DATA FORMATS AND FORMAL LAGUAGES (2/2): CONTEXT FREE GRAMMAR AND BACKUS-NAUR FORM

Matthieu Tixier – #5

Outline

- Formal langages theory
  - Chomsky's hierarchy
  - Context Free Grammar (CFG)
- Applications to software engineering
  - Specify data format and protocols
    - BNF
    - ABNF
- Discussion

Types of formal grammar

- Chomsky hierarchy (1959)
  - Gathers 4 classes of decreasing expressivity (from 0 to 3)
    - For $\gamma \in (N \cup \Sigma)^*$ and ε the empty symbol

Type *Type*	**Nom** *Name*	**Forme des règles** *Constraints on grammar rules*
0	Récursivement énumérable *Recursively enumerable*	$\alpha \rightarrow \gamma$ tel que $\gamma \neq \varepsilon$
1	Sensible au contexte *Context sensitive grammar*	$\alpha A \beta \rightarrow \alpha \gamma \beta$ tel que $\gamma \neq \varepsilon$
2	CFG (« Context Free Grammar ») Grammaire algébrique	$A \rightarrow \gamma$
3	**Grammaire régulière** *Regular grammar*	$\boldsymbol{A \rightarrow \gamma B}$ **ou** $\boldsymbol{A \rightarrow \gamma}$ **avec** $\boldsymbol{\gamma \in \Sigma^*}$

Formal languages theory

Formal grammar definition

L1, L2 ➔ share the same symbols but have a different structure

- $G = (N, \Sigma, S, R)$
  - N : a finite alphabet of **non-terminal** symbols
  - Σ : a finite alphabet of **terminal** symbols
  - S : a particular element of N as start symbol
  - R : A finite set of production rules defined on $(N \cup \Sigma)^*$

➔ How can we provide a minimal description of this difference?

L1 : $S \rightarrow baS$
$S \rightarrow cc$

L2 : $S \rightarrow aSc$
$S \rightarrow b$

Expressions en L1 : $bacc, babacc, bababababacc$ ...

Expressions en L2 : $abc, aabcc, aaabccc$ ...

Regular grammar

- An instance of regular grammar ($A \rightarrow \gamma B$ or $A \rightarrow \gamma$ with $\gamma \in \Sigma^*$)

The grammar of Dates (dd/mm) with D as start symbol:

$D \rightarrow 0J \mid 1J \mid 2J \mid 30/M \mid 31/M$

$J \rightarrow 0/M \mid 1/M \mid 2/M \mid 3/M \mid 4/M \mid 5/M \mid 6/M \mid 7/M \mid 8/M \mid 9/M$

$M \rightarrow 0N \mid 1O$

$N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$ $\quad O \rightarrow 0 \mid 1 \mid 2$

  - As shown above, several non-terminal symbols can be used inside a formal grammar (be it regular or not)
  - Alternative rewriting rules for a same non-terminal symbol can be written with the symbol " | ".

- Limits in term of expressivity
  - For instance:

$$\{a_n b_n\} \text{ (for } n > 0)$$

Types of formal grammar

- Chomsky hierarchy (1959)
  - Gathers 4 classes of decreasing expressivity (from 0 to 3)
    - For $\gamma \in (N \cup \Sigma)^*$ and ε the empty symbol

Type *Type*	**Nom** *Name*	**Forme des règles** *Constraints on grammar rules*
0	Récursivement énumérable *Recursively enumerable*	$\alpha \rightarrow \gamma$ tel que $\gamma \neq \varepsilon$
1	Sensible au contexte *Context sensitive grammar*	$\alpha A \beta \rightarrow \alpha \gamma \beta$ tel que $\gamma \neq \varepsilon$
2	CFG (« Context Free Grammar ») Grammaire algébrique	$A \rightarrow \gamma$
3	**Grammaire régulière** *Regular grammar*	$\boldsymbol{A \rightarrow \gamma B}$ **ou** $\boldsymbol{A \rightarrow \gamma}$ **avec** $\boldsymbol{\gamma \in \Sigma^*}$

Context free languages

- Instances of context free grammar ($A \rightarrow \gamma$)
  - The language of well-formed parenthesis expressions and the famous $\{a_n b_n\}$ (for n > 0)

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow ()$$

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

Context free languages

- Instances of context free grammar ($A \rightarrow \gamma$)
  - The language of well-formed parenthesis expressions and the famous $\{a_n b_n\}$ (for n > 0)

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow ()$$

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

  - Analyse descendante ou ascendante (« parsing »)

➔ produce a « Syntax tree » (Arbre d'analyse)

Top Down : from the start symbol, generate all the possible grammar rules combination that yield to a production of the language.

Bottom Up : from a language production go through from left (or right) to build the combination of rules application that has produced it.

Application for software engineering - BNF

- Top-down parsing example

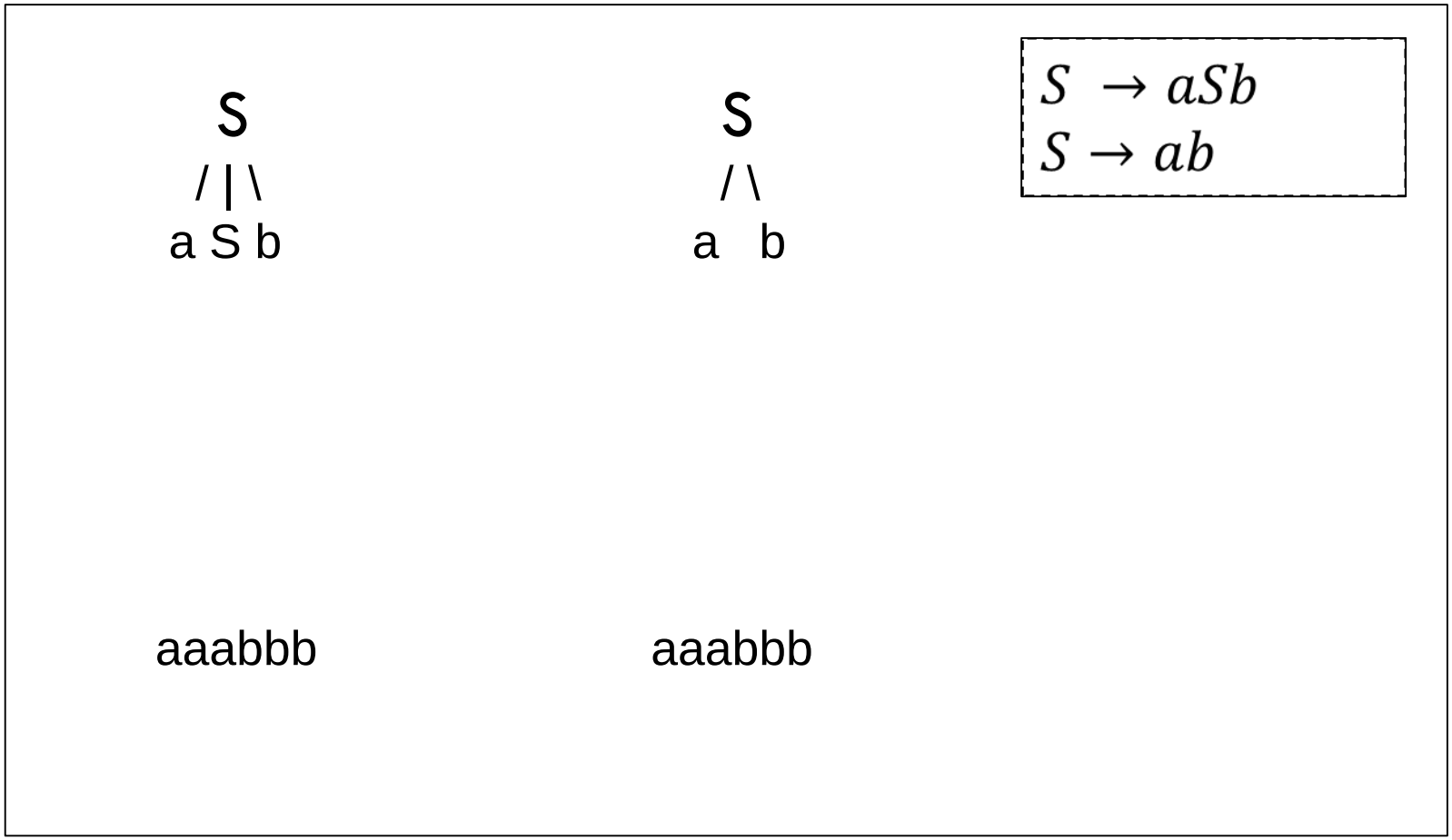
S

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

aaabbb

Application for software engineering - BNF

- Top-down parsing example



Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

S
/ | \
a S b

S
/ \
a b

= ab

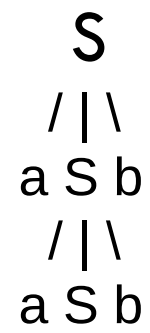
aaabbb

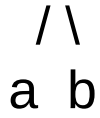
aaabbb

Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$





aaabbb

aaabbb

Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

```
      S
    / | \
   a S b
  / | \
 a S b
```

```
    / \
   a  b
```

= aabb

aaabbb

aaabbb

Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

```
      S
    / | \
   a S b
    / | \
   a S b
    / | \
   a S b

  aaabbb
```

```
    / \
   a   b

  aaabbb
```

Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

```
      S
    / | \
   a S b
    / | \
   a S b
    / | \
   a S b
     ?

  aaabbb
```

```
    / \
   a   b

  aaabbb
```

Application for software engineering - BNF

- Top-down parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

S
/ | \
a S b rule 1
/ | \
a S b rule 1
/ \
a b rule 2

aaabbb

Context free languages

- Instances of context free grammar ($A \rightarrow \gamma$)
  - The language of well-formed parenthesis expressions and the famous $\{a_n b_n\}$ (for n > 0)

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow ()$$

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

  - Analyse descendante ou ascendante (« parsing »)

➔ produce a « Syntax tree » (Arbre d'analyse)

Top Down : from the start symbol, generate all the possible grammar rules combination that yield to a production of the language.

Bottom Up : from a language production go through from left (or right) to build the combination of rules application that has produced it.

Application for software engineering - BNF

- Bottom-up parsing example

S

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

aaabbb

Application for software engineering - BNF

- Bottom-up parsing example

S

$S \rightarrow aSb$
$S \rightarrow ab$

rule1 rule2 ?

[aa]abbb

Application for software engineering - BNF

- Bottom-up parsing example

S

$S \rightarrow aSb$
$S \rightarrow ab$

rule1

rule2

[aa]abbb

Application for software engineering - BNF

- Bottom-up parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

S

```
  S
 | | \
a |  b
 / ?
[aa]abbb
```

rule1

Application for software engineering - BNF

- Bottom-up parsing example

S

$S \rightarrow aSb$
$S \rightarrow ab$

S rule 1 rule2 ?

| | \
a | b
/ ?
[aa]abbb

rule1

Application for software engineering - BNF

- Bottom-up parsing example

S

$S \rightarrow aSb$
$S \rightarrow ab$

S rule 1 rule2 ?

| | \
a | b
/ ?
a[aa]bbb

rule1

Application for software engineering - BNF

- Bottom-up parsing example

S

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

S
| | \
a | b
/ |
a[aa]bbb

rule 1

rule1

rule2

Application for software engineering - BNF

- Bottom-up parsing example

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

S

/ | \
a S b rule 1
| | \
a | b
/ | rule1
a[aa]bbb

Application for software engineering - BNF

- Bottom-up parsing example

$S \rightarrow aSb$
$S \rightarrow ab$

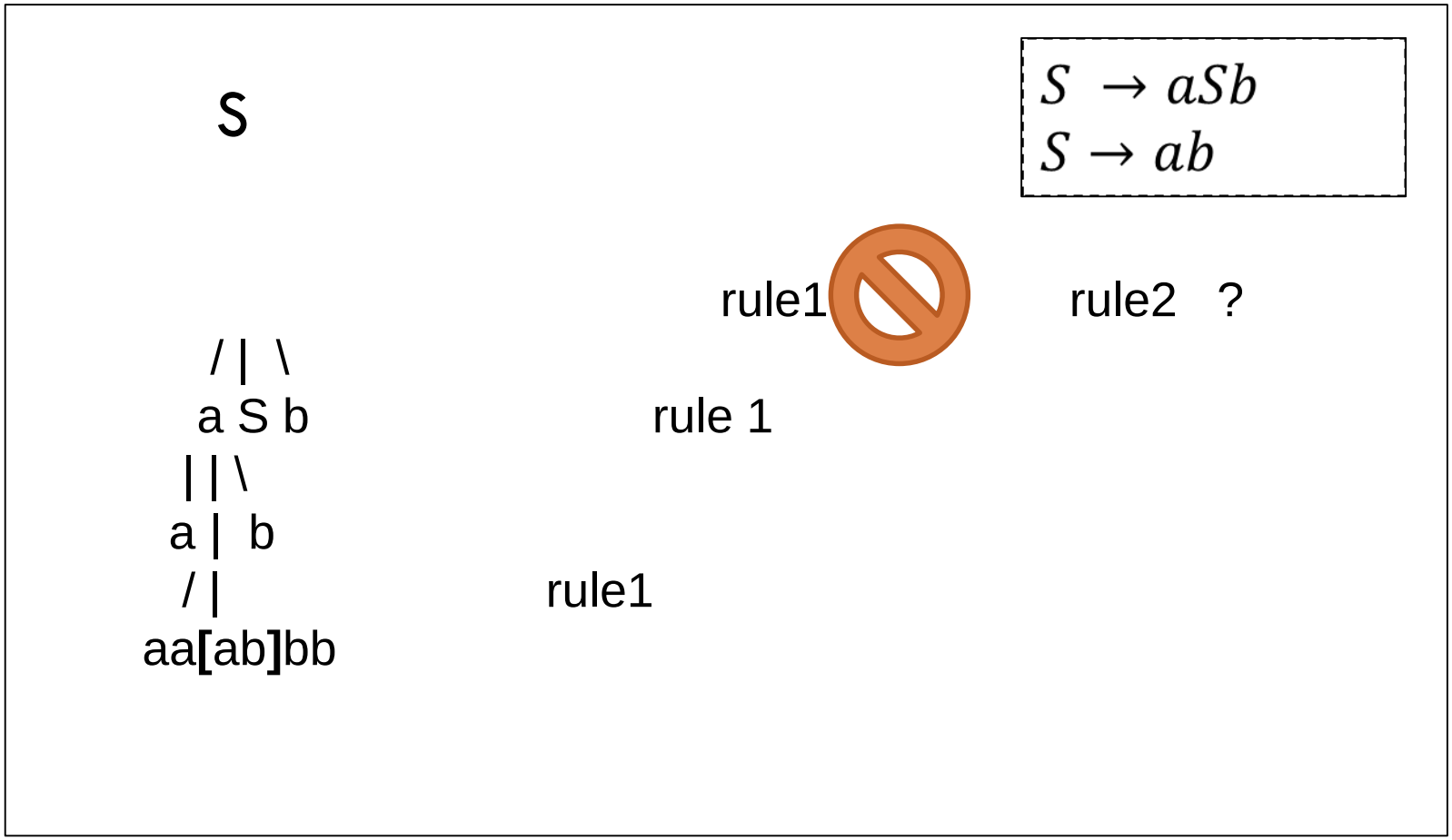
S

rule1 rule2 ?

/ | \
a S b rule 1
| | \
a | b
/ | rule1
aa[ab]bb

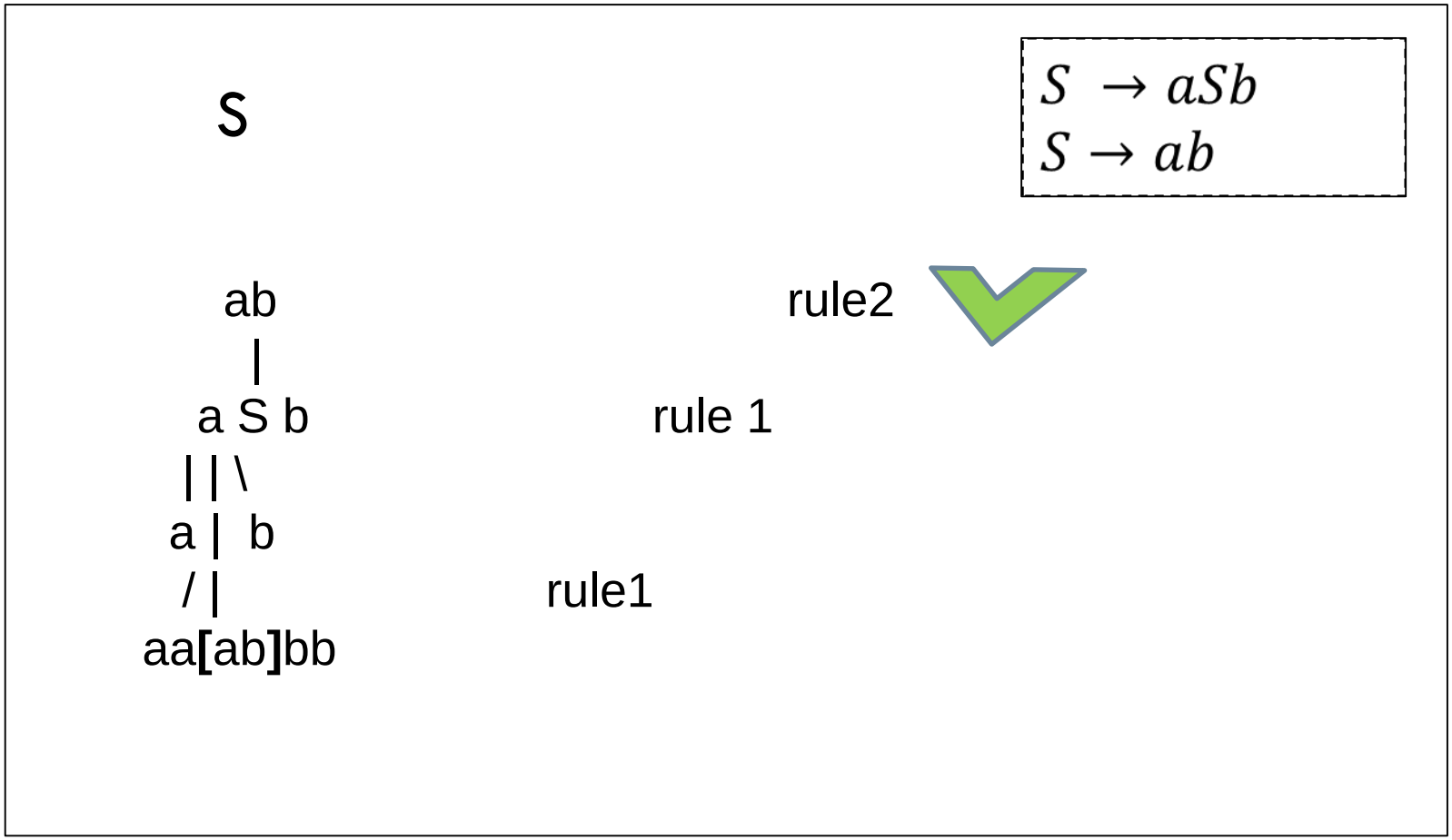
Application for software engineering - BNF

- Bottom-up parsing example



Application for software engineering - BNF

- Bottom-up parsing example



Context free languages

- Instances of context free grammar ($A \rightarrow \gamma$)
  - The language of well-formed parenthesis expressions and the famous $\{a_n b_n\}$ (for n > 0)

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow ()$$

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

Recognized by Pushdown Automata (PDA - automates à pile)

- Has a memory of the automaton last state
- Property
  - From any PDA, you can build an equivalent CFG

- Counter-example : a langage that would allow [(]) is not a CFG

➔ with the same number of opening and closing (and [but allowing crossing involves to take into account the context

Application for software engineering - BNF

- BNF (Backus–Naur Form)
  - Notation and formalism equivalent to CFG

<symbol> ::= __expression__

    - The non terminal symbols are used to be noted with <>
      - The non terminal symbol are virtually all left hand part of rules symbols
    - __expression__
      - One or several symbol sequences (that can be separated by |)
    - ::=
      - The left hand part of the rule can be substituted by the right hand (cf.) →

  - Used to specifiy network protocols and file formats (cf. IETF) as well as the syntax of many programming languages (cf. ALGOL, Javascript, Python …)

Application for software engineering - BNF

- BNF – Simplified postal address example

Dr. Wallace
8, rue Place (appartement : 8)
99880 Yères

M. Untel et Mme. Untel
6, rue de la Plaine
99880 Yères

Application for software engineering - BNF

- BNF – Simplified postal address example

```
<adresse> ::= <identite> <rue> <ville>

<identite> ::= <personne> " " <nom_de_famille> <EOL>
                | <personne> " et " <identite>

<personne> ::= <prenom> | <civilite> "."

<civilite> ::= "M" | "Mme" | "Dr" | "Pr"

<rue> ::= <numero> ", " <nom_de_rue> <opt_num_apt> <EOL>

<ville> ::= <code_postal> " " <nom_ville> <EOL>

<opt_num_apt > ::= " (appartement : " <numero> ")" | ""
```

Application for software engineering - BNF

- BNF – Simplified postal address example

Dr. Wallace
8, rue Place (appartement : 8)
99880 Yères

M. et Mme. Untel
6, rue de la Plaine
99880 Yères

Dr. et M. et Mme et Isabelle et Henri Wallace
8, rue Place (appartement : 8)
99880 Yères

Application for software engineering - BNF

- The BNF syntax can be specified with BNF…

```
<syntax> ::= <rule> | <rule> <syntax>

<rule> ::= <opt-whitespace> "<" <rule-name> ">" <opt-whitespace> "::="
                    <opt-whitespace> <expression> <line-end>

<opt-whitespace> ::= " " <opt-whitespace> | ""
<expression> ::= <list> | <list> "|" <expression>
<line-end> ::= <opt-whitespace> <EOL> | <line-end> <line-end> <list> ::= <term> | <term>
<opt-whitespace> <list>
<term> ::= <literal> | "<" <rule-name> ">"
<literal> ::= ‘ " ‘ <text> ‘ " ‘ | " ‘ " <text> " ‘ "
```

  - Several versions exist: EBNF, ABNF, Python formalism …
    - In GL02 we will use the ABNF that is an Internet standard

Augmented BNF - ABNF

- CFG formalism
  - Used in the Request For Comment (RFC) that specify almost all the Internet major protocols (HTTP, SMTP…)
- RFC5324 – Augmented BNF for Syntax Specifications
  - Supplement RFC7405 - Case-Sensitive String Support in ABNF
  - Several benefits compared with BNF :
    - Precise rules for symbol processing order
    - Repetitions
    - Alternatives
    - Value range
- In what follows the different section number of the RFC5234 are refered by: []

ABNF – Rules definition

- [2.1] Rule Naming
  - **A rule is a non-terminal symbol.**
  - A name is a sequence of alphabet characters, numbers or "-".
  - The "< >" used in BNF are optional (used to refer a rule name outside the grammar definition)
  - Rules name are case insensitive.
    - <rulename>, <Rulename>, <RULENAME>, and <rUlENamE>
  - [3.9] The lines that begin with “;” are comment

- [2.2] Rule form
  - A rule name symbol can be rewrite as a sequence of terminal and non-terminal symbols
  - Formating :
    - Rules are left aligned
    - Several lines long rules have to be indented for readability

règle = éléments crlf
éléments = élément

espace

élément

ABNF – Rules definition

- [2.3] Terminal values
  - All the ASCII adressable character representation
    - Binary (b), Decimal (d), Hexadecimal (x)
    - Ex: CR = %d13, est équivalent à CR = %x0D (carriage return char)
    - In case of concatenation,the symbols are separated by « . » :
      CRLF = %d13.10
  - Literal text strings
    - Enclosed in quotation marks

```
command = "chaine"
```

    - Warning : by default terminal are considered as case insensitive too ("Abc", "aBc", "abC" …)
    - RFC7405
      - %s : prefix a case sensitive string
      - %i prefix a case unsensitive string (by default, rétrocompatibility)
      - Ex. : rulename = %s"aBc"

ABNF – Operators

- [3.1] Concatenation : Rule1 Rule2
  - Space symbol count as concatenation for rule names and lieral strings
    - Ex. the rule <mumble1> match "aba".

```
foo = "a"
bar = "b"
mumble1 = foo bar foo
```

  - Spacing characters have to be precisely expressed

- [3.2] Alternatives : Rule1 / Rule2
  - Similar to « | » in BNF

```
mumble2 = foo / bar
; accept "a" or "b"
```

- [3.3] Incremental Alternatives : Rule1 =/ Rule2
  - Ease the definition and update of ruleset by allowing fragment definition

```
ruleset  = alt1 / alt2
ruleset  =/ alt3
ruleset  =/ alt4 / alt5
```

ABNF – Operators

- [3.4] Value range
  - Defined with « - »

```
DIGIT = %x30-39
```

- [3.5] Sequence Group : (Rule1 Rule2)
  - The symbols enclosed by « () » are treated as one element.
  - Inside () the symbol are strictly ordered.
  - Useful to prevent ambiguity when using « / »

```
rulex1 = el (bar / foo) end
; accept
; <el> "a" <end>,
; or <el> "b" <end>


; is different from
rulex2 = (el bar) / (foo end)
```

ABNF – Operators

- [3.6] Variables repetition : *Rule
  - * repetition between 0 and infinite
  - The complete form is <a>*<b> with a and b respectively minimum and maximum

```
rpt1 = 2*3foo
; accept "aa" or "aaa"
```

- [3.7] Specifc repetition : nRule
  - Repeat exactly n times
  - Equivalent to <n>*<n>

```
rpt2 = 2DIGIT
; accept two numbers from
; 0 à 9, ex : 22, 23, …
```

- [3.8] Optional Sequence : [Rule]
  - With « [] »
  - Equivalent to *1(symb)

ABNF – Operators

- [3.10] Operator Precedence
  - From the highest to the lowest
    - Rule name, Terminal value
    - Comment
    - Value range
    - Repetition
    - Grouping, Optional rule
    - Concatenation
    - Alternative
- [4] Here too, the syntax of ABNF can be expressed with ABNF

ABNF – Remarks

- [B.1] Core Rules (all in uppercase) – very useful

```
ALPHA      =  %x41-5A / %x61-7A
; A-Z / a-z
CHAR       =  %x01-7F                    ;
any 7-bit US-ASCII character,
; excluding NUL
CR         =  %x0D
; carriage return
CRLF       =  CR LF                      ;
Internet standard newline
WSP        =  SP / HTAB
```

```
DIGIT      =  %x30-39
DQUOTE     =  %x22                HEXDIG
           =  DIGIT / "A" / "B" / "C" /
"D" / "E" / "F"
HTAB       =  %x09
LF         =  %x0A
; linefeed
SP         =  %x20
VCHAR      =  %x21-7E
; caractères visibles
```

- [2.4] Character encoding
  - ABNF is defined for the ASCII characters (ie, vs UTF-8)
  - Flexibility for the GL02 project (accents included in VCHAR etc)
    - No need to re-define the « core rules » for UTF-8 or else…

ABNF – Remarks

- [B.1] Core Rules (all in uppercase) – very useful

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x

- ABNF is defined for the ASCII characters (ie, vs UTF-8)
- Flexibility for the GL02 project (accents included in VCHAR etc)
  - No need to re-define the « core rules » for UTF-8 or else…

ABNF – Remarks

- [B.1] Core Rules (all in uppercase) – very useful

```
ALPHA      =  %x41-5A / %x61-7A
; A-Z / a-z
CHAR       =  %x01-7F                    ;
any 7-bit US-ASCII character,
; excluding NUL
CR         =  %x0D
; carriage return
CRLF       =  CR LF                      ;
Internet standard newline
WSP        =  SP / HTAB
```

```
DIGIT      =  %x30-39
DQUOTE     =  %x22                HEXDIG
           =  DIGIT / "A" / "B" / "C" /
"D" / "E" / "F"
HTAB       =  %x09
LF         =  %x0A
; linefeed
SP         =  %x20
VCHAR      =  %x21-7E
; caractères visibles
```

- ~~[2.4] Character encoding~~
  - ABNF is defined for the ASCII characters (ie, vs UTF-8)
  - Flexibility for the GL02 project (accents included in VCHAR etc)
    - No need to re-define the « core rules » for UTF-8 or else…

Examples

- The JSON format – Javascript Object Notation

```
{
  "Image":
    {
      "Width": 800,
      "Height": 600,
      "Title": "View from 15th Floor",
      "Thumbnail": {
              "Url": "http://www.example.com/image/481989943",
              "Height": 125,
              "Width": 100
              },
      "Animated" : false,
      "IDs": [116, 943, 234, 38793]
      }
}
```

Examples

- Excerpt from the ABNF of the JSON format (RFC7159)

```
JSON-text = ws value ws

begin-array = ws %x5B ws
; [ left square bracket
begin-object = ws %x7B ws
; { left curly bracket

end-array = ws %x5D ws
; ] right square bracket
end-object = ws %x7D ws
; } right curly bracket

name-separator = ws %x3A ws
; : colon
```

```
ws = *(
        %x20 /   ; Space
        %x09 /   ; Horizontal tab
        %x0A /   ; Line feed
        %x0D )   ; Carriage return

value = false / null / true / object /
        array / number / string
false = %x66.61.6c.73.65
; false
null = %x6e.75.6c.6c
; null
true = %x74.72.75.65
; true
```

Exemples

- Excerpt from the Python grammar (2.7)

```
test: or_test ['if' or_test 'else' test] | lambdef
or_test: and_test ('or' and_test)*
and_test: not_test ('and' not_test)*
not_test: 'not' not_test | comparison
comparison: expr (comp_op expr)*
comp_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='|'in'|'not' 'in'|'is'|'is' 'not' expr:
xor_expr ('|' xor_expr)*
xor_expr: and_expr ('^' and_expr)*
and_expr: shift_expr ('&' shift_expr)*
shift_expr: arith_expr (('<<'|'>>') arith_expr)*
arith_expr: term (('+'|'-') term)*
term: factor (('*'|'/'|'%'|'//') factor)*
factor: ('+'|'-'|'~') factor | power
power: atom trailer* ['**' factor]
atom: NAME | NUMBER | STRING+ (entres autres)
```

Source : https://docs.python.org/2/reference/grammar.html

Discussion

- Nugget :
  - Different classes of formal languages
  - Relevance of formal grammar for software engineering
  - xBNF can express a CFG to support data formats, protocols and programming languages

- Parsing (« Analyse »)
  - Check the conformity to a grammar (syntax)
  - Build an operable data structure from an analyzed expression (semantic)

References

- Merci de votre attention
  - Question(s) ?
  - TD ~~in normal room~~ at a distance

- Chomsky, N. (1959). "On certain formal properties of grammars". *Information and Control* 2 (2): 137–167
- Backus, J.W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference". *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132.
- IETF (2008), Augmented BNF for Syntax Specifications: ABNF, http://tools.ietf.org/html/rfc5234
- Borges J. L. (1941) La bibliothèque de Babel (tr. « Fictions », Folio)