Alphabet Encodings and Formal languages

Prof. Dr. Raphael Volz

Hochschule Pforzheim

Table of Contents

# Alphabets

## Character-encoding schemes

* Interpretation function maps bit sequences to characters
* Function is a typically a bijective mapping table
* Example schemes:
	+ ASCII (American Standard Code for Information Interchange)
	+ Unicode (ISO 10646)
	+ Latin 1 (ISO 8859-1)
* ASCII Example
	+ Uppercase letter **A**
	+ Decimal number **65**
	+ Binary **01000001**

## First 128 symbols in ASCII



[Source: ascii-table.com](http://ascii-table.com/)

## Unicode Basic Multilingual Plane (BMP)



Source: Wikipedia

In the Unicode standard, a plane is a continuous group of 65,536 code points. There are 17 planes, identified by the numbers 0 to 16 decimal. The 17 planes can accommodate 1,114,112 code points, of which 2,048 are surrogates, 66 are non-characters, and 137,468 are reserved for private use, leaving 974,530 for public assignment.

# Grammars

## Formal languages

Exploration on the board. Learning questions:

* What is a terminal ?
* What is a non-terminal ?
* What constitutes a grammar ?
* What is meant by production rule ?

## Avram Noam Chomsky

Father of modern linguistics (Professor emeritus MIT)



[Noam Chomsky in 2004 by Duncan Rawlinson](http://flickr.com/photos/thelastminute/97182354/in/set-72057594061270615/) CC BY 2.0

## Chomsky Hierarchy 101

|  |  |  |
| --- | --- | --- |
| Type | Name | Additional restrictions |
| 0 | Phrase structure grammar | No restrictions on form of production rules |
| 1 | Context-sensitive grammar | Left-hand side shorter than right-hand side for all production rules |
| 2 | Context-free grammar | Left-hand side of production rule is only a variable (non-terminal) |
| 3 | Regular grammar | Right-hand side of production rule is either a terminal or a terminal plus a variable |

## Computational complexity

**Membership problem:**

Given a set of data over $Σ$ does it belong to $L(G)$ ?

|  |  |  |
| --- | --- | --- |
| Type | Membership problem decidable | Complexity |
| 0 | No | Undecidable |
| 1 | Yes | exponential complexity (NP-hard) |
| 2 | Yes | $O(n^{3})$ |
| 3 | Yes | $O(n)$ (linear complexity) |

## Recursion

* Production Rules can be recursive
* Recursion happens when variables appear (indirectly) on left and right-hand side of a production rule
* Often used in practice
* Example: Create a grammar for palindromes



Photo by M Disdero - Taken at Oppede, Luberon, France - CC BY-SA 3.0

## Movie: Grammar of happiness

# EBNF

## John Backus (1924 - 2007)



John Backus

**Turing Award (1977)**

*For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.*

## Peter Naur (1928 - 2016)



Peter Naur

**Turing Award (2005)**

*For fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming.*

## EBNF - Extended Backus-Naur Form

**Meta syntax** (Meta language) for definition of context free grammars

* Definitions are inline of production rules
	+ Terminal symbols (Alphabet)
	+ Non-Terminal symbols (Variables)
* Standard: ISO/IEC 14977:1996(E)
* Extended by Niklaus Wirth (ETH) to create a formal definition of the computer language Pascal

## EBNF Example

twelve = "1", "2";
non-zero-number = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
digit = "0" | non-zero-number ;
natural-number = non-zero-number, { Digit } ;
integer = "0" | [ "-" ], natural-number ;

## EBNF symbols

|  |  |
| --- | --- |
| Usage | Notation |
| definition | = |
| concatenation | , |
| termination | ; |
| alternation | | |
| optional | [ ... ] |
| repetition | { ... } |
| grouping | ( ... ) |
| terminal string | " ... " or ' ... ' |

# Parsers

## Parser

A **parser** is a computer program that

* performs lexical and syntactic analysis
* analyses whether data conforms to a formal grammar
* creates an object representation of the data that can be used within programs
* provides meaningful error messages and reporting
* is mostly generated from a grammar via generators
* is always part of compilers and interpreters that translate computer programs into executable binary code

## JEG.js

[Parser generator written in JavaScript](http://pegjs.org/online)

* Creates a parser program based on a grammar
* Metasyntax goes beyond EBNF
	+ Embeds code fragments into production rules
	+ Binds non-terminals in grammar to variables in code
	+ Embedded code executed while processing data
* Generated parser is itself a JavaScript program
	+ typically downloaded and embedded into own JavaScript programs (and Websites)
	+ executed by the browser (or in other JS environments)

## JEG.js example and exercise

* Example: [Simple grammar for basic arithmetics](http://pegjs.org/online)
* Exercise: Change the grammar to allow division with remainder (modulo) using **%** notation

# Student Evaluation

## Please participate in the questionaire

Wird geladen...