chomsky (b. 1928)

- studied *natural 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

John Backus

- studied *programming languages*

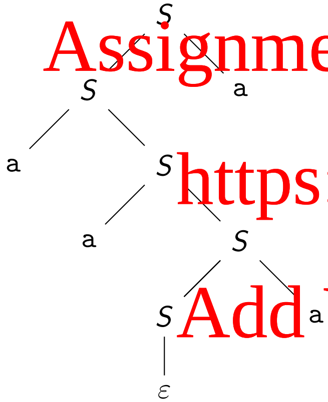
$S \to \mathrm{a}S \mid S\mathrm{a} \mid \varepsilon$

$$
\begin{array}{llll}
1. & S & \to & S\mathrm{a} \\
2. & S & \to & \mathrm{a}S \\
3. & S & \to & \varepsilon
\end{array}
$$

Derivation of aaaa

$$
\begin{array}{rlll}
S & \Rightarrow & S\mathrm{a} & \text{(Rule 1)} \\
 & \Rightarrow & \mathrm{a}S\mathrm{a} & \text{(Rule 2)} \\
 & \Rightarrow & \mathrm{aa}S\mathrm{a} & \text{(Rule 2)} \\
 & \Rightarrow & \mathrm{aa}S\mathrm{aa} & \text{(Rule 1)} \\
 & \Rightarrow & \mathrm{aa}\varepsilon\mathrm{aa} & \text{(Rule 3)} \\
 & = & \mathrm{aaaa} &
\end{array}
$$

Parse Tree



Derivation of aaaa

$$
\begin{array}{rlll}
S & \Rightarrow & S\mathrm{a} & \text{(Rule 1)} \\
 & \Rightarrow & \mathrm{a}S\mathrm{a} & \text{(Rule 2)} \\
 & \Rightarrow & \mathrm{aa}S\mathrm{a} & \text{(Rule 2)} \\
 & \Rightarrow & \mathrm{aa}S\mathrm{aa} & \text{(Rule 1)} \\
 & \Rightarrow & \mathrm{aa}\varepsilon\mathrm{aa} & \text{(Rule 3)} \\
 & = & \mathrm{aaaa} &
\end{array}
$$

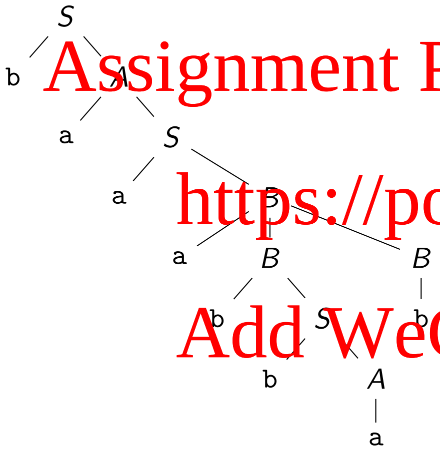
EQUAL

1.	S	$\rightarrow$	ε
2.	S	$\rightarrow$	$\mathrm{b}A$
3.	S	$\rightarrow$	$\mathrm{a}B$
4.	A	$\rightarrow$	a
5.	A	$\rightarrow$	$\mathrm{a}S$
6.	A	$\rightarrow$	$\mathrm{b}AA$
7.	B	$\rightarrow$	b
8.	B	$\rightarrow$	$\mathrm{b}S$
9.	B	$\rightarrow$	$\mathrm{a}BB$

Derivation of baaabbab

$$
\begin{aligned}
S &\Rightarrow \mathrm{b}A && \text{(Rule 2)}\\
&\Rightarrow \mathrm{ba}S && \text{(Rule 5)}\\
&\Rightarrow \mathrm{baa}B && \text{(Rule 3)}\\
&\Rightarrow \mathrm{baaa}BB && \text{(Rule 9)}\\
&\Rightarrow \mathrm{baaa}B\mathrm{b} && \text{(Rule 7)}\\
&\Rightarrow \mathrm{baaab}S\mathrm{b} && \text{(Rule 8)}\\
&\Rightarrow \mathrm{baaabb}A\mathrm{b} && \text{(Rule 2)}\\
&\Rightarrow \mathrm{baaabbab} && \text{(Rule 4)}
\end{aligned}
$$

Parse Tree



Derivation of baaabbab

$$
\begin{aligned}
S &\Rightarrow \mathtt{b}A && \text{(Rule 2)}\\
&\Rightarrow \mathtt{ba}S && \text{(Rule 5)}\\
&\Rightarrow \mathtt{baa}B && \text{(Rule 3)}\\
&\Rightarrow \mathtt{baaa}BB && \text{(Rule 9)}\\
&\Rightarrow \mathtt{baaa}B\mathtt{b} && \text{(Rule 7)}\\
&\Rightarrow \mathtt{baaab}S\mathtt{b} && \text{(Rule 8)}\\
&\Rightarrow \mathtt{baaabb}A\mathtt{b} && \text{(Rule 2)}\\
&\Rightarrow \mathtt{baaabbab} && \text{(Rule 4)}
\end{aligned}
$$

PARENTHESES: the Dyck Language

PARENTHESES is the language over the two-letter alphabet $\{\,(,)\,\}$ consisting of all strings of correctly matched parentheses.

PARENTHESES $= \{\epsilon, (), ()(), (()), ()()(), ()(()), (())(), (()()), ((())), \ldots\}$.

Non-members: $())$ $\qquad$ $((((()))$

Expressing PARENTHESES strings in terms of smaller PARENTHESES strings:

Any non-empty string of parentheses must start with (. Where is its matching) ?

It could be at the other end: $(\underbrace{\cdots\cdots\cdots\cdots}_{\text{smaller PARENTHESES string}})$

It could be before the other end: $\underbrace{(\cdots\cdots\cdots\cdots)}_{\text{smaller PARENTHESES string}}\underbrace{(\cdots\cdots\cdots)}_{\text{smaller PARENTHESES string}}$

PARENTHESES: the Dyck Language

Inductive Definition

Context-Free Grammar

A **string of parentheses** S is one of the following:

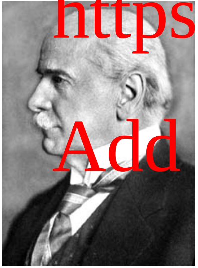
- the empty string, ε
- (S'), where S' is a string of parentheses
- $S_1 S_2$, where S_1, S_2 are strings of parentheses.

1.	S	$\rightarrow$	ε
2.	S	$\rightarrow$	(S)
3.	S	$\rightarrow$	SS

$$
\begin{array}{llll}
1. & S & \to & \varepsilon \\
2. & S & \to & (S) \\
3. & S & \to & SS
\end{array}
$$

Derivation of $()(())$

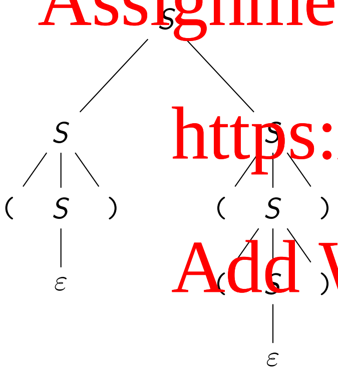
$$
\begin{array}{rll}
S & \Rightarrow SS & \text{(Rule 3)} \\
 & \Rightarrow (S)S & \text{(Rule 2)} \\
 & \Rightarrow (S)(S) & \text{(Rule 2)} \\
 & \Rightarrow ()(S) & \text{(Rule 1)} \\
 & \Rightarrow ()((S)) & \text{(Rule 2)} \\
 & \Rightarrow ()(()) & \text{(Rule 1)}
\end{array}
$$



Walther von Dyck (1856–1934)
https://mathshistory.st-andrews.ac.uk/Biographies/Von_Dyck/

PARENTHESES: the Dyck Language

Parse tree:



Derivation of ()(()):

$$
\begin{array}{rcll}
S & \Rightarrow & SS & \text{(Rule 3)} \\
 & \Rightarrow & (S)S & \text{(Rule 2)} \\
 & \Rightarrow & (S)(S) & \text{(Rule 2)} \\
 & \Rightarrow & ()(S) & \text{(Rule 1)} \\
 & \Rightarrow & ()((S)) & \text{(Rule 2)} \\
 & \Rightarrow & ()(()) & \text{(Rule 1)}
\end{array}
$$

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.

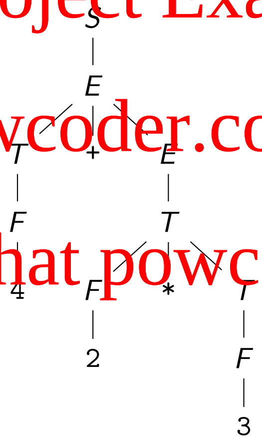
A simple property of derivations

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

$$
\begin{array}{rcll}
S & \Longrightarrow & \cdots & \\
\vdots & & \vdots & \\
& \Longrightarrow & x_1 \cdots x_k AB \cdots & \\
& \Longrightarrow & x_1 \cdots x_k aXYB \cdots & \text{(using } A \longrightarrow aXY\text{)} \\
\vdots & & \vdots & \\
& \Longrightarrow & x_1 \cdots x_k a \cdots\cdots & \text{(derived string)}
\end{array}
$$

`4 + 2*3`

$$
\begin{array}{lcl}
S & \longrightarrow & E \\
E & \longrightarrow & T + E \mid T - E \mid T \\
T & \longrightarrow & F * T \mid F / T \mid F \\
F & \longrightarrow & \text{integer} \mid (E)
\end{array}
$$



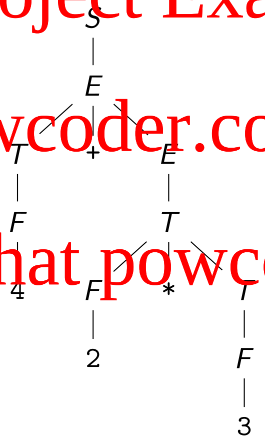
Leftmost derivation:

$$
\begin{array}{lcl}
S & \Rightarrow & \mathbf{E} \\
 & \Rightarrow & \mathbf{T} + E \\
 & \Rightarrow & \mathbf{F} + E \\
 & \Rightarrow & 4 + \mathbf{E} \\
 & \Rightarrow & 4 + \mathbf{T} \\
 & \Rightarrow & 4 + \mathbf{F} * T \\
 & \Rightarrow & 4 + 2 * \mathbf{T} \\
 & \Rightarrow & 4 + 2 * \mathbf{F} \\
 & \Rightarrow & 4 + 2 * 3
\end{array}
$$

`4 + 2*3`

Rightmost derivation:

$$
\begin{array}{lcl}
S & \longrightarrow & E \\
E & \longrightarrow & T + E \mid T - E \mid T \\
T & \longrightarrow & F * T \mid F / T \mid F \\
F & \longrightarrow & \text{integer} \mid (E)
\end{array}
$$



$$
\begin{array}{ll}
S & \Rightarrow \mathbf{E} \\
& \Rightarrow T + \mathbf{E} \\
& \Rightarrow T + \mathbf{T} \\
& \Rightarrow T + F * \mathbf{T} \\
& \Rightarrow T + F * \mathbf{F} \\
& \Rightarrow T + \mathbf{F} * 3 \\
& \Rightarrow \mathbf{T} + 2 * 3 \\
& \Rightarrow \mathbf{F} + 2 * 3 \\
& \Rightarrow 4 + 2 * 3
\end{array}
$$

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 4.

Does the same hold for rightmost derivations?

A simple property of leftmost derivations

Whenever a production

$$X \longrightarrow \text{terminals Non-terminal } \mathit{theRest}$$

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

$$
\begin{array}{lll}
S & \Longrightarrow & \cdots \\
\vdots & & \vdots \\
& \Longrightarrow & x_1 \cdots x_k \underbrace{A} B \cdots \\
& \Longrightarrow & x_1 \cdots x_k \underbrace{aXY} B \cdots \qquad (\text{using} \quad A \longrightarrow aXY) \\
\vdots & & \vdots \\
& \Longrightarrow & x_1 \cdots x_k a \cdots\cdots \qquad (\text{derived string})
\end{array}
$$

Revision

- Context Free Grammars
  - Definition. How to use them.
- Parse Trees
  - Definition. How to make them.
- The Dyck language
- Leftmost and rightmost derivations

Read:
Sipser, Ch. 2, pp. 101–108.