Culture Informatique

Luc Pellissier

2020 → 2021

Sommaire

Avertissement

Ce cours s'adresse aux étudiant·es de la L3 et de certains M1 de droit de l'Université Paris–Est Créteil Val-de-Marne. Il cherche à introduire à des étudiant·es n'ayant pas étudié des mathématiques depuis plus d'une dizaine d'années à la culture informatique.

Ce cours s'inspire du référentiel de la certification Pix, dont voici la liste des compétences évaluées [1] — on a noté en **gras** les thématiques dont ce cours traitera :

1. Informations et données
   a) Mener une recherche et une veille d'information

      Mener une recherche et une veille d'information pour répondre à un besoin d'information et se tenir au courant de l'actualité d'un sujet (avec un moteur de recherche, au sein d'un réseau social, par abonnement à des flux ou des lettres d'information, ou tout autre moyen).

      Thématiques associées : Web et navigation ; **Moteur de recherche et requête** ; Veille d'information, flux et curation ; Evaluation de l'information ; Source et citation ; **Gouvernance d'internet et ouverture du web** ; Abondance de l'information, filtrage et personnalisation ; Recul critique face à l'information et aux médias ; **Droit d'auteur**.

   b) Gérer des données

      Stocker et organiser des données pour les retrouver, les conserver et en faciliter l'accès et la gestion (avec un gestionnaire de fichiers, un espace de stockage en ligne, des tags, des classeurs, des bases de données, un système d'information, etc.).

      Thématiques associées : **Dossier et fichier** (Section 3.2) ; **Stockage et compression** ; **Transfert et synchronisation** ; Recherche et méta-données ; **Indexation sémantique et libellé (tag)** (Section 3.2) ; **Structuration des données** (Section 3.2) ; Système d'information ; **Localisation des données et droit applicable** ; Modèles et stratégies économiques ; Sécurité du système d'information.

   c) Traiter des données

      Appliquer des traitements à des données pour les analyser et les interpréter (avec un tableur, un programme, un logiciel de traitement d'enquête, une requête calcul dans une base de données, etc.).

      Thématiques associées : Données quantitatives, **type et format de données** ; Calcul, traitement statistique et représentation graphique ; Flux de données ; Collecte et exploitation de données massives ; **Pensée algorithmique et informatique** ; Vie privée et confidentialité ; **Interopérabilité**

2. Communication et collaboration
   a) Interagir

      Interagir avec des individus et de petits groupes pour échanger dans divers contextes liés à la vie privée ou à une activité professionnelle, de façon ponctuelle et récurrente (avec une messagerie électronique, une messagerie instantanée, un système de visio-conférence, etc.).

      Thématiques associées : Protocoles pour l'interaction ; Modalités d'interaction et rôles ; Applications et services pour l'interaction ; Vie privée et confidentialité ; Identité numérique et signaux ; Vie connectée ; Codes de communication et netiquette

   b) Partager et publier

      Partager et publier des informations et des contenus pour communiquer ses propres productions ou opinions, relayer celles des autres en contexte de communication publique (avec des plateformes de partage, des réseaux sociaux, des blogs, des espaces de forum et de commentaire, des CMS, etc.).

1. Récupérée sur `https://pix.fr/competences`

Thématiques associées : Protocoles et modalités de partage ; Applications et services pour le partage ; Règles de publication et visibilité ; Réseaux sociaux ; Liberté d'expression et droit à l'information ; Formation en ligne ; Vie privée et confidentialité ; Identité numérique et signaux ; Pratiques sociales et participation citoyenne ; e- Réputation et influence ; Ecriture pour le web ; Codes de communication et netiquette ; **Droit d'auteur**

c) Collaborer

Collaborer dans un groupe pour réaliser un projet, co-produire des ressources, des connaissances, des données, et pour apprendre (avec des plateformes de travail collaboratif et de partage de document, des éditeurs en ligne, des fonctionnalités de suivi de modifications ou de gestion de versions, etc.).

Thématiques associées : Modalités de collaboration et rôles ; Applications et services de partage de document et d'édition en ligne ; **Versions et révisions** ; **Droits d'accès et conflit d'accès** ; Gestion de projet ; **Droit d'auteur** ; Vie connectée ; Vie privée et confidentialité

d) S'insérer dans le monde numérique

Maîtriser les stratégies et enjeux de la présence en ligne, et choisir ses pratiques pour se positionner en tant qu'acteur social, économique et citoyen dans le monde numérique, en lien avec ses règles, limites et potentialités, et en accord avec des valeurs et/ou pour répondre à des objectifs (avec les réseaux sociaux et les outils permettant de développer une présence publique sur le web, et en lien avec la vie citoyenne, la vie professionnelle, la vie privée, etc.).

Thématiques associées : Identité numérique et signaux ; e-Réputation et influence ; Codes de communication et netiquette ; Pratiques sociales et participation citoyenne ; Modèles et stratégies économiques ; Questions éthiques et valeurs ; Gouvernance d'internet et ouverture du web ; Liberté d'expression et droit à l'information

3. Création de contenu

a) Développer des documents textuels

Produire des documents à contenu majoritairement textuel pour communiquer des idées, rendre compte et valoriser ses travaux (avec des logiciels de traitement de texte, de présentation, de création de page web, de carte conceptuelle, etc.).

Thématiques associées : **Applications d'édition de documents textuels** ; **Structure et séparation forme et contenu** ; Illustration et intégration ; Charte graphique et identité visuelle ; **Interopérabilité** ; **Ergonomie et réutilisabilité du document** ; **Accessibilité** ; **Droit d'auteur**

b) Développer des documents multimédia

Développer des documents à contenu multimédia pour créer ses propres productions multimédia, enrichir ses créations majoritairement textuelles ou créer une oeuvre transformative (mashup, remix, …) (avec des logiciels de capture et d'édition d'image / son / vidéo / animation, des logiciels utiles aux pré-traitements avant intégration, etc.).

Thématiques associées : Applications d'édition de documents multimédia ; Capture son, image et vidéo et numérisation ; **Interopérabilité** ; **Accessibilité** ; **Droit d'auteur** ; Charte graphique et identité visuelle

c) Adapter les documents à leur finalité

Adapter des documents de tous types en fonction de l'usage envisagé et maîtriser l'usage des licences pour permettre, faciliter et encadrer l'utilisation dans divers contextes (mise à jour fréquente, diffusion multicanale, impression, mise en ligne, projection, etc.) (avec les fonctionnalités des logiciels liées à la préparation d'impression, de projection, de mise en ligne, les outils de conversion de format, etc.).

Thématiques associées : **Licences** ; Diffusion et mise en ligne d'un document Ergonomie et réutilisabilité du document ; Ecriture pour le web ; **Interopérabilité** ; **Accessibilité** ; Vie privée et confidentialité

d) Programmer

Ecrire des programmes et des algorithmes pour répondre à un besoin (automatiser une tâche répétitive, accomplir des tâches complexes ou chronophages, résoudre un problème logique, etc.) et pour développer un contenu riche (jeu, site web, etc.) (avec des environnements de développement informatique simples, des logiciels de planification de tâches, etc.).

Thématiques associées : **Algorithme et programme** ; **Représentation et codage de l'information** ; **Complexité** ; **Pensée algorithmique et informatique** ; Collecte et exploitation de données massives ; Intelligence artificielle et robots

4. Protection et sécurité

a) Sécuriser l'environnement numérique

Sécuriser les équipements, les communications et les données pour se prémunir contre les attaques, pièges, désagréments et incidents susceptibles de nuire au bon fonctionnement des matériels, logiciels, sites internet, et de compromettre les transactions et les données (avec des logiciels de protection, des techniques de chiffrement, la maîtrise de bonnes pratiques, etc.).

Thématiques associées : **Attaques et menaces** ; **Chiffrement** ; Logiciels de prévention et de protection ; **Authentification** ; Sécurité du système d'information ; Vie privée et confidentialité

b) Protéger les données personnelles et la vie privée

Maîtriser ses traces et gérer les données personnelles pour protéger sa vie privée et celle des autres, et adopter une pratique éclairée (avec le paramétrage des paramètres de confidentialité, la surveillance régulière de ses traces par des alertes ou autres outils, etc.).

Thématiques associées : **Données personnelles et loi** ; Traces ; Vie privée et confidentialité ; Collecte et exploitation de données massives

c) Protéger la santé, le bien-être et l'environnement

Prévenir et limiter les risques générés par le numérique sur la santé, le bien- être et l'environnement mais aussi tirer parti de ses potentialités pour favoriser le développement personnel, le soin, l'inclusion dans la société et la qualité des conditions de vie, pour soi et pour les autres (avec la connaissance des effets du numérique sur la santé physique et psychique et sur l'environnement, et des pratiques, services et outils numériques dédiés au bien-être, à la santé, à l'accessibilité).

Thématiques associées : Ergonomie du poste de travail ; Communication sans fil et ondes ; **Impact environnemental** ; **Accessibilité** ; **Vie connectée** ; Capteurs ; Intelligence artificielle et robots ; Santé ; Vie privée et confidentialité

5. Environnement numérique

a) Résoudre des problèmes techniques

Résoudre des problèmes techniques pour garantir et rétablir le bon fonctionnement d'un environnement informatique (avec les outils de configuration et de maintenance des logiciels ou des systèmes d'exploitation, et en mobilisant les ressources techniques ou humaines nécessaires, etc.).

Thématiques associées : Panne et support informatique ; Administration et configuration ; **Maintenance et mise à jour** ; Sauvegarde et restauration ; **Interopérabilité** ; **Complexité**

b) Construire un environnement numérique

Installer, configurer et enrichir un environnement numérique (matériels, outils, services) pour disposer d'un cadre adapté aux activités menées, à leur contexte d'exercice ou à des valeurs (avec les outils de configuration des logiciels et des systèmes d'exploitation, l'installation de nouveaux logiciels ou la souscription à des services, etc.).

Thématiques associées : **Histoire de l'informatique** ; **Informatique et matériel** ; **Logiciels, applications et services** ; **Système d'exploitation** ; **Réseau informatique** ; **Offre (matériel, logiciel, service)** ; **Modèles et stratégies économiques**

Comme on peut le voir, on abordera pas toutes les thématiques, principalement du fait du gigantisme de la certification.

Première partie

Informatique

1

Les quatre concepts de l'informatique

L'*informatique* est à la fois une science (la branche des mathématiques qui répond à la question «Qu'est-ce qu'un calcul ?»), une industrie, un rayon de supermarché, un ensemble d'outils techniques que tout le monde utilise…

Autrement dit, l'informatique mélange des aspects théoriques, techniques, commerciaux. Pour y voir un peu plus clair, voyons quatre notions qui structurent l'informatique.

1.1 Algorithme

Le concept est rentré dans le débat public récemment et prend des connotations un peu mystérieuses. Un *algorithme* (déformation assez lointaine du nom du poète persan Abū ʿAbdallāh Muhammad ibn Mūsā al-Khwārizmī) est une suite d'opération ayant pour but d'arriver à un résultat. On a souvent comparé cette notion à celle d'une recette de cuisine : pour obtenir le même plat à partir des mêmes ingrédients, il peut y avoir plusieurs recettes, toutes ne prennent pas le même temps ou n'utilisent pas les mêmes ingrédients,… Mais dans tous les cas, les étapes doivent être précises ; et le résultat espéré toujours maintenu en vue.

Vous avez toustes appris, au cours de votre scolarité, quelques algorithmes. Citons-en un en particulier : l'algorithme de la multiplication.

1.2 Machine

Dans le cadre de ce cours, une *machine* est un objet (physique ou imaginaire) évoluant au cours du temps, et transformant ses entrées en sorties. Autrement dit, nous nous permettrons de parler de machine dès qu'on pourra identifier les trois éléments suivants :

— une évolution temporelle ;
— un sous-système dont on peut manipuler librement la position d'une manière qui affectera l'évolution temporelle. On le considérera comme l'*entrée* de la machine ;
— un sous-système dont on peut observer librement la position. On le considérera comme la *sortie*.

Par exemple, sur une calculatrice de poche, on considère comme entrée les pressions sur les touches, et comme sortie l'affichage de l'écran.

Il existe des machines électroniques, mais aussi fonctionnant sur d'autres principes. Ainsi, si je construit un modèle physique du bassin versant d'un fleuve, je peux considérer que la quantité d'eau que je place aux différentes sources sont les entrées, et les observations que je fais aux points d'intérêt sont les sorties.

1.3 Langage

On ne s'intéresse ici qu'aux langages artificiels, construits consciemment. Un *langage* est un ensemble restreint d'expression que l'on considère *valides*[1] décrivant quelque chose d'intéressant.

Il y a énormément de langages de cette forme. Par exemple, la Poste en a un pour décrire les adresses. La classification des êtres vivants en est un autre. Les conventions que l'on a pour écrire des plans (en architecture), ou des circuits électriques en forment.

1.4 Information

Un système peut contenir de l'information. Par exemple, une pièce de monnaie posée sur une table a deux états : un où pile est visible, un autre où face l'est ; et tous deux équiprobables. Un feu de signalisation a trois états : rouge, orange, et vert. Néanmoins, les trois états ne sont pas équiprobables : le système est moins souvent dans l'état orange que dans les autres (et possiblement les deux autres ne sont pas non plus équiprobables). Savoir que le système est dans l'état orange est donc plus étonnant que le reste. A fortiori pour un système complexe, comme un ascenseur : il peut être dans un certain nombre d'états (portes ouvertes au cinquième étage, porte fermées entre le premier et le deuxième,…), mais certains sont anormaux (portes ouvertes entre le premier et le deuxième). Savoir que l'ascenseur est dans un état rare contient donc plus d'informations que de savoir qu'il est dans un état normal. On choisit comme définition de l'*information* comme la différence entre connaître un système (et les probabilités de ces différents états) et connaître l'état dans lequel il est.

C'est une quantité physique, au même titre que la température, la masse,…

1. Cela signifie donc qu'il y a toujours des expressions invalides !

2

Codage de l'information

On montre comment représenter quelques types de données sur des dispositifs ayant deux états stables. On voit qu'il y a deux types de codes : ceux arbitraires, et ceux qui codent avec des nombres, eux-mêmes codés en binaire.

Nous allons d'abord voir comment rendre l'information manipulable, c'est-à-dire la représenter d'une manière permettant de :

— la récupérer ;
— faire les opérations qui nous intéressent dessus.

Pour ce faire, on va la *coder*, c'est-à-dire donner un système de représentation pour cette information. Chaque type d'information doit être codé différemment : en effet, on ne cherche pas à faire les mêmes opérations dessus (ainsi, on peut vouloir additionner ou multiplier des nombres, mais pas des mots). Néanmoins, on va tout coder dans le même système : le système *binaire*.

Il n'y a rien de particulier au système binaire, si ce n'est qu'il est le plus simple conceptuellement, et aussi le plus facile à réaliser physiquement.

Pour représenter une information codée en binaire, il suffit d'avoir un système à deux états. Par convention, on écrit un des deux 0 et l'autre 1, par exemple :

— un aimant a deux pôles : un pôle nord et un pôle sud. On peut décider qu'un aimant placé en présentant son pôle nord vaut 1 ;
— de l'électricité passe dans un fil. On décide que si le courant passe, cela vaut 1, 0 sinon ;
— de l'eau passe dans un tube ;
— …

On appellera un bit n'importe quel dispositif (physique ou non) ayant deux états stables.

2.1 Les nombres

Tout le monde a été exposé à deux systèmes de codage des nombres entiers. Étudions-les un peu.

les chiffres romains dans ce système [1], on dispose des symboles :

$$\mathrm{M}, \mathrm{C}, \mathrm{L}, \mathrm{X}, \mathrm{V}, \mathrm{I}$$

chacun représentant un nombre. Respectivement 1000, 100, 50, 10, 5 et 1.

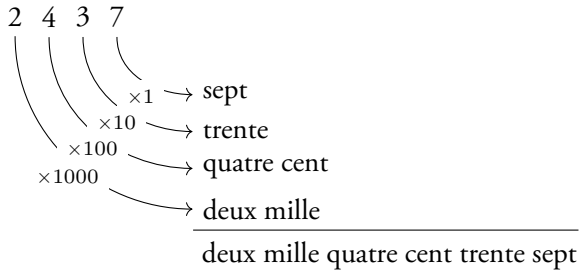
Pour écrire un nombre n'étant pas dans cette liste, on le décompose comme une addition d'éléments de la liste, et on écrit ces éléments dans l'ordre décroissant. Ainsi, 2020 sera MMXX.

la notation positionnelle grand-boutienne à base 10 ce système est celui qu'on utilise tous les jours. Néanmoins, observons un peu son fonctionnement. On dispose des symboles

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

Les nombres sont ensuite écrits comme des suites de ces chiffres, et chaque chiffre d'un nombre prend un sens dépendant de sa position.

En effet, on décide que le chiffre le plus à gauche est le chiffre des unités, c'est-à-dire doit être lu tel quel, tandis que le deuxième chiffre en partant de la gauche est lu comme le chiffre des dizaines, donc devant être lu comme multiplié par dix, et ainsi de suite. Autrement dit :



L'élément central de la base 10 est donc l'existence d'une échelle

$$1, 10, 100, 1000, ...$$

permettant de donner un sens aux différents chiffres selon leur position. Cette échelle (qui est celle des puissances de 10) ne sort pas de nulle part. Une manière de la voir est de dire que, pour lister les nombres, on commence par épuiser tous les symboles, c'est à dire

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9.$$

C'est suffisant pour représenter les nombres de zéro à neuf, mais on ne peut pas représenter le nombre dix avec un nombre à un chiffre. On décide donc de se donner une deuxième colonne pour représenter les nombres, et que un 1 dans cette colonne représente le plus petit nombre qu'on ne peut pas écrire avec un seul chiffre. Aussi, on va lire

$$10$$

comme dix (le plus petit nombre qu'on ne peut pas représenter avec un seul chiffre) plus zéro,

$$15$$

comme dix plus cinq, et ainsi de suite.

Plus généralement, le $i^{\text{ème}}$ chiffre dois être lu comme étant multiplié par le plus petit nombre ne pouvant pas être écrit avec $(i-1)$ chiffres, et additionné aux autres.

1. Il n'y a pas un seul système de notations en chiffres romains. On exposera ici la version la plus simple du système qui soit attestée.

0	0	2	10	4	100	8	1000	16	10000
1	1	3	11	5	101	9	1001	⋮	⋮
				6	110	10	1010		
				7	111	11	1011		
						12	1100		
						13	1101		
						14	1110		
						15	1111		

FIGURE 2.1 – Les premiers nombres, en décimal et en binaire

Pour coder les nombres en binaire, on va faire pareil. Introduisons tout d'abord une convention. Nous allons noter les nombres en toute lettres, et quand on voudra parler des nombres notés en base 10, on les écrira en une police à chasse fixe, avec un indice 10 ; et de même pour les nombres en base 2.

Ainsi, onze est un nombre, tandis que $\texttt{11}_{10}$ est la représentation en base 10 de ce nombre, et $\texttt{11}_2$ est la représentation binaire d'un autre nombre (en l'occurrence trois).

Remarque 1. On évitera de prononcer les suites de chiffres comme si c'était des nombres.

Ainsi, $\texttt{11}_{10}$ se prononce «un-un en base 10» et pas «onze».

Comme on a deux symboles en binaire (`0` et `1`), on peut noter deux nombres avec un seul chiffre : zéro et un. Donc $\texttt{0}_2$ représente zéro ; et $\texttt{1}_2$ représente 1. Le plus petit nombre qu'on ne peut pas écrire avec un seul chiffre est deux. Donc, on convient que un 1 dans la deuxième colonne signifie deux, et donc que $\texttt{10}_2$ représente une fois deux plus zéro, c'est-à-dire deux ; et de même

$$\texttt{11}_2$$

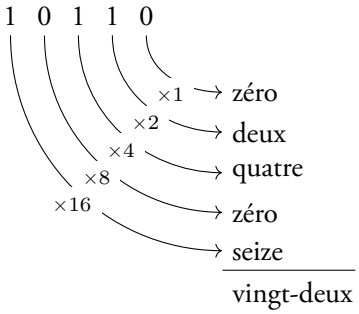
représente une fois deux plus un, c'est-à-dire trois. Autrement dit, les premiers nombres sont représentés en binaire par

$$\texttt{0}_2, \texttt{1}_2, \texttt{10}_2, \texttt{11}_2.$$

Là encore, on ne peut pas représenter au-delà de trois avec seulement deux chiffres.

Donc, on décide que quatre, le plus petit chiffre ne pouvant pas être représenté avec deux chiffres, est représenté par un 1 dans la troisième colonne. Et ainsi de suite. On obtient donc la table de la Figure 2.1.

Ainsi, on peut coder tous les nombres (entiers) comme des suites de 0 et de 1, de la même manière qu'on s'est habitué à les coder comme des suites de 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Et donc, on peut décoder un nombre écrit en binaire en multipliant chaque chiffre par un élément de l'échelle 1, 2, 4, 8, 16, 32, 64,…(où chaque élément de l'échelle est comme en base 10, le plus petit nombre ne pouvant pas être écrit avec moins de chiffre, et se calcule à partir de l'élément précédent en le multipliant par 2).



Exercice 1. Combien vaut 10101110_2 ?
Inversement comment écrire 2020_{10} ou 43589_{10} en binaire ?

Exercice 2. Comment reconnaître si un nombre écrit en binaire est pair ?
Comment multiplier par deux un nombre écrit en binaire ?
En déduire un algorithme permettant de passer du décimal au binaire.

Le codage et le décodage

On déduit de tout ce qui précède deux algorithmes :
— un de codage, qui permet d'écrire un nombre en binaire.
On part d'un nombre, disons n. On le divise par deux au sens des nombres entiers, c'est-à-dire qu'on l'écrit comme la somme d'un reste, et du produit de deux et d'un quotient :

$$n = 2q + r$$

Le reste r est soit 0 (si n était pair), soit 1 (si n était impair). On écrit le reste r le plus à droite, puis on recommence avec le quotient, tant que le quotient ne vaut pas 0.
— un de décodage, qui permet de lire un nombre en binaire.
De droite à gauche, on multiplie chaque chiffre par l'élément de l'échelle correspondant, puis on additionne le tout.

La longueur fixe

Ainsi, `1` code le nombre un, et `11` code l'entier trois. Mais il arrive aussi que l'on veuille coder deux fois l'entier un d'affilée (par exemple, dans un numéro de téléphone), auquel cas, on veut pouvoir savoir si on a face à nous deux fois de suite un, ou bien trois.

On ne peut pas utiliser de caractères en plus (comme une espace !) vu qu'il faudrait lui-même le coder (avec des `0` et des `1`) et l'ambiguïté resterait à l'identique. On prend donc la convention de toujours représenter les entiers avec un nombre fixé de bits. Pour reprendre l'exemple de la base 10, on pourrait décider de toujours écrire les nombres (ceux pour lesquels cela est possible en tout cas) sur cinq chiffres, et par exemple, au lieu d'écrire 10 pour le nombre dix, écrire 00010.

Pour des raisons historiques, on a choisi de coder les nombres sur huit bits, c'est à dire qu'on stocke facilement les nombres compris entre zéro et 255. On fait donc le choix que devant une chaîne de bits, on coupe tous les huit et on les lit comme un autre nombre. Donc zéro est représenté par `00000000` et huit par `00001000` . Plus généralement, on a la table de la Figure 2.2.

Inversement, `0011011001000011` représente deux nombres d'affilée, cinquante-quatre suivi de soixante-sept.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111	00001000	00001001	00001010	00001011	00001100	00001101	00001110	00001111
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00010000	00010001	00010010	00010011	00010100	00010101	00010110	00010111	00011000	00011001	00011010	00011011	00011100	00011101	00011110	00011111
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111	00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
00110000	00110001	00110010	00110011	00110100	00110101	00110110	00110111	00111000	00111001	00111010	00111011	00111100	00111101	00111110	00111111
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
01000000	01000001	01000010	01000011	01000100	01000101	01000110	01000111	01001000	01001001	01001010	01001011	01001100	01001101	01001110	01001111
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
01010000	01010001	01010010	01010011	01010100	01010101	01010110	01010111	01011000	01011001	01011010	01011011	01011100	01011101	01011110	01011111
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
01100000	01100001	01100010	01100011	01100100	01100101	01100110	01100111	01101000	01101001	01101010	01101011	01101100	01101101	01101110	01101111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
01110000	01110001	01110010	01110011	01110100	01110101	01110110	01110111	01111000	01111001	01111010	01111011	01111100	01111101	01111110	01111111
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
10000000	10000001	10000010	10000011	10000100	10000101	10000110	10000111	10001000	10001001	10001010	10001011	10001100	10001101	10001110	10001111
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
10010000	10010001	10010010	10010011	10010100	10010101	10010110	10010111	10011000	10011001	10011010	10011011	10011100	10011101	10011110	10011111
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
10100000	10100001	10100010	10100011	10100100	10100101	10100110	10100111	10101000	10101001	10101010	10101011	10101100	10101101	10101110	10101111
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
10110000	10110001	10110010	10110011	10110100	10110101	10110110	10110111	10111000	10111001	10111010	10111011	10111100	10111101	10111110	10111111
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
11000000	11000001	11000010	11000011	11000100	11000101	11000110	11000111	11001000	11001001	11001010	11001011	11001100	11001101	11001110	11001111
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
11010000	11010001	11010010	11010011	11010100	11010101	11010110	11010111	11011000	11011001	11011010	11011011	11011100	11011101	11011110	11011111
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
11100000	11100001	11100010	11100011	11100100	11100101	11100110	11100111	11101000	11101001	11101010	11101011	11101100	11101101	11101110	11101111
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
11110000	11110001	11110010	11110011	11110100	11110101	11110110	11110111	11111000	11111001	11111010	11111011	11111100	11111101	11111110	11111111

Figure 2.2 – Les nombres écrits sur huit bits

2.2 Les images

Comme on peut coder des nombres, on sait coder n'importe quoi qui peut être codé avec des nombres.

C'est par exemple le cas des couleurs. L'œil a trois récepteurs de couleur : rouge, bleu et vert. Toutes les couleurs qu'un humain voit sont en fait liées à l'excitation de ces trois récepteurs. Trois points de ces trois couleurs vus de suffisamment loin seront complètement mélangés en un seul point d'une couleur résultante.

Dans la norme RGB (red-green-blue), on code chaque couleurs comme trois nombres entre 0 et 255 : 0 correspondant à la lumière d'une couleure allumée minimalement, 255 maximalement. Puis, en donnant les trois niveaux des couleurs, on obtient toutes les couleurs qu'un œil humain peut voir (sous réserve que l'écran soit vraiment capable de faire ces couleurs !)

Exemple 1.

79,1,71 • contient donc 79 de rouge, 1 de vert et 71 de bleu, autrement dit sera codé par

`010011110000000101000111`.

119,45,139 •

130,108,148 •

162,136,227 •

18,53,91 •

141,170,157 •

On stocke des entiers entre 0 et 255, comme ça, chaque couleur est codée sur trois octets. On peut donc comme ça obtenir 16 777 216 de couleurs.

2.3 Les sons

— Fréquence d'échantillonage
— nombre de bits pour chaque échantillon
Pour un CD, 44100 échantillons par seconde, 16 bits, ce qui fait 1 411 200 bits par seconde
Pour un MP3, seulement 320 000 bits par seconde.

2.4 Le texte

Pour coder du texte, il a fallu faire un choix plus arbitraire. On a décidé dans les années 1960 de coder les lettres en usage aux États-Unis, ainsi qu'un certain nombre de caractères de contrôle, sur 7 bits, c'est à dire qu'il y a 2^7 possibilités : de `0000000` à `1111111`. Les premiers caractères sont des caractères de contrôle (retour à la ligne,…). Les caractères significatifs sont présentés Figure 2.3.

Cette norme, appelée ASCII, est devenue universelle, dans le sens où toutes les autres normes (pour contenir des accents, ou d'autres caractères) en sont des extensions. Aujourd'hui, la norme qui devient de plus en plus universelle est la norme Unicode [2], qui contient tous les accents du latin, mais aussi le grec, le cyrillique, les sinogrammes,… elle vise à contenir tous les caractères jamais écrit. Depuis une dizaine d'années, elle contient les emojis.

Par exemple les trois singes sont donnés par :

CODE	DESSIN	NOM
11111011001001000	🙈	SEE-NO-EVIL MONKEY
11111011001001001	🙉	HEAR-NO-EVIL MONKEY
11111011001001010	🙊	SPEAK-NO-EVIL MONKEY

Ainsi, selon le contexte, 1 pourra être codé comme `00000001` (s'il est considéré comme un nombre) ou `0110001` (s'il est considéré comme un caractère).

2.5 Quantité de données — Quantité d'information

L'unité de base est donc le système pouvant avoir deux valeurs.

On appelle un tel système, et la quantité d'information qu'il contient, un *bit* (symbole bit). Ainsi, la norme ASCII code les caractères sur 7 bit.

2. On peut la consulter en `https://www.unicode.org/charts/`

	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
0100000	0100001	0100010	0100011	0100100	0100101	0100110	0100111	0101000	0101001	0101010	0101011	0101100	0101101	0101110	0101111
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0110000	0110001	0110010	0110011	0110100	0110101	0110110	0110111	0111000	0111001	0111010	0111011	0111100	0111101	0111110	0111111
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1000000	1000001	1000010	1000011	1000100	1000101	1000110	1000111	1001000	1001001	1001010	1001011	1001100	1001101	1001110	1001111
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
1010000	1010001	1010010	1010011	1010100	1010101	1010110	1010111	1011000	1011001	1011010	1011011	1011100	1011101	1011110	1011111
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
1100000	1100001	1100010	1100011	1100100	1100101	1100110	1100111	1101000	1101001	1101010	1101011	1101100	1101101	1101110	1101111
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⌧
1110000	1110001	1110010	1110011	1110100	1110101	1110110	1110111	1111000	1111001	1111010	1111011	1111100	1111101	1111110	1111111

Figure 2.3 – Les caractères dans la norme ASCII

En général, on considère les bits par groupe de 8, un *octet*. Un octet peut prendre 256 valeurs différentes. On note cette unité d'information o. Ainsi, on a pu voir qu'on pouvait coder chaque pixel d'une image par une couleur, avec trois octets.

On utilise les préfixes usuels de multiples : ainsi, un kilo-octet (Ko) est mille octets,…

Toutefois, il y a une deuxième habitude de compter un multiple de $2^{10} = 1024$.

3

Architecture des ordinateurs

On voit les principes sur lesquels les ordinateurs sont construits, en particulier quelques notions de comment le matériel traite les données, puis comment le logiciel permet de se servir du matériel. On voit que le fonctionnement courant est de rajouter des couches d'abstraction logicielles pour résoudre des problèmes, ce qui est une solution souple, mais rajoutant des dépendances, techniques et politiques, ainsi que des externalités écologiques.

On va distinguer deux parties dans un ordinateur : le matériel et le logiciel.

3.1 Matériel

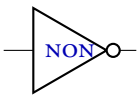
Le *matériel* est un dispositif physique. Il peut prendre beaucoup de formes (des transistors, des plantes carnivores,…) mais il a comme principale caractéristique d'exister dans le monde physique. Le matériel tel qu'il existe aujourd'hui est de plusieurs types, selon qu'il traite de l'information, la stocke ou soit une interface avec le monde extérieur (soit qu'il affiche de l'information, soit qu'il en récupère).

Porte non

La *porte* NON est une machine avec comme entrée un bit et comme sortie un bit et telle que, après un moment de stabilisation, si son entrée vaut 0, alors sa sortie vaut 1 ; et si son entrée vaut 1, alors, sa sortie vaut 0. Autrement dit, si on appelle X son entrée, la porte NON est régie par la table :

X	NON X
0	1
1	0

On peut la noter, sous forme de circuit :



Ce qui se lit de gauche à droite, avec à gauche l'entrée et à droite la sortie. On peut donc noter la table en dessinant :

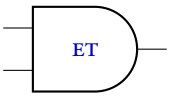
Son nom vient de ce que sa sortie vaut 1 si son entrée NE vaut PAS 1.

Porte et

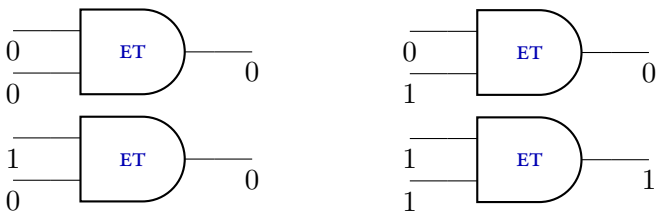
La *porte* ET est une machine avec comme entrée deux bits et comme sortie un bit et telle que, après un moment de stabilisation, si ses deux entrées valent 1, alors sa sortie vaut 1 ; et sinon, sa sortie vaut 0. Autrement dit, si on appelle X et Y ses deux entrées, la porte ET est régie par la table :

X	Y	X ET Y
0	0	0
0	1	0
1	0	0
1	1	1

On peut la noter, sous forme de circuit :



De même, la table peut être ré-écrite sous la forme des quatre circuits :



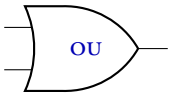
Son nom vient de ce que sa sortie vaut 1 si sa première entrée vaut 1 ET sa deuxième entrée vaut 1.

Porte ou

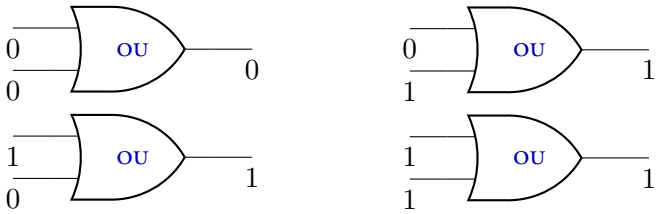
La *porte* OU est une machine avec comme entrée deux bits et comme sortie un bit et telle que, après un moment de stabilisation, si ses deux entrées valent 0, alors sa sortie vaut 0 ; et sinon, sa sortie vaut 1. Autrement dit, si on appelle X et Y ses deux entrées, la porte ET est régie par la table :

X	Y	X OU Y
0	0	0
0	1	1
1	0	1
1	1	1

On peut la noter, sous forme de circuit :



De même, la table peut être ré-écrite sous la forme des quatre circuits :



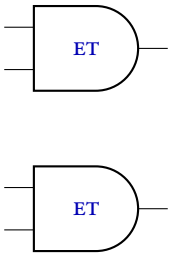
Son nom vient de ce que sa sortie vaut 1 si sa première entrée vaut 1 OU sa deuxième entrée vaut 1.

Portes constantes

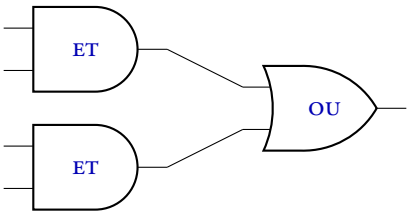
Les deux *portes constantes* sont des machines qui n'ont pas d'entrée, et une sortie, qui vaut soit toujours 0, soit toujours 1.

Circuits

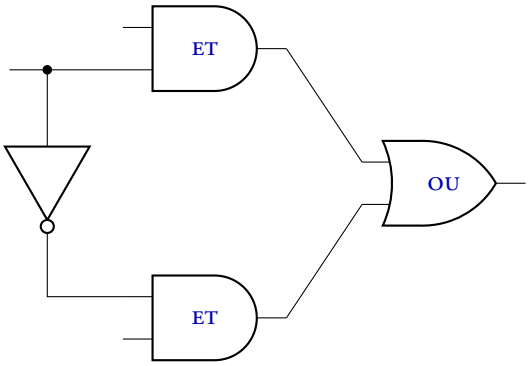
On peut assembler les portes pour construire des machines de plus en plus complexes. Par exemple, en positionnant deux portes ET l'une à côté de l'autre, on obtient une machine avec quatre bits en entrée et deux en sortie.



De même, on peut brancher la sortie d'une porte sur l'entrée d'une autre porte :



et faire des branchements arbitrairement compliqués :



Exercice 3. Donner, pour chacun des circuits dessinés plus haut, leur nombre d'entrées, de sorties, et leur table.

Théorème 1. *Toutes les fonctions d'un nombre arbitraire de bits dans un autre nombre arbitraire de bits peut être réalisée en connectant entre elles des portes* ET, OU *et* NON, *et les deux portes constantes égales à zéro et à un.*

Démonstration. Ce qui suit est une démonstration mathématique. Elle n'est pas exigible.

Commençons par reformuler l'énoncé, en explicitant les quantificateurs et les variables. Soient n, m deux entiers positifs. Toutes les fonctions de $\{0,1\}^n$ dans $\{0,1\}^m$ peuvent être écrites à bases de portes ET, OU et NON.

On remarque tout d'abord qu'il suffit de prouver le théorème pour $m = 1$: en effet, si l'on a montré qu'on sait réaliser toutes les fonctions de $\{0,1\}^n$ dans $\{0,1\}$, on va pouvoir réaliser une fonction dont la sortie est de plusieurs bits en la construisant bit à bit.

Pour prouver ce cas particulier (qui entraîne le cas général), on va utiliser le principe de récurrence, qui peut s'énoncer ainsi : si une propriété

— est vraie en 0 ;

— est telle que, si elle est vraie pour un entier n, alors elle est aussi vraie pour $n + 1$

alors elle est vraie de tous les entiers positifs. Une autre manière de voir cela est de dire que pour chaque entier n, soit on sait résoudre le problème (c'est le cas en 0), soit on sait se ramener au même problème mais pour un entier plus petit. Comme on se ramène à des problèmes de plus en plus petits, on sait résoudre systématiquement.

On a donc à montrer :

1. que toutes les fonctions de $\{0,1\}^0$ dans $\{0,1\}$ peuvent être réalisées. Ces fonctions sont nécessairement des constantes ; on les réalise avec les constantes.
2. que si on sait, pour n'importe quel entier n, réaliser toutes les fonctions de $\{0,1\}^n$ dans $\{0,1\}$, alors on sait aussi réaliser toutes les fonctions de $\{0,1\}^{n+1}$ dans $\{0,1\}$.

   Soit n un entier positif. Supposons qu'on sache réaliser toutes les fonctions de $\{0,1\}^n$ dans $\{0,1\}$.

Soit f une fonction quelconque de $\{0,1\}^{n+1}$ dans $\{0,1\}$. La fonction $f_1 : (x_1, \dots, x_n) \mapsto f(x_1, \dots, x_n, 1)$ est une fonction de $\{0,1\}^n$ dans $\{0,1\}$. Elle est donc réalisable. De même pour $f_0 : (x_1, \dots, x_n) \mapsto f(x_1, \dots, x_n, 0)$.

Donc, on peut réaliser f en branchant les sorties de f_0 et f_1 sur des portes ET avec x_{n+1} pour jeter le résultat selon la valeur de x_{n+1}.

En effet, le ET des sorties de f_1 avec x_{n+1} vaut les sorties de f_1 si x_{n+1} vaut 1 et zéro sinon.

En vertu du principe de récurrence, on peut donc réaliser, pour tout entier n, toutes les fonctions de $\{0,1\}^n$ dans $\{0,1\}$. ☺

En particulier, cela signifie, par exemple, que la fonction qui calcule l'addition d'entiers sur huit bits (en jettant la dernière retenue) est réalisable par un circuit contenant uniquement des portes ET, OU et NON.

Processeurs

Un *processeur* est constitué d'un ensemble de circuits, chacun réalisant une fonction. Ainsi, dans un processeur, il y aura un circuit réalisant l'addition, un réalisant la multiplication, et ainsi de suite pour toutes les opérations arithmétiques ; mais aussi des circuits plus spécialisées, réalisant des opérations utiles dans le cadre du calcul graphique (pour dessiner rapidement des objets en 3D, par exemple), pour décoder des vidéos ou des fichiers sons, etc…

Chaque processeur déclare donc une série d'instructions qu'il peut faire, chacune étant réalisée par un circuit [1]. Un processeur effectue ensuite toutes ses opérations en parallèle sur ses entrées : il duplique les entrées et en fait les entrées de chacun de ses circuits.

Cela signifie que plus un processeur a de circuits (y compris très spécialisés, qui ne servent rarement, voire ne servent plus mais on été jugés utiles autrefois), et donc d'instructions, plus il consomme d'énergie (car il fait tous les calculs possibles sur tous les circuits possibles), y compris si on ne se sert jamais des instructions.

Cela signifie que les entreprises qui conçoivent des processeurs doivent faire des choix extrêmement compliqués entre mettre plus d'instructions (qui donc, permettront de faire certains calculs de manière très rapide) et des considérations de consommation électrique (plus il y a de circuits allumés en même temps, plus ça consomme) et de dissipation thermique (plus il y a de circuits allumés en même temps, plus ça chauffe).

Chaque circuit donne donc un résultat en sortie, il faut ensuite pouvoir choisir entre les différentes opérations celle que l'on veut réellement. Supposons que l'on ait un processeur composé de deux circuits, un calculant l'addition, un autre la multiplication, sur trois bits. Plus précisément, on suppose qu'on a deux circuits, qui ont chacun six bits d'entrée (pour coder deux nombres, chacun sur huit bits), et trois bits de sortie (pour coder un nombre de résultat), comme sur la Figure 3.1. Chacun des circuits tourne en parallèle, et donne ses résultats en sortie. On jette ensuite le résultat qui ne nous intéresse pas : pour ce faire, on ajoute deux bits d'entrée, les *bits de contrôle*, représentés en pointillé sur la figure. L'un correspond au circuit additionneur, l'autre au circuit multiplicateur. Quand le bit associé à l'additionneur est à 1, les portes ET laissent passer la sortie de l'additionneur, mais quand il est à 0, les portes ET remplacent la sortie par des zéros. De même pour le multiplicateur.

Ainsi, si le bit associé à l'additionneur est à 1, et celui associé au multiplicateur à 0, la sortie du circuit complet vaut la sortie de l'additionneur ; et inversement si le bit associé au multiplicateur est à 1, et celui associé à l'additionneur à 0, la sortie du circuit complet vaut la sortie du multiplicateur. Si en entrée, on a :

```
10110001
```

1. Évidemment, je simplifie. Certaines correspondent à des circuits qui existaient dans une version précédente, et qui sont simulés avec d'autres circuits.

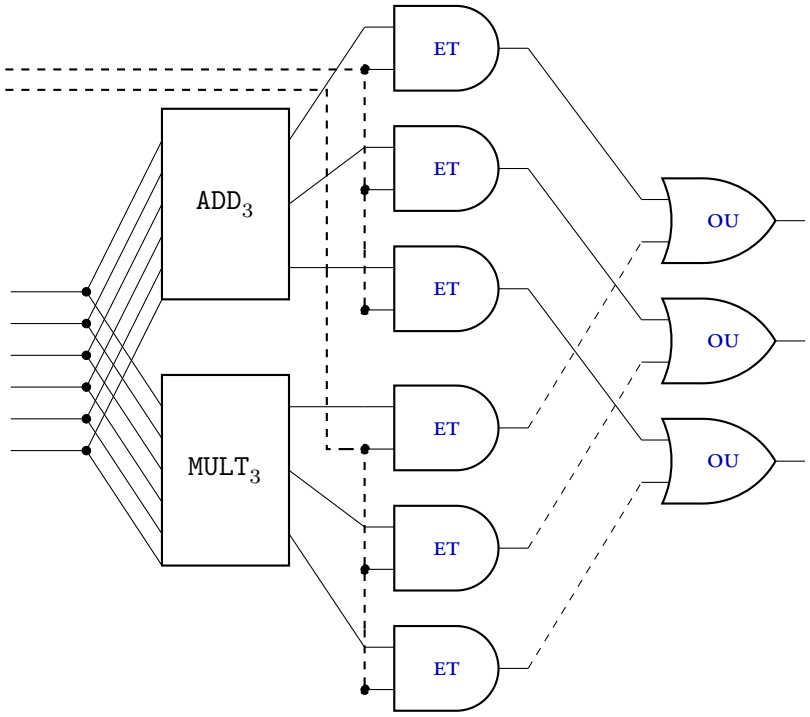


FIGURE 3.1 – Un processeur réalisant l'addition et la multiplication sur 3 bits

le processeur va donc calculer la somme du nombre représenté par `110` et celui représenté par `001`. Donc un processeur définit un langage d'opérations. Ici, `10` sur les bits de contrôle veut dire «addition», tandis que `01` veut dire «multiplication».

Mémoires

D'autres types de machines peuvent être réalisées avec des circuits. Une d'entre elle sont les *mémoires*. On ne cherchera pas à les décrire. Une mémoire est une machine qui prend comme entrée :

— une adresse
— un bit déterminant l'opération à faire
— un bit optionnel

et un bit en sortie. Une mémoire contient plusieurs *cases*, chacune contenant un bit. Chaque case a une *adresse*, qui est un nombre (compris entre 0 et un nombre maximal, qui est la taille de la mémoire). L'idée étant que l'on peut lire et écrire dans chaque case de la mémoire, c'est-à-dire soit récupérer le bit qui y est écrit, soit écrire par dessus. Ainsi, deux opérations peuvent être faites, et on indique avec un bit spécial en entrée si on souhaite écrire ou lire.

Donc, si on a écrit une adresse ℓ et le bit de contrôle à 0 (pour la lecture), après stabilisation, la sortie vaudra la valeur du bit dans la case numéro ℓ.

Si par contre, on a écrit une adresse ℓ et le bit de contrôle à 1 (pour l'écriture), après stabilisation, la case ℓ de la mémoire prendra la valeur du dernier bit, le bit optionnel, et la sortie prendra une valeur quelconque [2].

Ainsi, une mémoire permet de sauver des informations pour pouvoir s'en resservir après.

Il y en a de plusieurs types : certaines sont situées à l'intérieur même d'un processeur, d'autres peuvent être physiquement à l'autre bout du monde. Elles sont construites selon des technologiques

2. En général, on décide que la sortie vaut 0 si l'écriture s'est bien passée, et 1 sinon, mais ce n'est pas essentiel.

différentes, et se situent le long d'un axe où elles sont soit plus chères et rapides, soit meilleurs marché mais lentes.

On a ainsi tendance à placer les mémoires en cercles concentriques autour du processeur, avec des mémoires de plus en plus grosses, mais de plus en plus lentes.

Exécution d'un calcul

Au cours d'un calcul, l'ensemble du processeur et de ses mémoires va donc exécuter une à une des instructions, qui peuvent écrire dans une mémoire une information qui était dans une mémoire plus proche ou plus lointaine du processeur, voire un résultat à peine calculé ; effectué un calcul. La prochaine instruction à exécutée est elle aussi inscrite dans une mémoire (dans le type le plus rapide). Ainsi, programmer pour un ordinateur donné est donc donner une suite d'instructions qui va activer et désactiver certains circuits.

Comme chaque ordinateur a potentiellement des instructions un peu différentes, et qu'un processeur à 3 GHz (ce qui est tout à fait courant depuis une vingtaine d'années) effectue 3 milliards d'instructions chaque seconde, on voit que ce n'est pas possible à faire à la main.

Périphériques

Autour du système processeur + mémoire, se trouvent des *périphériques* qui permettent de communiquer avec l'ordinateur. Ils sont de deux sortes : les périphériques d'entrée et les périphériques de sortie.

Parmi les périphériques d'entrée, on peut remarquer :
- clavier
- souris
- trackpad
- GPS (sur les téléphones)
- capteurs de température
- micro
- webcam

Parmi les périphériques de sortie :
- écran
- haut-parleur
- imprimante

Ils consistent tous soit à écrire dans une mémoire particulière, soit à lire dans une mémoire particulière.

Ainsi, un écran lira dans la mémoire graphique ce qu'il doit afficher. Il dont donc adhérer à une convention commune avec le reste de l'ordinateur pour savoir comment la mémoire graphique est structurée, et ainsi quelles couleurs afficher et où. De même, un clavier écrira dans une zone de la mémoire quelle touche a été appuyée. Là encore, il faut que le clavier et le reste de l'ordinateur aient la même convention sur le sens d'un appui sur chacune des touches.

3.2 Logiciel

On voit qu'utiliser un ordinateur *particulier* nécessite de connaître précisément son architecture. Ainsi, si on veut calculer 4+7, il faut connaître les conventions de codage des nombres (sur combien de bits sont-ils représentés ? Est-ce qu'on écrit de gauche à droite ?) et le code de l'instruction d'addition (qui peut changer de processeur à processeur, et peut-être changé par la présence d'autres instructions : y-en-a-t'il une seule, ou plusieurs, selon le type de nombres ?). Plutôt que de devoir avoir une pratique hyper-spécifique à une machine donnée, on fait un autre choix : rajouter une couche au-dessus du matériel, le *logiciel*, que l'on peut qualifier de *ce qui est entre l'esprit et la matière* (Leroy 2019).

Ainsi, on peut écrire un logiciel qui, connaissant la disposition des touches d'un clavier, saura reconnaître que l'on a tapé 4+7, le traduira en le code de l'opération et le code des deux nombres, fera exécuter l'opération, récupérera le résultat et ira écrire, dans la mémoire graphique, une certaine configuration qui provoquera l'affichage, quelque part sur l'écran, d'un changement de couleur de pixels de manière à dessiner un 11.

On voit que cela nécessite de connaître beaucoup de conventions spécifiques à un ordinateur donné ; on voit surtout que sans ses conventions, il n'y a aucun *sens*, il y a des pressions sur des touches, des bits qui changent de valeur. Le sens, dans le fonctionnement d'un ordinateur, est un épiphénomène, qui dépend de ce que des conventions soient appliquées uniformément.

Le logiciel apparaît donc naturellement comme une surcouche abstrayant les particularités d'un ordinateur précis : ainsi, alors que chaque composante d'un ordinateur est variable, floue, sans aucune garantie d'être consistante, la machine entière (processeur, mémoires, périphériques et logiciel), elle propose une interface plus stable à son utilisateur humain. C'est encore une machine — elle a une évolution temporelle, des entrées (certains des périphériques) et des sorties (d'autres des périphériques) bien définies — mais son fonctionnement interne devient opaque.

Abstractions

De nombreux logiciels et du matériel hétérogène doivent donc s'accorder sur des conventions (comment la mémoire graphique doit être lue, ce que veut dire la touche numéro 45 d'un clavier,…). Cela rend le développement de logiciels *portables*, c'est-à-dire fonctionnant sur deux ordinateurs différents, extrêmement compliqué, voire impossible. On règle le problème en introduisant des *abstractions*.

Prenons l'exemple de l'affichage à l'écran : si un logiciel veut afficher un rectangle gris clair de 3cm de long et 1cm de large, entouré de gris foncé, avec écrit au milieu «OK», il peut :

— connaître la taille des pixels à l'écran, et calculer qu'il doit en afficher une ligne de disons 300 pixels noirs (c'est-à-dire, écrire dans la mémoire graphique, dans les cases correspondant à chacun de ces pixels, le code de la couleur noire), puis une ligne commençant par un pixel noir suivi de 298 pixels gris et terminant par un pixel noir (c'est-à-dire, là encore, écrire dans la mémoire graphique…), etc. Pour dessiner un caractère, c'est encore pire, d'ailleurs !

— utiliser une bibliothèque de fonctions qui sache tracer des lignes, des lettres et des rectangles. Une telle bibliothèque est donc une couche logicielle qui se charge de traduire entre «dessiner une ligne noire» et «placer du noir dans la mémoire correspondant à tel et tel pixel».
Cela simplifie la vie du logiciel, qui n'a plus besoin de connaître les détails du matériel. Ainsi, son logiciel fonctionnera sur n'importe quel ordinateur pour lequel cette bibliothèque existe, peut importe la manière dont elle marche.

— utiliser une bibliothèque de fonctions qui contienne, prédéfinie, une notion de bouton de confirmation. Auquel cas, il n'y a qu'à lui demander d'en afficher un.
Ici, le logiciel ne sait même pas quelle forme aura le bouton. L'entité bouton s'est mise à avoir un sens phénoménal, par l'abstraction.

Ainsi, les différentes abstractions permettent de rendre le logiciel plus portable et adaptable (un bouton de confirmation est un bouton de confirmation, quelle que soit la langue dans laquelle il est écrit : si quelqu'un traduit la bibliothèque de fonction, le logiciel au-dessus n'a même pas à s'en rendre compte).

En échange, il a besoin d'une couche logicielle pour tourner. Cela pose des problèmes de bien des manières :

contrôle cette couche logicielle est écrite par quelqu'un d'autre, qui n'a pas forcément les mêmes intérêts, peut l'abandonner ou la brider pour des raisons stratégiques ;

lenteur au lieu d'écrire dans la mémoire graphique, on donne une instruction, qui sera traduite, potentiellement à travers plusieurs couches, avant d'*in fine*, écrire dans la mémoire graphique. Ce qui prenait une instruction en prend maintenant quelques centaines ;

dépendances le logiciel est certes plus portable, mais uniquement tant que la couche de traduction est disponible pour le nouveau matériel ;

bugs rien ne garantit que la couche de traduction soit correcte. Cela rend la compréhension de qui est responsable du bug extrêmement compliquée.

Donnons un deuxième exemple d'abstraction : les systèmes de *fichiers*. Dans les mémoires d'un ordinateur, il n'y a que des bits, valant 0 ou 1, mais rien ressemblant à une unité conceptuelle. De plus, sont mélangés des résultats intermédiaires, des caches pas encore purgés, et des informations importantes pour l'utilisateur.

Un *système de fichiers* est une convention pour organiser l'information dans les mémoires et fournir une abstraction pour les logiciels. En règle générale, il assigne à une zone de la mémoire une fonction de répertoire : dans le répertoire, on code des informations du style «la zone qui va du bit numéro 234686 au bit numéro 89765567 est un fichier, nommé `fichier.txt`». Un fichier n'est donc qu'une zone de la mémoire, déclarée par un répertoire n'être qu'une. Ainsi, un fichier n'a de sens qu'au sein d'un système, mais pas en lui-même.

Ainsi, plutôt que d'accéder à un bit particulier de la mémoire, un logiciel utilisant un système de fichiers va demander à une couche logicielle de lire un bit d'un fichier ; cette couche sait lire le répertoire, et accédera à l'endroit pertinent. Là encore, cela permet plus de souplesse (un même fichier peut changer d'emplacement, et être toujours *le même*[3]) et de portabilité (on peut utiliser des mémoires excessivement différentes — disques dur, clefs USB, voire même Internet), ainsi que la possibilité d'exprimer des relations entre les fichiers : ainsi, dans les systèmes de fichiers *hiérarchiques*, le répertoire peut aussi contenir des *dossiers*, c'est-à-dire des listes nommées d'autres fichiers et dossiers.

Il y a de nombreux systèmes de fichiers. Parmi les plus courants, `FAT32`, `NTFS`, `ext3`, `APFS`.

Il est ici aussi possible de faire une deuxième couche d'abstraction par-dessus le système de fichier. Ainsi, les applications spécialisées pour gérer des bibliothèques de photos ou de musique en sont : chaque photo ou morceau est un fichier, mais abstrait de telle manière qu'il soit identifiable autrement que par son nom et sa position dans une hiérarchie de fichiers.

Il y a donc de nombreuses couches logicielles fournissant de l'abstraction, partiellement compatibles entre elles, et partiellement dépendantes des unes des autres. La situation est quelque peu chaotique.

Applications

Une application, ou un *programme* est donc une machine qui, au travers des périphériques, va recevoir des entrées et calculer des sorties. Pour cela, il fait appel à des couches d'abstractions. Ainsi, un programme ne pourra s'exécuter que si toutes les couches d'abstraction dont il dépend sont présentes.

Ainsi, il va dépendre des décisions politiques des groupes qui écrivent et maintiennent les couches d'abstraction. Chaque auteurice de programme est donc dans une position très difficile : iel se rend la vie plus facile en utilisant des couches d'abstraction plus riches, mais dépend de plus en plus du bon vouloir de leur auteurice. De même, maintenir une couche d'abstraction, et la rendre utilisable sur beaucoup de matériel différent est très compliqué, et peut être en désaccord avec la stratégie du groupe auteur.

Ainsi, les incompatibilités entre logiciel et ordinateurs précis viennent de ce genre de problèmes. Il n'y a pas de bonne solution : être très compatible a un coût, l'être peu revient à se couper de certaines utilisatrices… voire se faire taxer d'obsolescence programmée.

3. Là encore, on voit que le sens est un épiphénomène : en quoi le fichier est le même une fois qu'il a été déplacé ? Parce que les couches logicielles supérieures se comportent comme telles.

Orchestration

Même en supposant le problème de l'abstraction résolu, on se demande vite comment faire tourner plusieurs logiciels en même temps, et s'assurer que les différents logiciels ne se volent pas des ressources — par exemple, que deux logiciels écrivent dans la même zone de la mémoire, ou qu'un reçoive des lettres tapées destinées à un autre (imaginez la situation : n'importe quel site xeb doit pouvoir recevoir du texte, mais il faut empêcher un site web d'écouter quand ce n'est pas son tour, sinon, il doit pouvoir tout voler).

Ainsi, se pose un problème d'orchestration : s'assurer que chaque composant logiciel ne fasse que ce qu'il a le droit de faire. On rajoute donc une couche logicielle pour ça. On va présenter deux modèles, l'un très permissif, l'autre, très coercitif.

les machines LISP dans une machine LISP, chaque logiciel est une *fonction*, c'est à dire une machine qui transforme ses entrées en sorties. On peut à chaque instant, inspecter et modifier chaque fonction — imaginez que vous puissiez changer le comportement de chaque entrée de menu dans chaque logiciel que vous utilisez. Le modèle fonctionne parce qu'on suppose que c'est la même personne qui a écrit (ou a tout le moins relu) toutes les fonctions.

Une fonction est censée régulièrement demander si elle peut continuer ou au contraire, passer la main à une autre fonction, et peut appeler chaque autre fonction.

iOS les iPhone fonctionne à l'opposé : chaque application a le droit d'accéder uniquement à certaines ressources (et doit se justifier auprès du constructeur et demander explicitement l'autorisation à l'utilisateur) ; de plus, elle peut se faire arrêter à chaque instant si le logiciel central estime qu'elle prend trop de certaines ressources (batteries, calcul,…)

Système d'exploitation

La première fonction d'un *système d'exploitation* est de fournir une série d'abstraction cohérentes pour les logiciels. La liste des abstractions fournies par les systèmes d'exploitation a tendance à croître avec le temps, par exemple :

— communication avec les périphériques
— systèmes de fichiers
— réseau
— gestion de bibliothèques multimédia
— gestion de langues multiples
— position (calculée par GPS, triangulation WiFi,…)
— méthode de paiement
— …

De ce fait, il y a de moins en moins de systèmes d'exploitation.

La deuxième fonction d'un système d'exploitation est de protéger l'accès aux différentes ressources entre les logiciels eux-mêmes, et vis-à-vis de l'utilisateur. Plusieurs stratégies existent, plus ou moins contraignantes.

Parfois la situation peut devenir très étrange, avec des redondances. Considérons un navigateur web : c'est un logiciel qui transforme une description de pages web en une représentation graphique. À ce titre, un navigateur doit dessiner des rectangles, des lettres, des boutons,… Il utilise donc des bibliothèques de fonctions pour ce faire. Mais avec le temps, l'affichage de pages web a permis de faire des choses de plus en plus compliquées : on s'est rendu compte que c'était une bonne méthode pour fournir des petits logiciels qui puissent fonctionner sur tous les systèmes d'exploitation (bureau comme téléphone). Aujourd'hui, on peut considérer que le développement web est un système d'exploitation à part entière, qui fournit des abstractions aux pages web.

3.3 Le cloud

Du point de vue le plus terre-à-terre, le *cloud* n'est rien d'autre qu'un extension d'un système d'exploitation : on rajoute une mémoire en plus (située à distance), et on change le sens de l'abstraction de ce qu'est un fichier. En effet, il n'est plus nécessairement situé sur une mémoire locale, mais peut être ailleurs, et même se déplacer (ainsi, les photos que je regarde rarement ne sont plus physiquement sur mon téléphone, mais je n'ai aucun moyen de savoir qu'elles n'y sont pas : l'abstraction du fichier suit ses mouvements).

De même, certains calculs peuvent être faits à distance. D'une certaine manière, une solution de cloud (iCloud, Google Drive, Dropbox,…) est une extension du système d'exploitation — ce qui arrive tout le temps. Ce qui est problématique est que tous les problèmes (de contrôle, de coût énergétique,…) sont démultipliés. En effet, non seulement le fournisseur des couches d'abstraction dont je dépend peut avoir des intérêts qui ne sont pas les miens, mais leur fonctionnement est orchestré en temps réel par des ordinateurs que je ne contrôle pas : je n'ai même plus la possibilité de garder mon ancienne version le temps de transitionner.

Cela pose aussi des problèmes de droit : quel est le droit applicable aux données dont le lieu n'est pas clair ?

4

Format de fichiers

On voit ce qu'on appelle un format de fichier, et comment ils sont à la fois conventionnels et sémantiques.

De même, des conventions doivent être appliquées pour les formats de fichiers.

Un *format de fichier* est une série de convention sur la manière dont doit être lu un fichier. Par exemple, pour lire une vidéo, on indiquera où sont situées les différentes pistes son et vidéo, ainsi que les sous-titres éventuels, et des indications pour les synchroniser. Pour un format de texte enrichi, il indiquera comment choisir quelles polices doivent être utilisées, où sont stockées les images incluses,…

Il y a énormément de formats de fichiers, qui sont créés en permanence. Ils diffèrent par leurs objectifs :

— prendre le moins de place possible ;
— qu'il soit facile de travailler dessus ;
— être plus sémantique ;
— être plus accessible ;
— être adapté au partage ;
— être adapté à l'archivage de longue durée.

4.1 Quelques formats de documents principalement textuels

Word pré-docx

Historiquement, le format de Word (ce qu'on appelle en général le doc) était optimisé pour la facilité de calcul : en effet, à l'époque, les ordinateurs n'avaient pas une mémoire de travail capable de contenir un document en entier. Un document Word commençait donc par un répertoire (comme pour un système de fichier) décrivant les positions sur la mémoire longue durée des différents segments du document (pages ou groupes de pages). Puis, on chargeait dans la mémoire de travail exclusivement la page en cours d'édition et les paramètres globaux (s'il y en a — par exemple le contenu des en-têtes,…).

Ainsi, le document enregistré était une représentation à l'identique du contenu de la mémoire de travail : les conventions internes de la manière dont cette version précise du logiciel préférait stocker ses données étaient reflétées dans le format de fichier, ce qui explique pourquoi chaque mise-à-jour créait un nouveau format (ou chaque version : Word pour Mac n'était pas totalement compatible avec Word pour DOS/Windows).

Le format permettait à la fois de suivre des feuilles de style et de faire des modifications de mise-en-page locales, rendant les documents extrêmement complexes.

Office Open XML, OpenDocument Format

La taille des mémoires de travail ayant augmenté, on a pu donner des formats plus stables. La plupart des traitements de texte aujourd'hui utilise un format documenté décrivant la structure du document. C'est beaucoup plus portable, mais le format peut être très complexe : le texte de référence du format Office Open XML fait plus de 5000 pages, donc n'importe qui peut lire et écrire de tels documents, en étant prêt à y passer beaucoup de temps.

Le format permettait à la fois de suivre des feuilles de style et de faire des modifications de mise-en-page locales, rendant les documents extrêmement complexes.

OpenOffice a un format similaire.

Rich Text Format

Un format conçu pour le partage. Permet d'inclure des images, de faire de la mise en page légère,…

Il ne supporte que des modifications de mise-en-page locales, pour rester léger.

HTML

Le *Hyper-Text Markup Language* est le format des pages Web. Avec le temps, est devenu excessivement complexe. Se présente comme la déclaration d'un arbre contenant des éléments (qui peuvent être du texte, des vidéos,…). Chaque élément est d'une classe dont le style est défini par ailleurs : c'est ainsi que toutes les pages d'un même site Web se ressemblent, et parfois changent intégralement de style du jour au lendemain. On a séparé le fond de la forme.

Exemple 2. Sur le site du Monde, le fragment suivant, écrit en HTML :

```
<section id="en-continu" class="en-continu js-is-hide ">
  <li class="New">
  <a href="2020/10/26/le-japon-souhaite-atteindre-la-neutralite-carbone-d-ici-2050.html
  <div class="New__header">
    <div class="New__time">08:20</div>
  </div>
  <div class="New__content">Japon : vers la neutralité carbone d'ici 2050</div>
  </section>
```

définit un lien vers `https://lemonde.fr/2020/10/26/le-japon-souhaite-atteindre-la-neutralite-` `html` qui s'affiche sous la forme d'un texte présenté selon le style `New__content`, précédé de l'heure de parution de l'article présenté dans le style `New__time`, le tout présenté dans le style `New`.

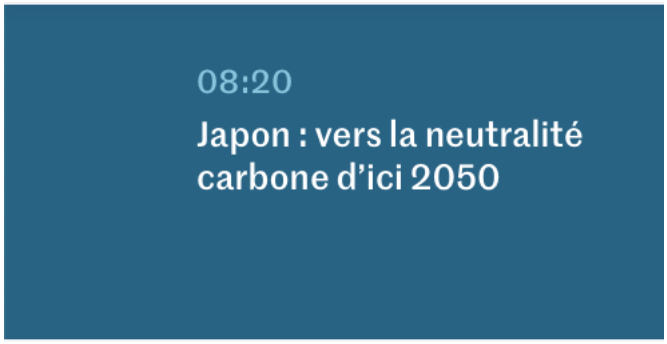
Ces différents styles sont définis dans un autre document, la *feuille de style* par :

```
.New {
  font-family: "Marr Sans",Helvetica,Arial,Roboto,sans-serif;
  background-color: #036487;
}

.New__time {
  color: #79c4df;
}

.New__content {
  color: #fff;
}
```

qui définit donc les couleurs et les polices qui doivent être utilisées. Le navigateur Web interprète ce document et produit



Le format a vogué à la dérive : les navigateurs Web font tout pour interpréter tout ce que les concepteurs de site Web font, même si c'est incorrect. Ainsi, le format n'est pas facile à écrire, pas facile à afficher,…

Portable Document Format

Le PDF est un format avant tout fait pour garantir que l'affichage soit le même partout — en particulier pour une impression. Ainsi, un PDF pourra inclure ses propres polices (pour pouvoir s'afficher même si les polices ne sont pas présentes).

La norme historique fait un millier de pages, depuis il y a eu de nombreuses extensions, pour supporter l'affichage de longue durée, la signature électronique, un peu d'interactivité…

Un très bon format pour l'archivage et la diffusion, mais pas pour le travail (un PDF est un document fini).

Markdown, org-mode

Enfin, il existe des formats de fichiers extrêmement légers. L'exemple le plus connu et élégant se nomme Markdown. La norme Markdown tient en quelques pages. Tout y est au format texte. Ainsi, un document Markdown va ressembler à

```
# Titre 1
## Sous-titre 1
- liste
- liste
- /italique/
- *gras*
```

Toutes les indications de structure et de mise-en-page sont placées au moyen de balises textuelles très légères. L'idée est que le texte brut est censé être lisible par un humain et facile à transformer en un autre format (HTML, PDF) plus apte à la consommation.

C'est donc un format de travail, et aussi un format d'archivage.

TEX, LATEX

TEX est un format de préparation de documents beaucoup plus lourd. Il a été le premier logiciel permettant une mise-en-page professionnelle (du niveau de ce que des ouvriers spécialisés faisaient dans les imprimeries), en particulier pour tout ce qui est gestion des césures, des lignes orphelines,…

C'est un format purement textuel qui ne fait aucune hypothèse sur la structure du document.

LATEX est un jeu de macros pour TEX permettant de décrire la structure d'un document. TEX est le seul logiciel que je connaisse qui soit attaché à une compatibilité parfaite : un document produit en 1982 est censé être produit à l'identique par un compilateur TEX d'aujourd'hui, et vice-versa.

Ce sont des formats de travail et d'archivage.

Ce document

Ce document est écrit en `org-mode` (qui peut ressembler à du markdown). Il est ensuite traduit en LATEX est compilé en PDF. Certaines parties sont directement en TEX (par exemples, les schémas). Le fichier source du cours (sur lequel je travaille et que j'archiverai et que je suis sûr de pouvoir réutiliser à l'identique dans quarante ans) est disponible en `https://lacl.fr/~lpellissier/culture.org`

4.2 D'autres formats

Il y a des formats spécialisés, avec un seul objectif en tête. Citons trois de ces objectifs.

Le versionnage

On peut vouloir un format qui se souvienne de l'historique d'un fichier, ce qui permet à la fois de faire des modifications sans prendre de risques, de pouvoir retourner en arrière, ou, dans des contextes où plusieurs personnes travaillent sur le même fichier, de mélanger les mises-à-jour que chacun à fait.

Cela peut se faire au niveau du système de fichier, en changeant la notion de fichier : on peut considérer que le répertoire, au lieu de déclarer seulement la position de début et la position de fin de chaque fichier, peut contenir la position de début et la position de fin de chaque version, et permette de passer de l'un à l'autre (c'est la solution qu'on adopté des logiciels comme DropBox ou TimeMachine). Cela a le mérite de marcher uniformément pour tous les fichiers, mais pas de permettre de mélanger les mises-à-jour faites par des personnes différentes.

On peut aussi stocker les modifications à l'intérieur-même du fichier, au moyen de symboles appropriés. Par exemple, je peux inventer un format de fichier texte qui au lieu de contenir le fichier, va contenir des modifications. Le fichier commence normalement, dans une version 0 :

```
Ceci est un fichier.
```

```
De trois lignes.
Après cette ligne, il est fini.
```

Puis on le modifie, en le fichier suivant :

```
Ceci est un fichier.
De trois lignes plus ou moins longues.
Après cette ligne, il est fini.
```

On va représenter ça de la manière suivante :

```
Ceci est un fichier.
>>>> 1
De trois lignes plus ou moins longues.
====
De trois lignes.
<<<<
Après cette ligne, il est fini.
```

Indiquant que la révision 1 a remplaçé la ligne «De trois lignes.» par «De trois lignes plus ou moins longues.» Ainsi, on garde un historique de toutes les transformations qu'a subi le contenu du fichier. On peut choisir de n'afficher que la dernière version, ou n'importe laquelle, et on peut aussi comparer des modifications, faites par des personnes différentes, pour pouvoir les fusionner.

Un tel format de fichier n'est fait que pour des manipulations autour du concept de modifications, et peut contenir, comme substrat, un autre format.

Compression

On peut vouloir prendre le moins de place possible. On dit d'un format qui est fait dans cette optique que c'est un format de *compression*. L'idée de la compression est de repérer des régularités des données et de représenter les données redondantes une seule fois. Par exemple, supposons qu'on ait un texte écrit en français. Chaque caractère est codé sur 7 bits.

On peut décider de coder chaque caractère sur 8 bits en rajoutant un 0 devant leur code. Cela fait qu'on a, d'un coup 128 autres possibilités pour coder : en effet, on peut coder 256 caractères sur 8 bits, mais 126 sont déjà pris par les caractères normaux. En faisant ça, on augmente la longueur du texte d'un huitième. On peut ensuite repérer les suites de caractères les plus courantes, et décider de les encoder comme un nouveau caractère. En français, les deux chaînes de caractères les plus courantes sont `er` et `st`. Supposons qu'on les code respectivement par `10000000` et `10000001`.

Le mot `mer` sans compression, était codé par `1101101 1100101 1110010` Tandis qu'après compression, il sera codé par `01101101 10000000` Et le document commencera par une table de décodage, présentant les codes spécifiques au document (en l'occurrence, ici, que les caractères sont codés sur huit bits, que ceux commençant par un zéro suivent le code ASCII usuel pour les sept derniers caractères, et que `er` est codé par `10000000` et `st` par `10000001`).

Ce principe est celui d'un algorithme de compression, le *Byte-pair encoding*. Tous repèrent les régularités d'un fichier, et ne représentent qu'une fois chaque redondance.

Code de correction d'erreurs

Inversement, il y a d'autres formats de fichier faits pour rajouter des redondances. Par exemple, les codes correcteurs d'erreur. Ils sont faits pour représenter la même information de plusieurs manières, de manière à la rendre moins sensible aux erreurs de transmission.

Les adresses, par la poste, utilisent un tel code : le code postal et le nom de la commune sont des informations redondantes, néanmoins, on écrit les deux. Si jamais une des deux informations est mal écrite, illisible, voire fausse, on peut reconstituer l'information correcte à partir de l'autre.

Les codes correcteurs d'erreur sont en particulier utilisés dans tous les dispositifs de lecture optique, comme les CD : un lecteur CD fait énormément d'erreur en lisant un CD (car on pousse la technologie à ses limites). On écrit donc sur un CD une information fortement redondante au moyen d'un code correcteur d'erreur, permettant au lecteur de corriger les erreurs faites par la tête de lecture.

5

Cryptographie

On introduit rapidement la cryptographie, en particulier asymétrique, et deux applications : la signature numérique et le vote électronique.

Depuis longtemps, on a voulu transmettre des messages secrets, c'est-à-dire des messages ayant les trois propriétés suivantes :

fidélité le message est fidèle au message que la personne ayant écrit souhaite avoir transmis ;

confidentialité seul·es les destinataires du message peuvent le lire ;

authenticité les destinataires peuvent se convaincre que l'émetteur du message est bien la personne qu'elle prétend être.

Les techniques pour faire de tels messages se divisent en gros en deux familles. La *stéganographie* cache l'existence même d'un message, en le dissimulant sous un message anodin : ainsi, on prend les initiales de chaque vers d'un poème pour obtenir une autre phrase,… La *cryptographie* au contraire, rend le message illisible : tout le monde peut voir qu'il y a là un message caché, mais pas forcément le lire.

Parmi les techniques cryptographiques, les plus simples utilisent des noms de code : c'est-à-dire que l'émetteur et le destinataire se sont mis d'accord, à l'avance, sur le sens qu'ont certains messages codés. Ainsi, le *colonel Berthier* désigne Jean-Pierre Vernant et l'*abbé Pierre* Henri Grouès, et toutes les personnes dans la confidence savent décrypter.

Cela nécessite d'avoir déjà décidé, avant de transmettre le message, tous les messages possibles que l'on pourrait vouloir transmettre, et en avoir trouvé une version secrète : on se retrouve soit à se limiter à peu de messages, soit à devoir inventer une langue à part entière [1].

Une autre possibilité et de donner un procédé systématique pour *chiffrer* n'importe quel message, accompagné d'un autre procédé permettant de le déchiffrer. C'est ce qui va nous intéresser ici.

5.1 Chiffrement symétrique

Une telle technique de chiffrement existe depuis au moins 2000 ans, connue sous le nom de *code César*. Les deux personnes souhaitant communiquer (appelons-les Alice — que nous noterons parfois A

1. On pensera à l'armée américaine utilisant des Navajos (dont la langue était rare et que l'ennemi ne parlait pas) comme opérateurs radio pour communiquer secrètement.

— et Bob — que nous noterons B) choisissent d'abord un nombre secret, une *clef*, que nous noterons K.

Quand Alice veut envoyer un message à Bob, il lui suffit de décaler chaque lettre de son message de la valeur de la clef. De même, Bob décale à l'envers de la valeur de la clef pour réobtenir le message initial.

Exemple 3. Alice et Bob ont choisi la clef 3. Alice veut envoyer le message «Bonjour» à Bob. Elle considère donc chaque lettre de son message :

1. B est la deuxième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la cinquième lettre de l'alphabet : E.
2. o est la quinzième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la dix-huitième lettre de l'alphabet : r.
3. n est la quatorzième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la dix-septième lettre de l'alphabet : q.
4. j est la dixième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la treizième lettre de l'alphabet : m.
5. u est la vingt-et-unième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la vingt-quatrième lettre de l'alphabet : x.
6. r est la dix-huitième lettre de l'alphabet. Elle la décale de trois lettres, et obtient la vingt-et-unième lettre de l'alphabet : u.

Ainsi, elle écrit «Erqmxu» à Bob.

Comme Bob connaît la clef, il lui suffit d'appliquer la transformation inverse pour retrouver le message de départ.

N'importe qui écoutant aux portes ne pourra que lire «Erqmxu» sans savoir ce que cela veut dire.

On appelle ce monde de chiffrement un *chiffrement symétrique*, car la même clef est utilisée pour chiffrer et déchiffrer.

Ce mode de chiffrement est toutefois facile à attaquer : en effet, le texte ainsi obtenu a des caractéristiques héritées du texte de départ. Supposons que le texte soit écrit en français. On sait qu'en français la lettre la plus courante est le «e». On peut regarder la lettre la plus courante dans le message chiffré, faire l'hypothèse que c'est la version chiffrée du «e», et en déduire la clef. Dès que le message est long, ce genre d'hypothèse devient très souvent vérifié.

Pour contrer ça, on peut rallonger la clef, et décider qu'une lettre sur deux sera chiffrée avec une clef, et une lettre sur deux avec une autre. Néanmoins, on peut montrer que, pour que le chiffrement symétrique résiste à ce genre d'attaques statistiques, la clef doit être du même ordre de grandeur que le message.

5.2 Chiffrement asymétrique

Un autre type de chiffrement est le *chiffrement asymétrique*, dans lequel on utilise deux clefs différentes, une pour chiffrer, une pour déchiffrer. L'algorithme de chiffrement asymétrique le plus courant se nomme RSA. Il repose sur une hypothèse mathématique qu'il existe des opérations qui ont un sens dans deux sens, sont faciles à faire dans un sens, mais très difficiles à faire dans l'autre [2]. Dans ce cas, les clefs vont par paires, et ce que chiffre l'une est déchiffré par l'autre et vice versa.

À grands traits, le chiffrement asymétrique fonctionne de la manière suivante :

2. Pour faire une analogie, ça a du sens de mélanger deux tubes de peinture, et c'est plutôt facile, mais c'est très dur de séparer les couleurs une fois mélangées. On a de bonnes raisons de croire que certains problèmes arithmétiques ont une telle propriété, mais aucune preuve.

1. Alice commence par choisir une paire de clefs. Il est relativement facile de générer une paire de clefs allant bien ensemble, mais il est très dur de deviner quelle est l'autre clef quand on n'en a qu'une.
2. Alice envoie une de ses deux clefs à tout le monde. On appellera cette clef la *clef publique d'Alice*. Elle garde l'autre secrète, ce sera sa *clef privée*.
3. Quand Bob veut envoyer un message à Alice, il la chiffre avec la publique d'Alice. Ainsi, seule Alice (avec sa clef privée) pourra la déchiffrer.

De même, si Alice veut envoyer un message à Bob, elle le chiffre avec la clef publique de Bob, qui est le seul à pouvoir déchiffrer, avec sa propre clef privée.

5.3 Signature numérique

Ce dispositif peut aussi être utilisé pour faire une *signature*. En effet, supposons qu'Alice chiffre un message avec sa clef privée. N'importe qui peut le déchiffrer (avec la clef publique d'Alice). Mais si elle communique le message en clair et le message chiffré, alors tout le monde peut vérifier qu'elle est bien l'autrice du message : en effet, seule sa clef privée peut avoir chiffré le message d'une telle manière qu'il soit déchiffré par sa clef publique.

Cela permet d'authentifier des documents, mais aussi des votes,…

5.4 Faire confiance à des clefs ?

On voit que tout un tas de problèmes (de secret et d'authentification) sont ramenés au problème de se convaincre qu'une clef publique appartient bien à la personne qui prétend en être l'émettrice : en effet, on sait que le message ne peut être lu que par la personne possédant une certaine clef privée, ou provient de cette personne… Mais rien ne dit que ce soit la personne que l'on croit !

Il n'y a pas de solution technologique à ce problème, la technologie a juste — sous une hypothèse mathématique forte — ramené tout le problème de la confiance à la confiance en une clef publique. Mais les seuls dispositifs permettant de prouver que cette confiance est bien placée sont des dispositifs sociaux (de notarisation, d'obligations légales,…).

Bibliographie

Xavier Leroy (avr. 2019). *Le logiciel, entre l'esprit et la matière*. T. 284. Leçons inaugurales du Collège de France. Fayard.

Index

Table des matières