Alphabet Encodings and Formal languages

Prof. Dr. Raphael Volz

Hochschule Pforzheim
Contents

1 Alphabets  
   1.1 Character-encoding schemes ............................ 5 
   1.2 First 128 symbols in ASCII ............................. 6 
   1.3 Unicode Basic Multilingual Plane (BMP) .............. 7 

2 Grammars  
   2.1 Formal languages ........................................ 9 
   2.2 Avram Noam Chomsky ...................................... 9 
   2.3 Chomsky Hierarchy 101 .................................. 10 
   2.4 Computational complexity ............................... 11 
   2.5 Recursion .................................................. 11 
   2.6 Movie: Grammar of happiness ........................... 12 

3 EBNF  
   3.1 John Backus (1924 - 2007) ............................. 13 
   3.2 Peter Naur (1928 - 2016) ............................... 14 
   3.3 EBNF - Extended Backus-Naur Form .................... 14 
   3.4 EBNF Example ............................................. 14 
   3.5 EBNF symbols ............................................. 15 

4 Parsers  
   4.1 Parser ..................................................... 17 
   4.2 JEG.js .................................................... 17 
   4.3 JEG.js example and exercise ............................ 18 

5 Student Evaluation  
   5.1 Please participate in the questionnaire ............... 19
Chapter 1

Alphabets

1.1 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 $01000001$
1.2 First 128 symbols in ASCII

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>space</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>del</td>
</tr>
</tbody>
</table>

Figure 1.1: Source: ascii-table.com
1.3 Unicode Basic Multilingual Plane (BMP)

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.
Chapter 2

Grammars

2.1 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?

2.2 Avram Noam Chomsky

Father of modern linguistics (Professor emeritus MIT)
2.3 Chomsky Hierarchy 101

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Additional restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Phrase</td>
<td>No restrictions on form of production rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grammar</td>
</tr>
</tbody>
</table>
2.5. *RECURSION*

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Additional restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Context-sensit<em>gram</em>mar</td>
<td>Left-hand side shorter than right-hand side for all production rules</td>
</tr>
<tr>
<td>2</td>
<td>Context-free <em>gram</em>mar</td>
<td>Left-hand side of production rule is only a variable (non-terminal)</td>
</tr>
<tr>
<td>3</td>
<td>Regular <em>gram</em>mar</td>
<td>Right-hand side of production rule is either a terminal or a terminal plus a variable</td>
</tr>
</tbody>
</table>

2.4 Computational complexity

Membership problem:

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Membership problem decidable</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
<td>Undecidable</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
<td>exponential complexity (NP-hard)</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>$O(n)$ (linear complexity)</td>
</tr>
</tbody>
</table>

2.5 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
2.6 Movie: Grammar of happiness
Chapter 3

EBNF

3.1 John Backus (1924 - 2007)

Figure 3.1: 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.
3.2 Peter Naur (1928 - 2016)

Figure 3.2: 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.

3.3 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

3.4 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 ;

3.5  **EBNF symbols**

<table>
<thead>
<tr>
<th>Usage</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>definition</td>
<td>=</td>
</tr>
<tr>
<td>concatenation</td>
<td>,</td>
</tr>
<tr>
<td>termination</td>
<td>;</td>
</tr>
<tr>
<td>alternation</td>
<td></td>
</tr>
<tr>
<td>optional</td>
<td>[ ... ]</td>
</tr>
<tr>
<td>repetition</td>
<td>{ ... }</td>
</tr>
<tr>
<td>grouping</td>
<td>( ... )</td>
</tr>
<tr>
<td>terminal string</td>
<td>&quot;...&quot; or ‘...’</td>
</tr>
</tbody>
</table>
Chapter 4

Parsers

4.1 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

4.2 JEG.js

Parser generator written in JavaScript

- 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)
4.3  JEG.js example and exercise

- Example: Simple grammar for basic arithmetics
- Exercise: Change the grammar to allow division with remainder (modulo) using % notation
Chapter 5

Student Evaluation

5.1 Please participate in the questionnaire

Wird geladen...