

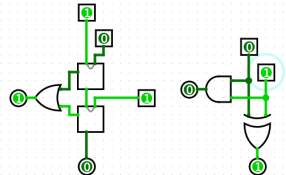
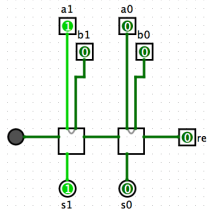
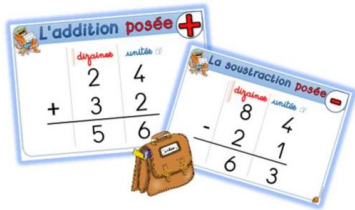
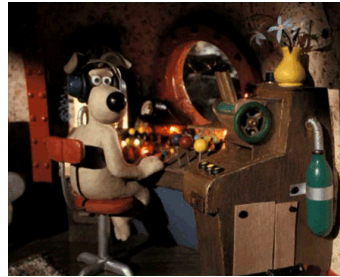
Concepts Informatiques I

2025–2026

Matthieu Picantin



- ♦ systèmes de numération & arithmétique associée
- ♦ **représentation des nombres & arithmétique machine**
- ♦ codes, codages, entropie, compression
- ♦ contrôle d'erreur (détection, correction)
- ♦ crypto (confidentialité, authenticité, intégrité)
- ♦ logique et calcul propositionnel
- ♦ circuits combinatoires



- (si) on manipule rarement les très très très graaaaaaaaands nombres
 (si) on manipule rarement les nombres avec une graaaaande précision

Choix de fixer la taille des représentations

- architectures communes 32 ou 64 bits: arithmétique sur des nombres représentés sur 32 ou 64 bits

$$2^{32} \sim 4,3 \times 10^9 \quad 2^{64} \sim 18,4 \times 10^{18}$$

binary digit

- une infinité de choix pour représenter des entiers!

Un choix parmi les 2^{32} choix pour représenter les entiers de 0 à $2^{32} - 1$

mots sur $\{0, 1\}$ dans l'ordre lexicographique	base 2	base 10
00000000 00000000 00000000 00000000	0	0
00000000 00000000 00000000 00000001	1	1
00000000 00000000 00000000 00000010	10	2
00000000 00000000 00000000 00000011	11	3
...		...
11111111 11111111 11111111 11111110		4 294 967 294
11111111 11111111 11111111 11111111		4 294 967 295

représentation
non-signée

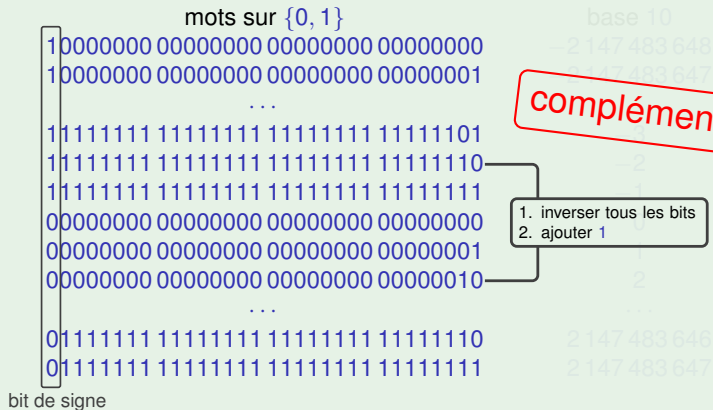
Un parmi les 2^{32} choix pour représenter les entiers de $-2^{31} + 1$ à $2^{31} - 1$

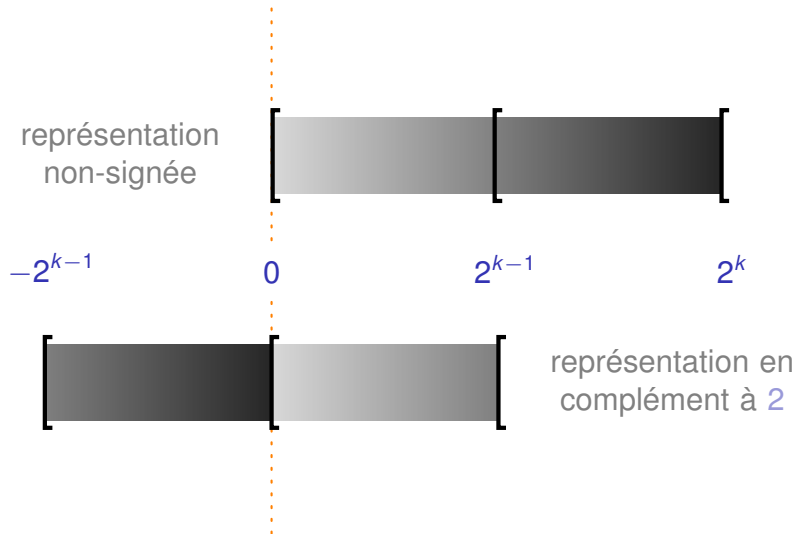
mots sur $\{0, 1\}$	base 10
11111111 11111111 11111111 11111111	-2 147 483 647
11111111 11111111 11111111 11111110	-2 147 483 646
...	...
10000000 00000000 00000000 00000010	-2
10000000 00000000 00000000 00000001	-1
10000000 00000000 00000000 00000000	0
00000000 00000000 00000000 00000000	0
00000000 00000000 00000000 00000001	1
00000000 00000000 00000000 00000010	2
...	...
01111111 11111111 11111111 11111110	2 147 483 646
01111111 11111111 11111111 11111111	2 147 483 647

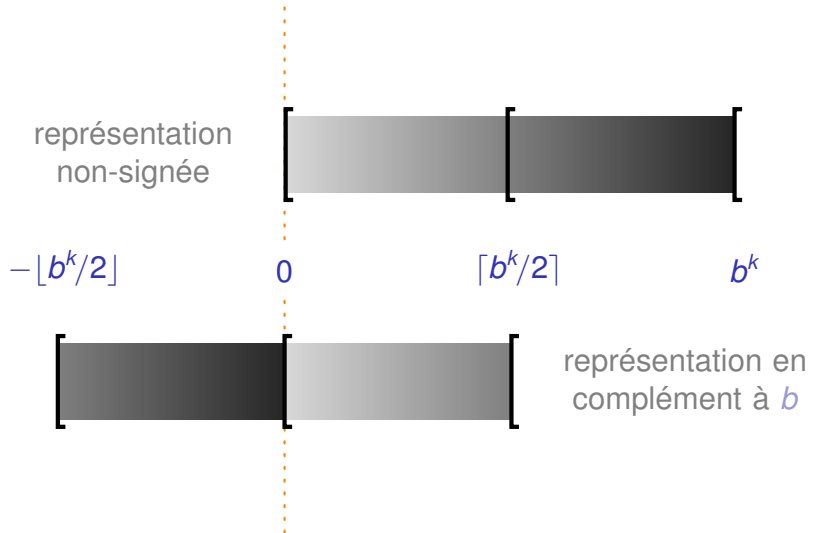
bit de signe

*inadapté
au calcul*

Un parmi les 2^{32} choix pour représenter les entiers de -2^{31} à $2^{31} - 1$







- (si) on manipule rarement les très très très graaaaaaaaands nombres
 (si) on manipule rarement les nombres avec une graaaaande précision

Choix de fixer la taille des représentations

- architectures communes 32 ou 64 bits: arithmétique sur des nombres représentés sur 32 ou 64 bits

$$2^{32} \sim 4,3 \times 10^9 \quad 2^{64} \sim 18,4 \times 10^{18}$$

- une infinité de choix pour représenter des **réels**!

Normes IEEE 754

binary32

1	11000110	10010011110000111000000
s	e	m
↓	↓	↓
$(-1)^s$	$\times 2^{e-B}$	$\times 1 \bullet m$
$(-1)^1$	$\times 2^{198-127}$	$\times 1.10010011110000111000000_2$
environ $-3.7240626 \cdot 10^{21}$		

Normes IEEE 754

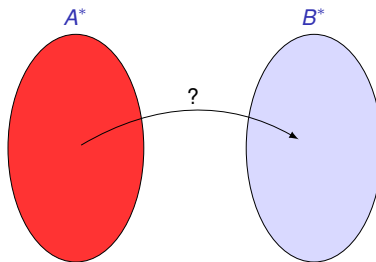
binary32

1	11000110	10010011110000111000000
s	e	m
↓	↓	↓
$(-1)^s$	2^{e-B}	$1 \bullet m$
$(-1)^1$	$2^{198-127}$	$1.10010011110000111000000_2$
environ $-3.7240626 \cdot 10^{21}$		

- ♦ pour $0 < e < e_{max}$, nombre normal de valeur $(-1)^s \times 1.m \times 2^{e-B}$
- ♦ pour $e = 0$, nombre dénormalisé de valeur $(-1)^s \times 0.m \times 2^{1-B}$
- ♦ pour $e = e_{max}$ et $m = 0$, deux valeurs $-\infty$ et $+\infty$
- ♦ pour $e = e_{max}$ et $m \neq 0$, valeur indéterminée NaN (Not-a-Number)

- ♦ mode arrondi au plus près (mode par défaut)
- ♦ mode arrondi vers zéro
- ♦ mode arrondi vers plus l'infini
- ♦ mode arrondi vers moins l'infini

Comment passer d'une représentation par des mots sur un alphabet donné A en une représentation par des mots sur un alphabet donné B ?

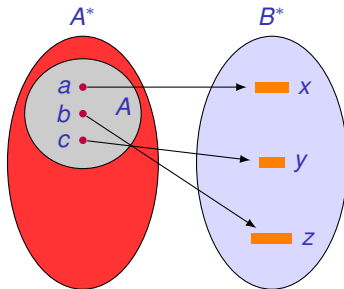


- ♦ représenter des objets dans un système numérique (codage)
- ♦ économiser de l'espace / de la bande passante (compression)
- ♦ résister aux altérations, pertes ou mutations (contrôle d'erreur)
- ♦ rendre illisibles aux non-initiés des données (cryptographie)

Comment passer d'une représentation par des mots sur un alphabet donné A en une représentation par des mots sur un alphabet donné B ?

- ♦ **Alphabet** de n lettres:
 $\Sigma = \{a_0, a_1, a_2, \dots, a_{n-1}\}$
- ♦ **Mot**: suite finie de lettres
 $m = m_0 m_1 \dots m_{\ell-1}$ avec $m_i \in \Sigma$
 - ▶ ℓ : longueur $|m|$ du mot m
- ♦ **Langage** Σ^* : l'ensemble des mots pouvant être écrits sur Σ
 - ▶ tous les mots de longueur 0
 - ▶ tous les mots de longueur 1
 - ▶ tous les mots de longueur 2
 - ▶ ...

- ♦ **Alphabet** de 3 lettres:
 $\{a, b, c\}$
- ♦ **Mot**: par exemple
 $w = aba^3b^2cbac \in \{a, b, c\}^{11}$
 (de longueur $|w| = 11$)
- ♦ **Langage** $\{a, b, c\}^*$: l'ensemble des mots sur l'alphabet $\{a, b, c\}$
 - $\{\epsilon,$
 - $a, b, c,$
 - $aa, ab, ac, ba, bb, bc, ca, cb, cc,$
 - $\dots\}$



On définit un codage des lettres de A en des mots de B^* à l'aide d'une fonction $\tau : A \rightarrow B^*$.

Le **codage** par τ d'un mot $m = m_0 m_1 \cdots m_{\ell-1}$ de A^* consiste alors à coder chaque lettre et à concaténer les codages dans l'ordre :

$$\tau(m) = \tau(m_0)\tau(m_1) \cdots \tau(m_{\ell-1}).$$

(τ est alors un **morphisme** de A^* dans B^*)

On doit pouvoir retrouver le mot original: le morphisme τ doit être **injectif** !

Autrement dit, l'ensemble $\mathcal{C} = \tau(A) = \{ \text{orange}, \text{orange}, \text{orange} \}$ doit être un **code**:
tout mot de \mathcal{C}^* doit se décomposer d'une seule façon sur \mathcal{C} .

Exemples avec $A = \{a, b, c\}$ et $B = \{0, 1\}$

$$\begin{cases} \tau_1(a) = 00 \\ \tau_1(b) = 11 \\ \tau_1(c) = 111110 \end{cases} \quad \begin{array}{l} \tau_1 \text{ définit bien} \\ \text{un codage injectif} \end{array}$$

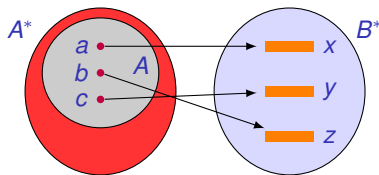
$$\begin{cases} \tau_2(a) = 0 \\ \tau_2(b) = 01 \\ \tau_2(c) = 10 \end{cases} \quad \begin{array}{l} \tau_2 \text{ définit} \\ \text{un codage} \\ \text{non injectif} \end{array}$$

$\Leftrightarrow \mathcal{C}_1 = \{00, 11, 111110\}$ est un **code**

quid de 010: est-ce 0 10 ou 01 0 ?

- Si les images des lettres de A par τ sont toutes d'une même longueur k , le code $\tau(A)$ est dit de **longueur fixe**

$$\exists k : \tau(A) \subseteq B^k$$



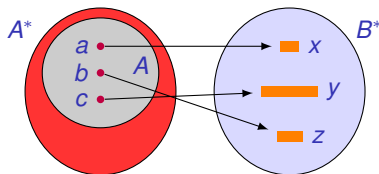
- Il faut/suffit d'avoir $k \geq \log_{|B|}(|A|)$ pour coder A sur B^* (puis coder A^* sur B^*)

- $A = \{a, b, c\}$ et $B = \{0, 1\}$ donnent $k \geq 2$, par exemple $\begin{cases} \tau_3(a) = 00 \\ \tau_3(b) = 01 \\ \tau_3(c) = 10 \end{cases}$
- $A = \{a, b, \dots, z\}$ et $B = \{0, 1\}$ donnent $k \geq 5$, par exemple $\begin{cases} \tau_4(a) = 00000 \\ \vdots \\ \tau_4(z) = 11001 \end{cases}$
- $A = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9\}$ et $B = \{0, 1\}$ donnent $k \geq 6$, etc.

- Le décodage est facile à partir du découpage du mot image en blocs de k lettres

- $0100001000 = 01\ 00\ 00\ 10\ 00$ est le codage par τ_3 de *baaca*
- $0100001000 = 01000\ 01000$ est le codage par τ_4 de *hh*

- ♦ Si les images des lettres de A par τ ne sont pas toutes de même longueur, le code $\tau(A)$ est de **longueur variable**



- ▶ utiles si la fréquence des symboles de A n'est pas uniforme
- ▶ en général plus difficiles à construire et/ou à décoder
- ♦ Parmi eux, les codes préfixes sont les plus faciles à construire et à décoder

Un langage **préfixe** est un langage dans lequel aucun mot n'est le préfixe d'un autre mot

Tout langage préfixe est un code!

$\{0, 10, 110, 1110\}$ est un exemple de code préfixe

$\{0, 01, 011, 0111\}$ n'est pas préfixe (mais c'est un code suffixe!)

Exemple de code de longueur variable

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A • • —
 B — • • •
 C — • • — •
 D — • • •
 E •
 F • • — •
 G — — • •
 H • • • •
 I • •
 J • — — —
 K — • — —
 L — — • •
 M — —
 N — •
 O — — —
 P — — — •
 Q — — — • —
 R — • • •
 S • • • •
 T —

U • • —
 V • • • —
 W • — — —
 X — • • —
 Y — — • —
 Z — — — • •

1 • — — — — —
 2 • • — — — — —
 3 • • • — — — —
 4 • • • • — — —
 5 • • • • •
 6 — • • • • •
 7 — — • • • •
 8 — — — • • •
 9 — — — — •
 0 — — — — —

Exemple de code de longueur fixe

ASCII étendu

ISO/CEI 8859-15																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	non utilisé															
1x																
2x		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8x	non utilisé															
9x																
Ax		ı	ç	£	€	¥	Š	š	©	ª	«	¬		®	ˆ	
Bx	°	±	²	³	Ž	μ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	ı
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ