Alphabet Encodings and Formal languages

Prof. Dr. Raphael Volz

Hochschule Pforzheim

Contents

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

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

Figure 1.1: Source: ascii-table.com

1.3 Unicode Basic Multilingual Plane (BMP)

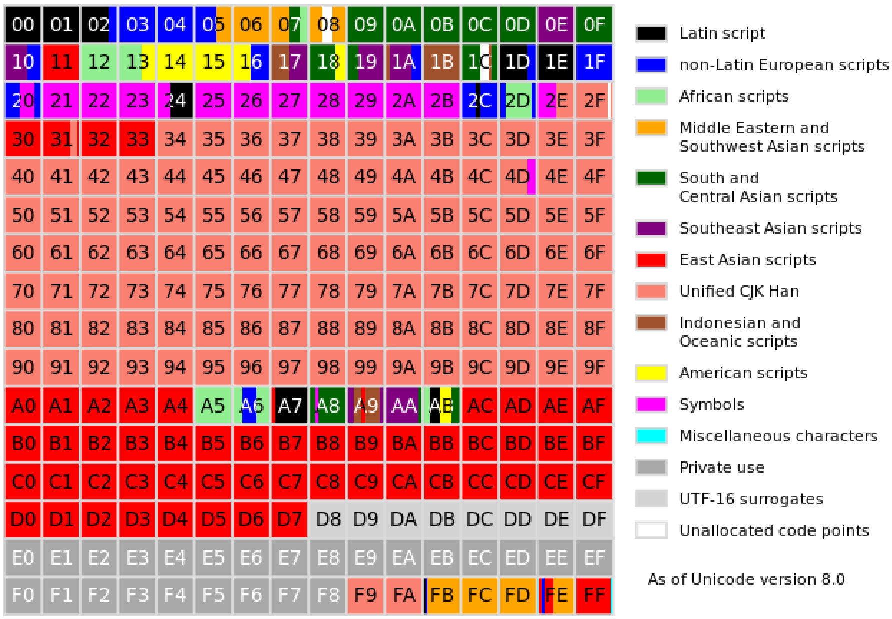

Figure 1.2: Source: Wikipedia

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.

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)

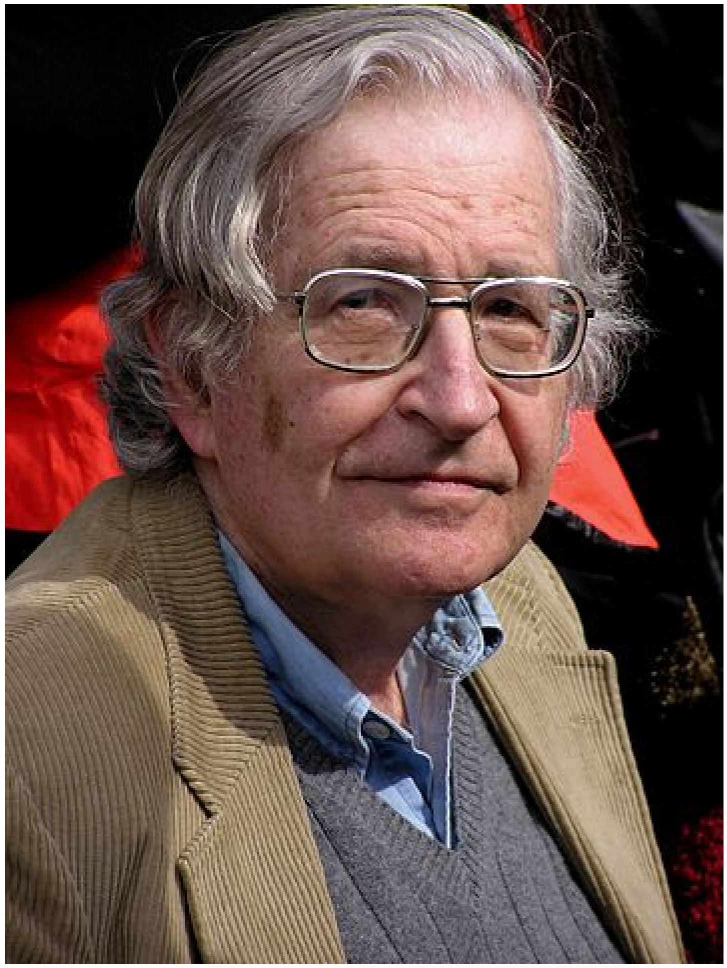

Figure 2.1: Noam Chomsky in 2004 by Duncan Rawlinson CC BY 2.0

2.3 Chomsky Hierarchy 101

Type	Name	Additional restrictions
0	Phrase structure grammar	No restrictions on form of production rules

Type	Name	Additional restrictions
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

2.4 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)

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

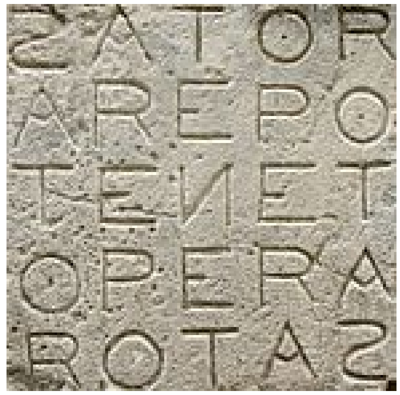

Figure 2.2: Photo by M Disdero - Taken at Oppede, Luberon, France - CC BY-SA 3.0

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

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

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 questionaire

Wird geladen…