Licence de Sciences Economiques et de Gestion

Option : Mathématiques pour l'informatique

Joëlle Cohen

19 mars 2009

Table des matières

Chapitre 1

Ensemble et fonctions

Le but d'un programme sera de manipuler des *objets* en exécutant des *actions*. Une abstraction de ces deux termes mènent à définir les notions d'*ensembles* pour les objets et de *fonctions* pour les actions.

1.1 Définitions

1.1.1 Notion d'ensembles

Une collection d'objets est un ensemble s'il existe un critère permettant de savoir si un objet donné appartient à cette collection ou non.

Mais attention ce critère peut être quelconque.

***exemple* 1.1** : Soit L_1 l'ensemble des étudiants inscrits en premiŕe année de Licence de Sciences Economiques et de Gestion.

Soit L_1' l'ensemble des élèves de terminale qui s'inscriront en première année de Licence de Sciences Economiques et de Gestion.

On peut donner les éléments de L_1 mais pas ceux de L_1'.

□

Si x est dans l'ensemble E on note $x \in E$ (x appartient à E ou x est un élément de E).

Remarquons qu'il existe un ensemble sans aucun élément appelé ensemble vide et noté $\emptyset$.

On peut construire de nouveaux ensembles par certains procédés bien connus.

1.1.2 Produit cartésien

Le produit cartésien repose sur la notion de *couple* : si x et y sont 2 objets on construit l'objet (x, y) tel que $(x, y) = (x', y')$ *si et seulement si* $x = x'$ et $y = y'$.

En couplant (y, z) à x, on construit un triplet $(x, (y, z))$ et on peut ainsi construire des n-uplets $(x_1, (x_2, (\dots (x_{n-1}, x_n) \dots)))$ pour un entier n quelconque.

Définition 1.1 *Le produit cartésien de deux ensembles E et F est l'ensemble des couples (x, y) avec $x \in E$ et $y \in F$. Il est noté $E \times F$.*

1.1.3 Opérations ensemblistes

Définition 1.2 *L'union de deux ensembles E et F est l'ensemble des éléments qui sont dans E ou dans F. Elle est notée $E \cup F$.*

Définition 1.3 *L'intersection de deux ensembles E et F est l'ensemble des éléments qui sont dans E ou dans F. Elle est notée $E \cap F$.*

Définition 1.4 *Si tous les éléments d'un ensemble A sont dans l'ensemble E alors on dit que A est une partie de E (on dit aussi sous-ensemble). On note $A \subset E$.*

Définition 1.5 *Si A est une partie de l'ensemble E alors le complémentaire de A dans E noté $\overline{A}$ est le sous-ensemble de E dont les éléments sont les éléments de E qui ne sont pas dans A.*

L'ensemble des parties de E est noté $\mathcal{P}(E)$. On remarquera que $\mathcal{P}(E)$ contient E et $\emptyset$.

1.1.4 Fonctions

De façon informelle une fonction d'un ensemble E dans un ensemble F est une règle qui associe à un élément de E au plus un élément de F. Nous allons donner une définition plus ensembliste lié à la notion de graphe d'une fonction.

Définition 1.6 *Une fonction f de E dans F est donnée par une partie G_f de $E \times F$ telle que pour tout $x \in E$ il existe au plus un élément $y \in F$ tel que $(x, y) \in G_f$. G_f est le* graphe *de f.*

Si $(x, y) \in G_f$ alors on notera $y = f(x)$ et on dira que x a pour image y par la fonction f ou que x est un antécédent de y par la fonction f.

On appelle *domaine* de f, noté $dom(f)$, la partie de E qui contient tous les éléments x pour lesquels il existe un $y \in F$ tel que $(x, y) \in G_f$.

On dit que la fonction f de E dans F est une *application* si $dom(f) = E$.

On appelle *image* de f, noté $im(f)$, la partie de F qui contient tous les éléments y pour lesquels il existe un $x \in E$ tel que $(x, y) \in G_f$.

Définition 1.7 *Une application f de E dans F est*
- *injective si, lorsque $f(x) = f(x')$ alors $x = x'$ (tout élément de F a au plus un antécédent),*
- *surjective si $im(f) = F$ (tous les éléments de F ont un antécédent),*
- *bijective si f est injective et surjective.*

On comprend vite que pour qu'une application soit bijective il faut que les deux ensembles E et F soient « de même taille ».

On va préciser cette notion.

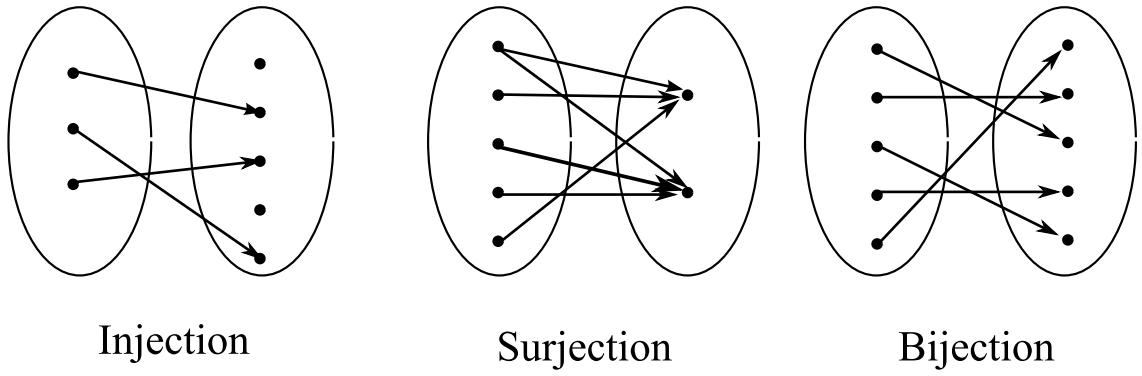


Fig. 1.1 – Injection-Surjection-Bijection

1.2 Ensembles dénombrables

Définitions

Définition 1.8 *Deux ensembles E et F sont* équipotents *s'il existe une bijection de E dans F. On le note $E \approx F$.*

Définition 1.9 *Un ensemble E est* dénombrable *s'il existe une injection de E dans $\mathbb{N}$.*

Un ensemble E est fini *s'il est équipotent à une partie de $\mathbb{N}$ de la forme $\{1 \dots n\}$ pour un entier n. Cet entier est alors le* cardinal *de E, noté $|E|$ ou $card(E)$.*

Il est clair qu'une partie d'un ensemble dénombrable est dénombrable et qu'une partie d'un ensemble fini est fini et de cardinal inférieur.

opérations sur les ensembles

Théorème 1.1 $E \times F \approx F \times E$

$E \times (F \times G) \approx (E \times F) \times G$

On note alors $E \times (F \times G)$ par $E \times F \times G$

On peut alors définir des fonctions d'un produit cartésien $E_1 \times \cdots \times E_n$ dans un ensemble F : on parle de fonction *n-aire* ou à n variables ou d'*arité n*.

Propriété 1.2 *Soient deux ensembles finis E et F. L'ensemble $E \times F$ est fini et de cardinal $|E| \times |F|$.*

Pour deux ensembles dénombrables, le résultat sera une conséquence du théorème suivant :

Théorème 1.3 $\mathbb{N}^2$ *est dénombrable.*

preuve : considérons l'application $\mathbb{N} \times \mathbb{N} \longmapsto \mathbb{N}$ qui à tout couple (n, m) associe $2^m(2n+1) - 1$.

C'est une injection car si $2^m(2n+1) - 1 = 2^{m'}(2n'+1) - 1$ alors on a $2n+1 = 2^{m'-m}(2n'+1)$ avec $m' \geq m$.

Mais $2n+1$ est un nombre impair alors que si $m' > m$, le nombre $2^{m'-m}(2n'+1)$ est pair ; on en déduit que $m' = m$ puis que $n' = n$.

C'est une surjection : soit p un entier. On considère trois cas :

– $p = 0$ alors $p = 2^0(2.0 + 1) - 1$
– p est impair alors $p + 1$ est pair et soit m l'exposant de la plus grande puissance de 2 qui divise $p + 1$; par définition de m il existe un entier impair de la forme $2n + 1$ tel que $p + 1 = 2^m(2n + 1)$
– p est pair alors $p + 1$ est impair et s'écrit $2n + 1$ donc $p = 2^0(2n + 1) - 1$

■

On peut ainsi montrer que $\mathbb{Z}$ est dénombrable en établissant l'injection suivante

$$\begin{cases} \mathbb{Z} & \to & \mathbb{N} \times \mathbb{N} \\ n & \longmapsto & (n, 0) \ si \ n \geq 0 \\ n & \longmapsto & (0, -n) \ si \ n < 0 \end{cases}$$

Puis on montre que $\mathbb{Q}$ est dénombrable par l'injection suivante

$$\begin{cases} \mathbb{Q} & \to & \mathbb{Z} \times \mathbb{N} \\ \frac{a}{b} & \longmapsto & (a, b) \text{ si } a \neq 0 \\ 0 & \longmapsto & (0, 0) \end{cases}$$

où $\frac{a}{b}$ est la représentation d'un rationnel sous la forme d'une fraction irréductible avec $a \in \mathbb{Z}^*$ et $b \in \mathbb{N}^*$.

Par contre $\mathbb{R}$ n'est pas dénombrable.

En effet si tel était le cas, on pourrait numéroter les éléments de $\mathbb{R}$ en particulier l'intervalle $[0, 1[$ et on aurait $[0, 1[= \{a_1, a_2, \ldots\}$.

Construisons alors le réel z dans $[0, 1[$ par $z = 0, x_1x_2 \ldots$ avec x_1 différent de la première décimale de a_1, ainsi $z \neq a_1$, puis x_2 différent de la deuxième décimale de a_2, ainsi $z \neq a_2$, ... On aboutit donc à une contradiction puisque $z \in [0, 1[$ mais $z \notin \{a_1, a_2, \ldots\}$.

Propriété 1.4 *Soient E et F deux ensembles dénombrables alors $E \times F$, $E \cup F$ et $E \cap F$ sont dénombrables.*

Puisque toute une partie d'un ensemble dénombrable est dénombrable, il est clair que le complémentaire dans un ensemble dénombrable est dénombrable.

Ensembles d'applications

Considérons maintenant l'ensemble des fonctions entre deux ensembles.

Théorème 1.5 *Soient E et F deux ensembles finis. L'ensemble des applications de E dans F est fini de cardinal $|F|^{|E|}$. Il est noté F^E.*

preuve : soit n le cardinal de E, on pose $E = \{x_1, \ldots, x_n\}$. Définir une application de E dans F revient à donner un n-uplet $(y_1, \ldots, y_n)$ où chaque y_i est élément de F et image de x_i. On a donc une bijection entre l'ensemble des applications de E dans F et $\underbrace{F \times F \times \cdots \times F}_{n \ termes}$. ■

Théorème 1.6 *Soit E un ensemble fini. L'ensemble $\mathcal{P}(E)$ des parties de E est fini de cardinal $2^{|E|}$.*

preuve : on établit une bijection entre $\mathcal{P}(E)$ et $\mathbf{B}^E$ où $\mathbf{B}$ est l'ensemble $\{0, 1\}$ en associant à chaque partie X de E sa fonction caractéristique notée $\mathbf{1}_X$ définie par $\mathbf{1}_X(x) = 1$ si $x \in X$ et $\mathbf{1}_X(x) = 0$ sinon. D'après le théorème 1.5, l'ensemble $\mathbf{B}^E$ est de cardinal $|B|^{|E|} = 2^{|E|}$ ■

Maintenant ce résultat s'étend-il aux ensembles non finis dénombrables ? La réponse est non et on a plus précisément :

Théorème 1.7 *Soient E un ensemble à au moins deux éléments. L'ensemble $E^{\mathbb{N}}$ des applications de $\mathbb{N}$ dans E n'est pas dénombrable.*

preuve : supposons que $E^{\mathbb{N}}$ est dénombrable et on note alors $E^{\mathbb{N}} = \{f_1, f_2 \ldots\}$. On va construire une application f de $\mathbb{N}$ dans E qui ne peut pas être l'une des fonctions $\{f_1, f_2 \ldots\}$. Pour chaque entier n, on choisit $f(n) \neq f_n(n)$. On obtient ainsi une contradiction. ■

Une conséquence de ce théorème concerne l'ensemble $\mathcal{P}(E)$ des parties d'un ensemble E non fini dénombrable.

Corollaire 1.8 *Soit E un ensemble non fini dénombrable. $\mathcal{P}(E)$ n'est pas dénombrable.*

preuve : on établit une bijection entre $\mathcal{P}(E)$ et $\mathbf{B}^E$ où $\mathbf{B}$ est l'ensemble $\{0, 1\}$ en associant à chaque partie X de E sa fonction caractéristique notée $\mathbf{1}_X$ définie par $\mathbf{1}_X(x) = 1$ si $x \in X$ et $\mathbf{1}_X(x) = 0$ sinon. Le théorème 1.7 permet alors de conclure. ■

1.3 Récursion

Nous allons voir ici une autre façon de définir des ensembles et des fonctions qui diffèrent de la simple énumération des éléments d'un ensemble en ce sens que la définition sera plus *constructive* que *descriptive*.

Mais avant nous allons rappeler brièvement la notion de récurrence sur les entiers.

1.3.1 Premier principe d'induction sur $\mathbb{N}$

Théorème 1.9 *Soit $\mathcal{P}(n)$ une prédicat dépendant de l'entier n.*

***Si** les deux conditions suivantes sont vérifées*

(base) $\mathcal{P}(0)$ est vrai,

(induction) pour tout entier n, si $\mathcal{P}(n)$ est vrai alors $\mathcal{P}(n+1)$ est vrai,

***alors** pour tout $n \in \mathbb{N}$ $\mathcal{P}(n)$ est vrai.*

Une démonstration par récurrence est l'utilisation de ce théorème.

remarque : Ne pas oublier la base ; l'étape d'induction n'est pas suffisante comme le montre l'exemple suivant.

***exemple* 1.2** : : soit $\mathcal{P}(n)$ le prédicat "$8^n + 1$ est un multiple de 7". Montrons que $\mathcal{P}$ vérifie la condition d'induction.

Soit $n \in \mathbb{N}$ quelconque. On suppose que, pour cette valeur n quelconque mais fixée, $8^n + 1$ est bien un multiple de 7.

Alors on peut écrire $8^{n+1} + 1 = 8(8^n + 1) - 7$.

Par hypothèse il existe un entier k tel que $8^n + 1 = 7k$.

Donc $8^{n+1} + 1 = 8(8^n + 1) - 7 = 8 \times 7k - 7 = 7(8k - 1)$. On en déduit que $8^{n+1} + 1$ est un multiple de 7.

Mais il est faux que, pour tout $n \in \mathbb{N}$, $8^n + 1$ soit un multiple de 7. En effet, par exemple pour $n = 0$, $8^0 + 1 = 2$ n'est pas un multiple de 7. Pour $n = 3$, $8^3 + 1 = 513$ n'est pas un multiple de 7 $\cdots$ La propriété $\mathcal{P}(n)$ ne vérifie pas la condition de base.

□

Corollaire 1.10 *Soient $\mathcal{P}(n)$ un prédicat dépendant de l'entier n et un entier $n_0 \geq 0$.*

***Si** les deux conditions suivantes sont vérifées*

(base) $\mathcal{P}(n_0)$ est vrai,

(induction) pour tout entier $n \geq n_0$, si $\mathcal{P}(n)$ est vrai alors $\mathcal{P}(n+1)$ est vrai,

***alors** pour tout $n \geq n_0$, $\mathcal{P}(n)$ est vrai.*

***exemple* 1.3** : montrons que pour tout entier n, $8^n - 1$ est un multiple de 7.

Pour $n = 0$, $8^0 - 1 = 0$ est un multiple de 7.

Soit $n \geq 0$, supposons que $8^n - 1$ est un multiple de 7, $8^n - 1 = 7k$ pour un entier k. On a alors $8^{n+1} - 1 = 8(8^n - 1) + 7 = 7(8k + 1)$ et $8^{n+1} - 1$ est aussi un multiple de 7.

□

preuve du théorème 1.11 : Soit $\mathcal{P}(n)$ un prédicat vérifiant les deux conditions *base* et *induction*. Soit $X = \{p \in \mathbb{N} \mid \mathcal{P}(p) \text{ est faux}\}$. Supposons que X n'est pas vide. Alors puisque X est un ensemble d'entiers naturels non vide, X a un plus petit élément k.

$k \neq 0$ car $\mathcal{P}(0)$ est par hypothèse vrai. Donc $k \geq 1$ et $k-1$ est un entier naturel tel que $\mathcal{P}(k-1)$ est vrai, par définition de X. Or, par hypothèse, le prédicat $\mathcal{P}$ vérifie la condition d'induction et sachant que $\mathcal{P}(k-1)$ est vrai on en déduit que $\mathcal{P}(k)$ est vrai ce qui contredit l'appartenance de k à X. ■

remarque : la preuve repose sur la propriété suivante : toute partie non vide de $\mathbb{N}$ a un plus petit élément.

remarque : lorsque l'on applique un raisonnement par récurrence, il faut veiller à ne pas oublier l'étape de base comme on l'a vu plus haut mais aussi vérifier que le raisonnement est valide pour les valeurs que l'on manipule comme le montre l'exemple classique suivant

***exemple* 1.4** : on veut montrer que tous les crayons de couleur ont la même couleur. Pour cela, après avoir vérifié la propriété pour un crayon, on suppose que tout lot de n ($n \geq 1$) crayons quelconques est de couleur unique et on considère un lot de $n+1$ crayons. On numérote les $n+1$ crayons de 1 à $n+1$ et on applique l'hypothèse de récurrence aux deux lots de n crayons suivants : du numéro 1 au numéro n et du numéro 2 au numéro $n+1$. Par hypothèse, dans chaque lot tous les crayons ont la même couleur, mais le crayon numéro 2 est commun aux deux lots donc les couleurs des deux lots sont identiques et donc les $n+1$ crayons sont de la même couleur. La conclusion serait donc que tous les crayons de couleur ont effectivement la même couleur ! Le raisonnement comporte manifestement une erreur.

□

***exemple* 1.5** : montrer par récurrence sur n que pour tout $n \geq 0$, $2^{2n} - 1$ est un multiple de 3.

□

***exemple* 1.6** : montrer par récurrence sur n que pour tout $n \geq 0$, $n^3 + 5n$ est un multiple de 6.

□

1.4 Deuxième principe d'induction sur $\mathbb{N}$

Théorème 1.11 *Soit $\mathcal{P}(n)$ une prédicat dépendant de l'entier n.*

***Si** les deux conditions suivantes sont vérifées*

- *$\mathcal{P}(0)$ est vrai,*
- *pour tout entier n, si $\mathcal{P}(0) \cdots \mathcal{P}(n)$ sont vrais alors $\mathcal{P}(n+1)$ est vrai,*

***alors** pour tout $n \in \mathbb{N}$ $\mathcal{P}(n)$ est vrai.*

Les deux principes sont équivalents sur $\mathbb{N}$ et on a aussi le corollaire suivant :

Corollaire 1.12 *Soient $\mathcal{P}(n)$ un prédicat dépendant de l'entier n et un entier $n_0 \geq 0$.*

***Si** les deux conditions suivantes sont vérifées*

- *$\mathcal{P}(n_0)$ est vrai,*
- *pour tout entier $n \geq n_0$, si $\mathcal{P}(n_0) \cdots \mathcal{P}(n)$ sont vrais alors $\mathcal{P}(n+1)$ est vrai,*

***alors** pour tout $n \geq n_0$, $\mathcal{P}(n)$ est vrai.*

Ce deuxième principe permet de résoudre le type de problème suivant :

***exemple* 1.7** : on veut montrer que tout nombre naturel supérieur ou égal à 2 est un produit de nombres premiers.

Pour $n = 2$ c'est clair. Soit n un entier quelconque supérieur à 2 supposons que tout entier k, $2 \leq k \leq n$ est un produit de nombres premiers. On peut alors considérer $n+1$:

soit $n+1$ est premier et il vérifie alors la propriété,

soit $n+1$ n'est pas premier et il existe deux entiers m, p, $2 \leq m, p \leq n$ tels que $n+1 = m \times p$. L'hypothèse s'applique alors à m et p qui sont donc des produits de nombres premiers. Donc $n+1$ lui aussi est un produit de nombres premiers.

□

1.5 Induction

1.5.1 Ensemble défini inductivement

C'est une définition constructive qui donne les moyens de "construire" petit à petit l'ensemble ainsi caractérisé.

Ces moyens sont de deux types :

- un ensemble d'éléments de base
- des "constructeurs"

Ces derniers utilisent des éléments déjà connus de l'ensemble en construction pour en fabriquer de nouveaux ; les premières applications de cette construction utilisent les éléments de base.

***exemple* 1.8** : on considère l'ensemble E des suites finies non vides de 0 et de 1. $E = \{0, 1, 00, 01, 10, 11, 000, \cdots\}$. On définit inductivement le sous-ensemble X_d par

- 1 est dans X_d

– si x est dans X_d alors la suite x suivie de 0 est une suite de X_d.

Les "premiers" éléments de X_d sont 1, 10, 100, $\cdots$

□

Définition 1.10 *Soit E un ensemble. Une définition inductive d'une partie X de E est la donnée*
– *d'une partie B de E*
– *d'un ensemble $\mathcal{F}$ d'opérations sur les éléments de E.*

X est alors le **plus petit ensemble** *vérifiant*
– *(base) $B \subset X$*
– *(induction) pour toute opération $\varphi \in \mathcal{F}$ de E^k dans E avec $k \in \mathbb{N}$, pour tout $(x_1, \cdots, x_k) \in X^k$, on a $\varphi(x_1, \cdots, x_k) \in X$*

E vérifie les deux étapes de base et d'induction mais ce n'est pas nécessairement le **plus petit**. X est le plus petit sous-ensemble de E contenant B et clos pour les opérations de $\mathcal{F}$: X est donc l'intersection de toutes les parties de E vérifiant ces deux critères.

exemple 1.8 (suite) : X_d peut être défini comme suit
– (base) $1 \in X_d$
– (induction) si $x \in X_d$ alors $x0 \in X_d$

***exemple* 1.9** : dans une entreprise, la distribution des primes aux groupes suit la régle suivante : a une prime le groupe dont
– tous les employés non manager ont la note B ou
– le manager a la note B et dont un de ses sous-groupes a une prime ou
– le manager a la note M et dont tous les sous-groupes ont une prime

c'est une définition inductive de l'ensemble des groupes qui ont une prime : on doit appliquer la régle tout d'abord aux plus "petits" groupes pour pouvoir déterminer toutes les primes attribuées. La première partie de la règle est la base et les deux suivantes forment la définition purement inductive.

□

***exemple* 1.10** : soit $\mathcal{E}$ un ensemble fini appelé ensemble d'étiquettes. On va définir l'ensemble $\mathcal{AB}$ des arbres binaires étiquetés de la façon suivante :
– $\emptyset \in \mathcal{AB}$
– si $g, d \in \mathcal{AB}$ alors pour tout $x \in \mathcal{E}, \alpha = (x, g, d) \in \mathcal{AB}$. x est appelé la racine de α, g le sous-arbre gauche de α et d le sous-arbre droit de α.

Sur le schéma suivant, α_0 est l'arbre $(a, (c, \emptyset, \emptyset), (b, (c, \emptyset, \emptyset), \emptyset))$ α_1 est l'arbre $(b, (a, (b, \emptyset, \emptyset), (c, \emptyset, \emptyset)), \emptyset)$ et α_2 est l'arbre $(b, (c, \emptyset, (a, (b, \emptyset, \emptyset), \emptyset)), (a, \emptyset, (c, (a, \emptyset, \emptyset), (b, \emptyset, \emptyset))))$. Les arbres $\emptyset$ ne sont pas représentés.

□

On utilise une telle définition lorsqu'à priori, on ne sait pas énumérer les éléments de l'ensemble que l'on cherche à décrire. Mais on sait les construire :

Théorème 1.13 *tout élément de X peut s'obtenir à partir des éléments de la base en appliquant un nombre fini d'étapes inductives.*

preuve du théorème 1.13 : on définit la suite d'ensembles suivants

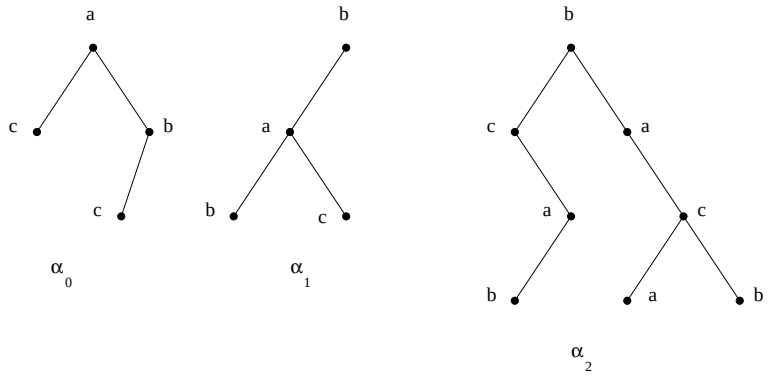


Fig. 1.2 – des arbres binaires d'étiquettes $\mathcal{E} = \{a, b, c\}$

- $X_0 = B$
- $X_{n+1} = X_n \cup \{\varphi(x_1, \cdots, x_k) \mid (x_1, \cdots, x_k) \in X_n^k \text{ et } \varphi \in \mathcal{F}\}$

On va montrer par récurrence sur n, que pour tout entier n, $X_n \subset X$.

- $X_0 = B \subset X$ par définition de X
- supposons que pour un entier n, $X_n \subset X$. Alors pour tout $(x_1, \cdots, x_k) \in X_n^k$ on a $(x_1, \cdots, x_k) \in X^k$ donc pour tout $\varphi \in \mathcal{F}$ de E^k dans E, d'après l'étape inductive $\varphi(x_1, \cdots, x_k) \in X$. Donc $X_{n+1} \subset X$.

Donc l'ensemble $X' = \bigcup_{n \in \mathbb{N}} X_n$ est inclus dans X.

Réciproquement, X' vérifie les deux conditions de la définition 1.10 donc X étant le plus petit ensemble vérifiant ces deux conditions, on a nécessairement $X \subset X'$.

Finalement $X = X'$ et X' est bien l'ensemble des éléments de E obtenus à partir de $X_0 = B$ en appliquant un nombre fini d'étapes inductives. ■

Pour pouvoir manipuler ces éléments sans tous les connaître, on doit utiliser les moyens de les construire fournis par la définition inductive. On peut ainsi définir des fonctions.

1.5.2 Fonction définie inductivement

Définition 1.11 *Soit E un ensemble et la définition inductive d'une partie X de E par la donnée*

- *d'une partie B de E*
- *d'un ensemble $\mathcal{F}$ d'opérations sur les éléments de E.*

La définition inductive d'une fonction f sur X est la donnée de :

- *$f(x)$ pour tout $x \in B$*
- *$f(\varphi(x_1, \cdots, x_k))$ en fonction des $(x_1, \cdots, x_k)$ et des $f(x_1) \cdots f(x_k)$ pour tout $\varphi \in \mathcal{F}$ de E^k dans E.*

***exemple* 1.11** : la fonction factorielle fac peut être définie inductivement sur $\mathbb{N}$ de la façon suivante

$fac(0) = 1$

$fac(n+1) = (n+1) \times fac(n)$

□

***exemple* 1.12** : (suite de l'exemple 1.8) on définit une fonction l de X_d dans $\mathbb{N}$ par

$l(1) = 0$

$l(x0) = 1 + l(x)$ pour tout $x \in X_d$.

On a par exemple $l(10000) = 4$ en calculant successivement $l(10) = 1, l(100) = 2$ et $l(1000) = 3$.

Puis on définit une fonction p de X_d dans $\mathbb{N}$ par

$p(1) = 1$

$p(x0) = 2p(x)$

On a par exemple $p(1000) = 8$ en calculant successivement $p(10) = 2$ et $p(100) = 4$.

□

On remarque facilement que $l(x)$ représente le nombre de 0 dans la suite x et que $p(x)$ est la valeur décimale du nombre binaire que la suite x représente.

***exemple* 1.13** : (suite de l'exemple 1.10) sur l'ensemble $\mathcal{AB}$, on peut définir différentes fonctions

- la taille d'un arbre
  - $t(\emptyset) = 0$
  - $t(x, g, d) = 1 + t(g) + t(d)$ pour toute étiquette x et pour tous arbres binaires g et d
- la hauteur d'un arbre
  - $h(\emptyset) = -1$
  - $h(x, g, d) = 1 + max\{h(g), h(d)\}$ pour toute étiquette x et pour tous arbres binaires g et d
- le nombre de feuilles d'un arbre
  - $f(\emptyset) = 0$
  - $f(x, \emptyset, \emptyset) = 1$ pour toute étiquette x
  - $f(x, g, d) = f(g) + f(d)$ pour toute étiquette x et pour tous arbres binaires g et d tels que $(g, d) \neq (\emptyset, \emptyset)$.

Par exemple, dans la figure 1.3, $t(\alpha_2) = 8$ (on dit que α_2 a 8 noeuds ce qui correspond au nombre de ses étiquettes), $h(\alpha_2) = 3$ (ce qui correspond au nombre de niveaux de α_2) et $f(\alpha_2) = 3$ (ce qui correspond au nombre de noeuds dont les deux sous-arbres sont vides).

□

Pour prouver de telles propriétés, on s'appuiera sur la définition inductive de l'ensemble X.

1.5.3 Preuve par induction structurelle

C'est une généralisation du principe de récurrence : on l'appelle preuve par induction structurelle.

Théorème 1.14 *Soit X un ensemble défini inductivement.*

Soit $\mathcal{P}(x)$ un prédicat dépendant de $x \in X$.

***Si** les deux conditions suivantes sont vérifiées :*

- *(base) $\mathcal{P}(x)$ est vrai pour tout $x \in B$,*
- *(induction) pour tout $\varphi \in \mathcal{F}$ de E^k dans E si pour tout $i, 1 \leq i \leq k$, $\mathcal{P}(x_i)$ est vrai alors $\mathcal{P}(\varphi(x_1, \cdots, x_k))$ est vrai,*

***alors** $\mathcal{P}(x)$ est vrai pour tout $x \in X$.*

***exemple* 1.14** : (suite de l'exemple 1.12) montrons que $p(x) = 2^{l(x)}$ pour tout $x \in X_d$.

- (base) $p(1) = 1$ et $2^{l(1)} = 2^0 = 1$

– (induction) soit $x \in X_d$, supposons que $p(x) = 2^{l(x)}$. On peut alors écrire $p(x0) = 2p(x) = 2 \times 2^{l(x)}$ par hypothèse. Mais $2 \times 2^{l(x)} = 2^{l(x)+1} = 2^{l(x0)}$ par définition de l. Donc on a bien $p(x0) = 2^{l(x0)}$ sous l'hypothèse $p(x) = 2^{l(x)}$.

On a donc montré par induction structurelle que $p(x) = 2^{l(x)}$ pour tout $x \in X_d$.

□

***exemple* 1.15** : (suite de l'exemple 1.13) montrons que pour tout arbre binaire α, on a les inégalités : $t(\alpha) \leq 2^{h(\alpha)+1} - 1$ et $f(\alpha) \leq 2^{h(\alpha)}$.

– (base)
  – $t(\emptyset) = 0$ et $2^{h(\emptyset)+1} - 1 = 2^0 - 1 = 0$
  – $f(\emptyset) = 0$ et $2^{h(\emptyset)} = 2^{-1} = \frac{1}{2}$
  – $f(x, \emptyset, \emptyset) = 1$ et $2^{h(x,\emptyset,\emptyset)} = 2^0 = 1$
– (induction) supposons la propriété vraie pour une étiquette x et deux arbres g et d. Alors
  – $t(x, g, d) = 1 + t(g) + t(d) \leq 1 + 2^{h(g)+1} - 1 + 2^{h(d)+1} - 1$. Mais $1 + 2^{h(g)+1} - 1 + 2^{h(d)+1} - 1 = 2 \times 2^{h(g)} + 2^{h(d)} - 1$ et $2^{h(g)} + 2^{h(d)} \leq 2 \times 2^{max\{h(g),h(d)\}} = 2^{max\{h(g),h(d)\}+1} = 2^{h(x,g,d)}$ donc $t(x, g, d) \leq 2^{h(x,g,d)+1} - 1$.
  – $f(x, g, d) = f(g) + f(d) \leq 2^{h(g)} + 2^{h(d)} \leq 2 \times 2^{max\{h(g),h(d)\}} = 2^{max\{h(g),h(d)\}+1} = 2^{h(x,g,d)}$

L'égalité est réalisée dans les deux cas pour les arbres *complets* c'est-à-dire dont tous les niveaux sont "remplis".

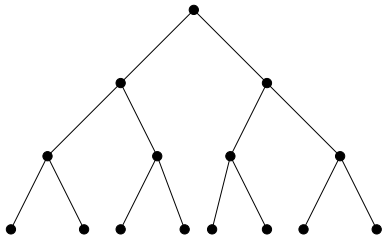


FIG. 1.3 – arbre binaire complet de hauteur 3, de taille 15 ayant 8 feuilles

□

preuve du théorème 1.14 : Soit $Y = \{x \in X \mid \mathcal{P}(x) \text{ est vrai}\}$. D'après la première condition $B \subset Y$; de plus, pour tout $\varphi \in \mathcal{F}$ de E^k dans E, pour tout $x_i \in Y, 1 \leq i \leq k$, puisque $\mathcal{P}(x_i)$ est vrai par définition de Y, $\mathcal{P}(\varphi(x_1, \cdots, x_k))$ est vrai d'après la deuxième condition : donc $\varphi(x_1, \cdots, x_k) \in Y$ par définition de Y.

Par conséquent, Y vérifie les deux conditions de la définition 1.10 de X donc $X \subset Y$. ■

Chapitre 2

Algèbre de Boole

2.1 Définition

Une *algèbre de Boole* est un ensemble non vide E dont deux éléments distincts seront notés 0 et 1 muni de trois opérations notées $+$, $\cdot$ d'arité 2 et $^-$ d'arité 1 ayant les propriétés suivantes :

- les opérations $+$, $\cdot$ sont commutatives, associatives, idempotentes et distributives l'une par rapport à l'autre,
- 0 est neutre pour $+$ et absorbant pour $\cdot$,
- 1 est neutre pour $\cdot$ et absorbant pour $+$,
- $\overline{0} = 1$ et $\overline{1} = 0$,
- $x + \overline{x} = 1$,
- $x \cdot \overline{x} = 0$.

C'est-à-dire que pour tous $x, y, z \in E$, on a

- $x + y = y + x$ *commutativité*
- $x \cdot y = y \cdot x$
- $(x + y) + z = x + (y + z)$ *associativité*
- $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
- $x + x = x$ *idempotence*
- $x \cdot x = x$
- $x + (y \cdot z) = (x + y) \cdot (x + z)$ $+$ *est distributive par rapport à* $\cdot$
- $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ $\cdot$ *est distributive par rapport à* $+$
- $x \cdot 1 = 1 \cdot x = x$ 1 *est neutre pour l'opération* $\cdot$
- $x + 0 = 0 + x = x$ 0 *est neutre pour l'opération* $+$
- $x + 1 = 1 + x = 1$ 1 *est absorbant pour l'opération* $+$
- $x \cdot 0 = 0 \cdot x = 0$ 0 *est absorbant pour l'opération* $\cdot$
- $\overline{0} = 1$ et $\overline{1} = 0$
- $x + \overline{x} = 1$
- $x \cdot \overline{x} = 0$

***exemple* 2.1** : Soit $\mathcal{P}(F)$ l'ensemble des parties d'un ensemble non vide F. $\mathcal{P}(F)$ muni de l'union, de l'intersection, de la complémentation et de $0 = \emptyset$, $1 = F$ est une algèbre de Boole.

□

2.2 Propriétés

On peut démontrer les propriétés suivantes :

Propriété 2.1 *: pour tous $x, y \in E$,*

$$\begin{aligned} x + (x \cdot y) &= x \\ x \cdot (x + y) &= x \\ \overline{x + y} &= \overline{x} \cdot \overline{y} \quad \textit{Loi de De Morgan} \\ \overline{x \cdot y} &= \overline{x} + \overline{y} \quad \textit{Loi de De Morgan} \\ \overline{\overline{x}} &= x \end{aligned}$$

preuve :

- $x + (x \cdot y) = (x \cdot 1) + (x \cdot y) = x \cdot (1 + y) = x \cdot 1 = x$
- $x \cdot (x + y) = (x \cdot x) + (x \cdot y) = x + (x \cdot y) = x$

Pour prouver les deux lois de De Morgan, on va tout d'abord donner une caractérisation de $\overline{x}$ pour un élément x quelconque de E :

Propriété 2.2 *: Soit $x' \in E$.*

$$x' + x = x + x' = 1 \textit{ et } x' \cdot x = 0 \iff x' = \overline{x}$$

preuve :

Par définition $\overline{x}$ vérifie $\overline{x} + x = x + \overline{x} = 1$ et $\overline{x} \cdot x = 0$

Réciproquement soit x' tel que $x' + x = x + x' = 1$ et $x' \cdot x = 0$.

On a alors $x' = x' \cdot 1 = x' \cdot (\overline{x} + x) = x' \cdot \overline{x} + x' \cdot x = x' \cdot \overline{x}$.

De même $\overline{x} = \overline{x} \cdot 1 = \overline{x} \cdot (x' + x) = \overline{x} \cdot x' + \overline{x} \cdot x = \overline{x} \cdot x' = x' \cdot \overline{x} = x'$ ■

Pour prouver les lois de de Morgan il suffit alors d'utiliser la propriété 2.2.

Montrons que $z = (\overline{x} \cdot \overline{y})$ vérifie $z \cdot (x + y) = 0$ et $z + (x + y) = 1$.

$(\overline{x} \cdot \overline{y}) \cdot (x + y) = (\overline{x} \cdot \overline{y} \cdot x) + (\overline{x} \cdot \overline{y} \cdot y) = 0 + 0 = 0$

$(\overline{x} \cdot \overline{y}) + (x + y) = (\overline{x} + (x + y)) \cdot (\overline{y} + (x + y)) = (1 + y) \cdot (1 + x) = 1 \cdot 1 = 1$

De même montrons que $u = \overline{x} + \overline{y}$ vérifie $u \cdot (x \cdot y) = 0$ et $u + (x \cdot y) = 1$.

$(\overline{x} + \overline{y}) \cdot (x \cdot y) = (\overline{x} \cdot (x \cdot y)) + (\overline{y} \cdot (x \cdot y) = 0 + 0 = 0)$

$(\overline{x} + \overline{y}) + (x \cdot y) = (\overline{x} + \overline{y} + x) \cdot (\overline{x} + \overline{y} + y) 1 \cdot 1 = 1$

Il reste à montrer que $\overline{\overline{x}} = x$.

En posant $w = \overline{x}$, il suffit de vérifier que x satisfait $x + w = 1$ et $x \cdot w = 0$ pour conclure que $x = \overline{w} = \overline{\overline{x}}$. ■

***exemple* 2.2** : Soient $A, B \in \mathcal{P}(F)$. On a bien $A \cup (A \cap B) = A$.

Le complémentaire de $A \cup B$ est bien l'intersection des complémentaires de A et B.

Le complémentaire de A dans F est le sous-ensemble de F qui contient tous les éléments de F qui ne sont pas dans A : c'est bien le sous-ensemble de F d'intersection vide avec A et dont l'union avec A donne F.

□

Pour simplifier l'écriture des expressions booléennes, on adopte la convention suivante : dans une écriture sans parenthèse l'opération $\cdot$ a priorité sur l'opération $+$.

Ces propriétés permettent de simplifier des expressions booléennes. Par exemple :

$$\overline{x + \overline{y} \cdot z} = \overline{x} \cdot (\overline{\overline{y} \cdot z}) = \overline{x} \cdot (y + \overline{z})$$

$$\begin{aligned} \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot \overline{z} + x \cdot \overline{y} \cdot \overline{z} + x \cdot \overline{y} \cdot z + x \cdot y \cdot \overline{z} &= \overline{x} \cdot (\overline{y} \cdot z + y \cdot \overline{z}) + x \cdot (\overline{y} \cdot z + y \cdot \overline{z}) + x \cdot \overline{y} \cdot \overline{z} \\ &= (x + \overline{x}) \cdot (\overline{y} \cdot z + y \cdot \overline{z}) + x \cdot \overline{y} \cdot \overline{z} \\ &= \overline{y} \cdot z + y \cdot \overline{z} + x \cdot \overline{y} \cdot \overline{z} \end{aligned}$$

2.3 Algèbre de Boole minimale

2.3.1 Définition

On appelle algèbre de Boole minimale, notée $\mathbb{B}$, l'algèbre de Boole réduite aux deux éléments distincts $0, 1$. Les opérations $+$, $\cdot$ peuvent être alors décrites par leur table :

$+$	0	1
0	0	1
1	1	1

$\cdot$	0	1
0	0	0
0	0	1

Dans $\mathbb{B}$, 0 et 1 représentent aussi les valeurs de vérité $Vrai$ et $Faux$ et les opérations $+$, $\cdot$,$^-$ représentent respectivement les connecteurs logiques $\vee, \wedge, \neg$ *i.e.* le *ou* logique, le *et* logique, et la *négation*.

2.3.2 fonctions booléennes

Une fonction booléenne est une fonction de $\mathbb{B}^n$ dans $\mathbb{B}^m$ avec $n, m \in \mathbb{N}$.

Théorème 2.3 *: toute fonction booléenne peut s'exprimer en fonction de ses variables et des opérations* $+$, $\cdot$, $^-$, 0, 1 *(On dit qu'elle est polynomiale).*

preuve : On raisonnera par récurrence sur le nombre d'arguments de f (son arité).

• arité 0 : f est alors une fonction à valeurs constantes donc on a soit $f = 0$ soit $f = 1$.

• soit $n \geq 0$, supposons que toute fonction booléenne d'arité n est polynomiale. Soit g une fonction booléenne d'arité $n + 1$.

On définit alors 2 fonctions booléennes f_0, f_1 par
– $f_0(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, 0)$
– $f_1(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, 1)$

On remarque alors que $g(x_1,\ldots,x_n,x_{n+1}) = \overline{x_{n+1}} \cdot g(x_1,\ldots,x_n,0) + x_{n+1} \cdot g(x_1,\ldots,x_n,1)$.

En effet si $x_{n+1} = 1$ alors $g(x_1,\ldots,x_n,x_{n+1}) = g(x_1,\ldots,x_n,1) = \overline{x_{n+1}} \cdot g(x_1,\ldots,x_n,0) + x_{n+1} \cdot g(x_1,\ldots,x_n,1)$ car $\overline{x_{n+1}} = 0$.

De même si $x_{n+1} = 0$ alors $g(x_1,\ldots,x_n,x_{n+1}) = g(x_1,\ldots,x_n,0) = \overline{x_{n+1}} \cdot g(x_1,\ldots,x_n,0) + x_{n+1} \cdot g(x_1,\ldots,x_n,1)$ car $\overline{x_{n+1}} = 1$.

Par hypothèse de récurrence f_0, f_1 sont polynomiales donc la fonction $(x_1,\ldots,x_n,x_{n+1}) \mapsto \overline{x_{n+1}} \cdot f_0(x_1,\ldots,x_n) + x_{n+1} \cdot f_1(x_1,\ldots,x_n)$ est aussi polynomiale et d'après la remarque précédente cette fonction est la fonction g. ■

***exemple* 2.3** : pour une fonction booléenne d'arité 1, on a les quatre possibilités suivantes :

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	1	0	1
1	1	0	0	1

f_1 est la fonction Identité donc $f_1(x) = x$ pour tout $x \in \mathbb{B}$.

f_2 correspond à la complémentation donc $f_2(x) = \overline{x}$ pour tout $x \in \mathbb{B}$.

f_3 est la fonction constante 0 donc $f_3(x) = 0$ pour tout $x \in \mathbb{B}$.

f_4 est la fonction constante 1 donc $f_4(x) = 1$ pour tout $x \in \mathbb{B}$.

□

Les tables de vérité donnent une description complète de toute fonction booléenne et permettent d'exprimer de telles fonctions sous la forme d'expressions booléennes mais la dimension de ces tables augmentent exponentiellement par rapport au nombre d'entrées et leur lecture devient difficile.

Ces tables correspondent tout simplement au graphe de la fonction.

***exemple* 2.4** : soit f la fonction de $\mathbb{B}^3$ dans $\mathbb{B}$ dont les valeurs sont données par la table suivante :

a	b	c	$f(a,b,c)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

En appliquant le théorème 2.3 on peut exprimer f sous forme polynomiale ainsi

$$\begin{aligned}
f(a,b,c) &= \overline{c} \cdot f(a,b,0) + c \cdot f(a,b,1) \\
&= \overline{c} \cdot (\overline{b} \cdot f(a,0,0) + b \cdot f(a,1,0)) + c \cdot (\overline{b} \cdot f(a,0,1) + b \cdot f(a,1,1)) \\
&= \overline{c} \cdot (\overline{b} \cdot (\overline{a} \cdot f(0,0,0) + a \cdot f(1,0,0)) + b \cdot (\overline{a} \cdot f(0,1,0) + a \cdot f(1,1,0))) + \\
&\quad c \cdot (\overline{b} \cdot (\overline{a} \cdot f(0,0,1) + a \cdot f(1,0,1)) + b \cdot (\overline{a} \cdot f(0,1,1) + a \cdot f(1,1,1))) \\
&= \overline{c} \cdot \overline{b} \cdot a + \overline{c} \cdot b \cdot \overline{a} + \overline{c} \cdot b \cdot a + c \cdot \overline{b} \cdot \overline{a} + c \cdot \overline{b} \cdot a
\end{aligned}$$

On peut simplifier f par $f(a,b,c) = \overline{c} \cdot \overline{b} \cdot a + \overline{c} \cdot b + c \cdot \overline{b}$.

Plus rapidement on peut remarquer que l'on a $f = 1$ sur les lignes $2, 3, 5, 6, 7$. Chaque ligne correspond à un triplet particulier de $\mathbb{B}^3$. Par exemple sur la ligne 2, on a $a = 0,\ b = 0,\ c = 1$ ce qui équivaut dans $\mathbb{B}$ à $\overline{a} \cdot \overline{b} \cdot c = 1$. Cela permet d'écrire f comme la "somme" de 5 "produits" correspondants aux 5 lignes où f vaut 1.

De façon duale, on peut écrire que $f = 0$ sur les lignes $1, 4, 8$ et en déduire que

$$f(a,b,c) = \overline{\overline{a} \cdot \overline{b} \cdot \overline{c} + \overline{a} \cdot b \cdot c + a \cdot b \cdot c} = (a + b + c) \cdot (a + \overline{b} + \overline{c}) \cdot (\overline{a} + \overline{b} + \overline{c})$$

par les lois de De Morgan.

□

De façon plus générale on considère les fonctions de $\mathbb{B}^n$ dans $\mathbb{B}^m$. Soit f une telle fonction, $f(x_1, x_2, \cdots, x_n) = y_1, y_2, \cdots, y_m$. Définissons alors les m fonctions booléennes f_i par $f_i(x_1, x_2, \cdots, x_n) = y_i$, $i = 1 \ldots m$. Chacune d'elles est polynomiale et on peut ainsi donner une expression de f d'après sa table de vérité.

***exemple* 2.5** : soit f la fonction de $\mathbb{B}^3$ dans $\mathbb{B}^3$ dont les valeurs sont données par la table suivante :

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0, 0, 0
0	0	1	1, 0, 0
0	1	0	1, 0, 0
0	1	1	1, 1, 0
1	0	0	1, 0, 0
1	0	1	1, 1, 0
1	1	0	1, 1, 0
1	1	1	1, 0, 1

Ici, f peut être décrite de cette façon : y_1 est vrai si au moins un des arguments de f est vrai, y_2 est vrai si deux arguments de f exactement sont vrais et y_3 est vrai si les trois arguments de f sont vraies. On peut donc écrire :$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3,\ x_1 \cdot x_2 \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_2 \cdot x_3,\ x_1 \cdot x_2 \cdot x_3)$.

□

Ces fonctions sont utilisées dans l'étude des circuits booléens.

2.4 Circuits logiques

Le traitement de l'information dans un ordinateur consiste à traiter, transférer, mémoriser des signaux électriques.

Dans les ordinateurs *digitaux*, l'information traitée qui est de type binaire est représentée par des signaux électriques à deux états qui peuvent être rendus très distincts l'un de l'autre.

Dans les ordinateurs *analogiques*, on utilise des signaux à variation continue. Une grandeur quelconque sera représentée par une tension, cette tension étant maintenue au cours du traitement aussi proportionnelle que possible à la grandeur représentée.

L'information digitale se transmet mieux. En effet, le moindre bruit créera une modification de l'information contenue dans un signal analogique alors qu'un signal digital ne peut être perturbé que si l'on ne peut plus distinguer l'état 1 de l'état 0.

L'information digitale se mémorise mieux : il est assez facile de réaliser des systèmes à deux états d'équilibre capables de mémoriser un bit.

L'information sera traitée à travers des *circuits* : un circuit est un ensemble électronique possédant des signaux en entrée et des signaux de sortie.

Un circuit *logique* est un circuit qui traite des signaux logiques, *i.e.* ayant deux valeurs possibles 0 et 1. On distingue les circuits logiques *combinatoires* dont les signaux de sortie ne dépendent que des signaux d'entrée et les circuits logiques *séquentiels* dont les signaux de sortie dépendent des signaux d'entrée et du *temps*. Ces derniers possèdent des éléments introduisant des délais ou des éléments mémoire.

L'étude des circuits combinatoires repose sur l'algèbre de Boole.

L'étude des circuits séquentiels repose sur la théorie des automates.

2.4.1 Portes logiques

Comme on l'a vu toute fonction booléenne est polynomiale *i.e.* s'exprime en fonction de ses variables (signaux d'entrée) et des opérateurs $+$, $\cdot$, $\bar{}$.

Ces opérateurs sont réalisés par des dispositifs électroniques appelés portes : porte OU ($+$), porte ET ($\cdot$) et porte NON ($\bar{}$). On schématise ces portes par les figures 2.1, 2.2, 2.3.

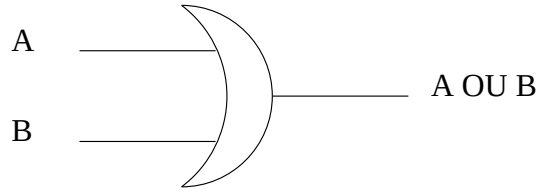


Fig. 2.1 – porte OU

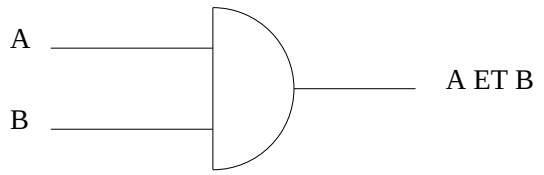


Fig. 2.2 – porte ET

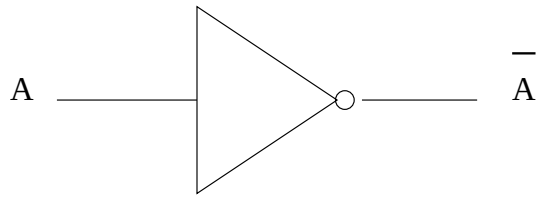


Fig. 2.3 – porte NON

On peut aussi remarquer que l'on peut limiter les trois portes ET, OU et NON à deux portes ET et NON ou bien OU et NON. En effet, par les lois de De Morgan, $a \cdot b = \overline{\overline{a} + \overline{b}}$ et $a + b = \overline{\overline{a} \cdot \overline{b}}$

2.4.2 conception

La conception des circuits logiques combinatoires repose sur les étapes suivantes :

- la génération de l'expression booléenne correspondant à la fonction désirée connue par les valeurs d'entrées et les valeurs de sortie des variables.
- la simplification de cette expression en vue d'obtenir le circuit le plus simple possible
- éventuellement la recherche d'une expression permettant de réaliser le circuit correspondant avec un jeu restreint d'opérateurs donnés.

exemple : prenons la fonction f à trois variables dont les valeurs sont données par la table suivante :

a	b	c	$f(a,b,c)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

On a obtenu $f(a,b,c) = (a+b+c)\cdot(a+\overline{b}+\overline{c})\cdot(\overline{a}+\overline{b}+\overline{c})$. On peut alors établir le circuit réalisant f en utilisant les portes ET, OU et NON (cf figure 2.4).

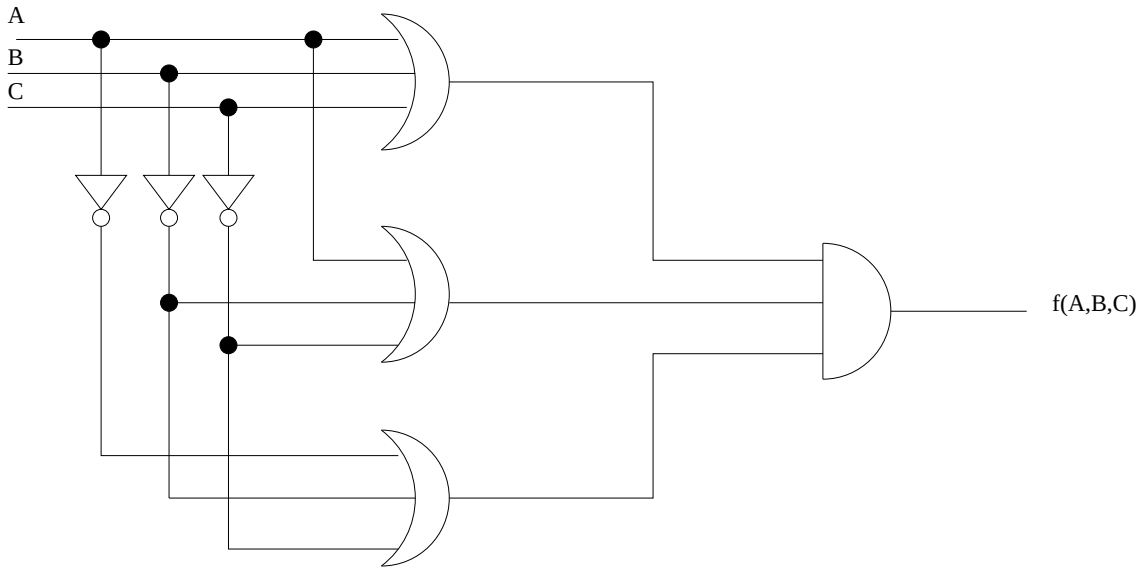


Fig. 2.4 – fonction f

(afin d'alléger le schéma on admet plus de deux entrées pour les portes ET et OU).

exemple : soit un circuit à 2 entrées qui doit donner la sortie 1 si les deux entrées ont la même valeur et 0 sinon. On remarque que $a = b$ équivaut à $\overline{a}\cdot\overline{b} + a\cdot b = 1$. Cela permet d'établir deux circuits selon les portes de base choisies (cf figure 2.5).

remarque : les entrées des circuits peuvent être des sorties d'autres circuits. Ainsi en branchant les sorties de 2 circuits comme entrées du circuit décrit ci-dessus, on peut tester l'équivalence des 2 circuits.

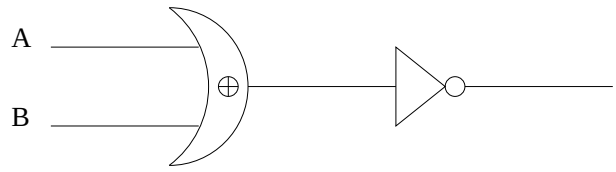


Fig. 2.5 – circuit d'équivalence

2.5 Logique propositionnelle

Le calcul propositionnel permet une modélisation du raisonnement mathématique simple : il traite des problèmes où les assertions ne peuvent prendre que deux valeurs possibles. Le calcul propositionnel a une propriété fondamentale : il est complet c'est-à-dire que tout ce qui est vrai est démontrable. Mais ce cadre ne suffit pas à décrire certaines situations mathématiques courantes comme l'existence d'un objet satisfaisant à une propriété donnée.

De façon générale, la logique fait apparaître la différence entre la *syntaxe* – les règles formelles de manipulation des symboles utilisés – et la *sémantique* – l'interprétation des formules.

2.5.1 Définition des formules propositionnelles

Soit $\mathcal{A}$ un ensemble de symboles constitué de la façon suivante :
- $\wedge, \vee, \longrightarrow, \longleftrightarrow, \neg, (,) \ \in \mathcal{A}$
- $T, F \in \mathcal{A}$
- $\mathcal{V} \subset \mathcal{A}$ où $\mathcal{V}$ est un ensemble dont les éléments sont appelés variables propositionnelles ou atomes

$\mathcal{A} = \{\wedge, \vee, \longrightarrow, \longleftrightarrow, \neg, (,), T, F\} \cup \mathcal{V}$

Définition 2.1 *L'ensemble $\mathcal{P}$ des formules propositionnelles construites sur $\mathcal{V}$ est défini inductivement par*
- *$T \in \mathcal{P}$, $F \in \mathcal{P}$, $\mathcal{V} \subset \mathcal{P}$ (les éléments de $\mathcal{V}$ sont alors appelés formules atomiques)*
- *si $\varphi \in \mathcal{P}$ alors $\neg\varphi \in \mathcal{P}$*
- *si $\varphi, \psi \in \mathcal{P}$ alors $(\varphi \wedge \psi) \in \mathcal{P}, (\varphi \vee \psi) \in \mathcal{P}, (\varphi \longrightarrow \psi) \in \mathcal{P}, (\varphi \longleftrightarrow \psi) \in \mathcal{P}$*

$\wedge, \vee, \longrightarrow, \longleftrightarrow, \neg$ sont des symboles logiques.

***exemple* 2.6** : Soit $\mathcal{V} = \{A, B, C\}$.

$\varphi = (A \longrightarrow (B \longrightarrow \neg C)) \in \mathcal{P}$ et $\psi = (\neg(A \longrightarrow C) \longrightarrow B) \in \mathcal{P}$.

□

2.5.2 Représentation arborescente

On peut considérer que la structure des éléments de $\mathcal{P}$ est arborescente : les feuilles sont les variables propositionnelles et les noeuds internes sont des symboles logiques.

***exemple* 2.7** : (suite de l'exemple 2.6)

□

On peut alors parler de la *hauteur* d'une formule propositionnelle comme étant la hauteur de l'arbre la représentant.

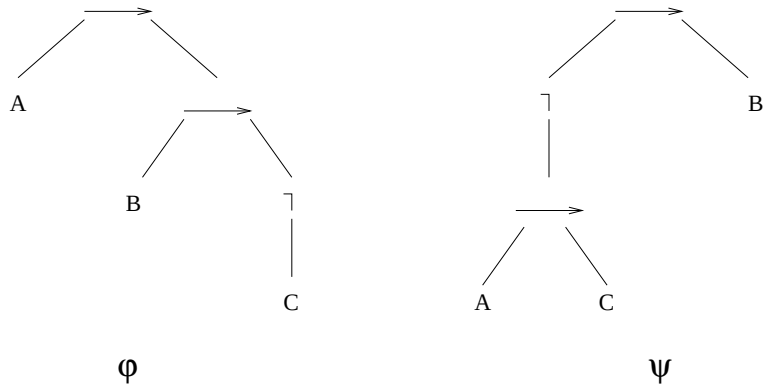


FIG. 2.6 – représentation arborescente de formules propositionnelles

***exemple* 2.8** : (suite de l'exemple 2.6) $h(\varphi) = h(\psi) = 3$

□

***exemple* 2.9** : $h((A \longrightarrow C)) = 1$ et $h(A) = 0$

□

On peut aussi définir de façon ascendante l'ensemble $\mathcal{P}$ en construisant les formules de "proche en proche" selon la définition inductive. On définit la suite d'ensembles de formules propositionnelles suivante :

- $\mathcal{P}_0 = \mathcal{V}$
- $\mathcal{P}_{n+1} = \mathcal{P}_n \cup \{\neg\varphi \,;\varphi \in \mathcal{P}_n\} \cup \{(\varphi \wedge \psi) \,;\varphi, \psi \in \mathcal{P}_n\} \cup \{(\varphi \vee \psi) \,;\varphi, \psi \in \mathcal{P}_n\}$
  $\cup\{(\varphi \longrightarrow \psi) \,;\varphi, \psi \in \mathcal{P}_n\} \cup \{(\varphi \longleftrightarrow \psi) \,;\varphi, \psi \in \mathcal{P}_n\}$

remarque : on a alors $\mathcal{P}_0 \subset \mathcal{P}_1 \subset \cdots \mathcal{P}_n \cdots$

Théorème 2.4 $\mathcal{P} = \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$

preuve $\mathcal{P}_0 = \mathcal{V} \subset \mathcal{P}$ par définition de $\mathcal{P}$. Supposons que pour un entier $n \geq 0$, on a $\mathcal{P}_n \subset \mathcal{P}$ (*). Soit $\phi \in \mathcal{P}_{n+1}$. On a les cas suivants

- $\phi \in \mathcal{P}_n$ et donc $\phi \in \mathcal{P}$ par hypothèse
- $\phi = \neg\varphi$ pour une formule $\varphi \in \mathcal{P}_n$; puisque $\varphi \in \mathcal{P}$ par hypothèse (*), on a, par définition de $\mathcal{P}$, $\varphi \in \mathcal{P}$
- $\phi = \varphi \wedge \psi$ pour deux formules $\varphi, \psi \in \mathcal{P}_n$; puisque $\varphi \in \mathcal{P}$ et $\psi \in \mathcal{P}$ par hypothèse (*), on a, par définition de $\mathcal{P}$, $\varphi \wedge \psi \in \mathcal{P}$
- $\phi = \varphi \vee \psi$ pour deux formules $\varphi, \psi \in \mathcal{P}_n$; puisque $\varphi \in \mathcal{P}$ et $\psi \in \mathcal{P}$ par hypothèse (*), on a, par définition de $\mathcal{P}$, $\varphi \vee \psi \in \mathcal{P}$
- $\phi = \varphi \longrightarrow \psi$ pour deux formules $\varphi, \psi \in \mathcal{P}_n$; puisque $\varphi \in \mathcal{P}$ et $\psi \in \mathcal{P}$ par hypothèse (*), on a, par définition de $\mathcal{P}$, $\varphi \longrightarrow \psi \in \mathcal{P}$
- $\phi = \varphi \longleftrightarrow \psi$ pour deux formules $\varphi, \psi \in \mathcal{P}_n$; puisque $\varphi \in \mathcal{P}$ et $\psi \in \mathcal{P}$ par hypothèse (*), on a, par définition de $\mathcal{P}$, $\varphi \longleftrightarrow \psi \in \mathcal{P}$

Donc $\mathcal{P}_{n+1} \subset \mathcal{P}$.

On a bien montré que pour tout entier n, $\mathcal{P}_n \subset \mathcal{P}$ et donc que $\bigcup_{n\in\mathbb{N}} \mathcal{P}_n \subset \mathcal{P}$.

Réciproquement, on va démontrer par induction sur $\mathcal{P}$ que $\mathcal{P} \subset \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$.

Par définition de $\mathcal{P}_0 = \mathcal{V}$, $\mathcal{V} \subset \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$.

Supposons que pour deux formules $\varphi, \varphi' \in \mathcal{P}$ on a $\varphi, \varphi' \in \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$; alors il existe k et k' tels que $\varphi \in \mathcal{P}_k$ et $\varphi' \in \mathcal{P}_{k'}$.

Si $K = sup\{k, k'\}$ on a donc $\varphi \in \mathcal{P}_K$ et $\varphi' \in \mathcal{P}_K$. On en déduit que $\neg\varphi \in \mathcal{P}_{K+1}$ et que $\varphi \diamond \varphi' \in \mathcal{P}_{K+1}$ pour $\diamond \in \{\wedge, \vee, \longrightarrow, \longleftrightarrow\}$. D'où $\neg\varphi \in \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$ et $\varphi \diamond \varphi' \in \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$ pour $\diamond \in \{\wedge, \vee, \longrightarrow, \longleftrightarrow\}$.□

2.5.3 Sous-formules

Définition 2.2 *l'ensemble des sous-formules d'une formule propositionnelle φ est l'ensemble des sous-arbres au sens large de φ.*

***exemple* 2.10** : (suite de l'exemple 2.6) $Ssf(\varphi) = \{\varphi; (B \longrightarrow \neg C); \neg C; A; B; C\}$

On peut aussi définir l'ensemble des sous-formules d'une formule propositionnelle φ inductivement par

- si $\varphi = A$ où $A \in \mathcal{V}$ alors $Ssf(\varphi) = \{A\}$
- si $\varphi = \neg\psi$ alors $Ssf(\varphi) = \{\neg\psi\} \cup Ssf(\psi)$
- si $\varphi = (\varphi_1 \diamond \varphi_2)$ alors $Ssf(\varphi) = \{(\varphi_1 \diamond \varphi_2)\} \cup Ssf(\varphi_1) \cup Ssf(\varphi_2)$ pour $\diamond \in \{\wedge, \vee, \longrightarrow, \longleftrightarrow\}$.

2.5.4 Substitution

***exemple* 2.11** : (suite de l'exemple 2.6) en remplaçant les feuilles de φ par des formules propositionnelles – A par $(B \longrightarrow C)$, B par $\neg C$ et C par $\neg(A \longrightarrow C)$ – on obtient une nouvelle formule propositionnelle.

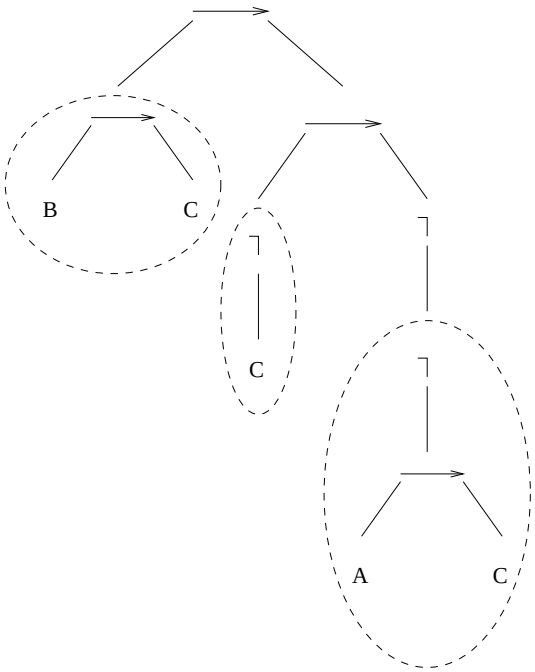


FIG. 2.7 – substitution dans φ

□

Définition 2.3 *Soient $A_1, \cdots A_n$ n variables propositionnelles distinctes deux à deux. Soient $\varphi_1, \cdots \varphi_n$ n formules propositionnelles et ψ une formule propositionnelle. On définit $\psi' = \psi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ par*
- *si $\psi \in \mathcal{V}$ alors $\psi' = \psi$ si $\psi \notin \{A_1, \cdots A_n\}$ et $\psi' = \varphi_k$ si $\psi = A_k$*
- *si $\psi = \neg\phi$ alors $\psi' = \neg\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$*
- *si $\psi = (\alpha \diamond \beta)$ alors $\psi' = \alpha_{\varphi_1/A_1,\cdots\varphi_n/A_n} \diamond \beta_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ pour $\diamond \in \{\wedge, \vee, \longrightarrow, \longleftrightarrow\}$.*

Théorème 2.5 *$\psi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ est une formule propositionnelle.*

On verra, qu'une fois interprétées, ψ et $\psi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ ont des liens.

2.5.5 Sémantique : Interprétation – valeurs de vérité

Définition 2.4 *une interprétation est une application $v_{\mathcal{V}}$ de $\mathcal{V}$ dans $\mathbb{B}$(à chaque variable propositionnelle est associé un élément de $\mathbb{B}$).*

Le résultat suivant permet d'étendre la définition de $v_{\mathcal{V}}$ à $\mathcal{P}$.

Théorème 2.6 *il existe une unique application v de $\mathcal{P}$ dans $\mathbb{B}$prolongeant $v_{\mathcal{V}}$ telle que*
- $v(A) = v_{\mathcal{V}}(A)$ *pour tout* $A \in \mathcal{V}$,
- $v(T) = 1$ *et* $v(F) = 0$,
- $v(\neg\varphi) = \overline{v(\varphi)}$,
- $v(\varphi \wedge \psi) = v(\varphi) \cdot v(\psi)$
- $v(\varphi \vee \psi) = v(\varphi) + v(\psi)$,
- $v(\varphi \longrightarrow \psi) = \overline{v(\varphi)} + v(\psi)$
- $v(\varphi \longleftrightarrow \psi) = \overline{v(\varphi)} \cdot \overline{v(\psi)} + v(\varphi) \cdot v(\psi)$

***exemple* 2.12** : (suite de l'exemple 2.6)

$v(\varphi) = \overline{v(A)} + v((B \longrightarrow \neg C) = \overline{v(A)} + \overline{v(B)} + v(\neg C) = \overline{v(A)} + \overline{v(B)} + \overline{v(C)}$

Si $v(A) = 0, v(B) = 1$ et $v(C) = 0$ alors $v(\varphi) = \overline{0} + \overline{1} + \overline{0} = 1 + 0 + 1 = 1$.

□

Définition 2.5 *une formule φ est dite* satisfiable *s'il existe une interprétation v telle que $v(\varphi) = 1$. On dit alors que v est un modèle de φ.*

***exemple* 2.13** : soit $\alpha = ((A \longrightarrow B) \longrightarrow (A \longrightarrow C))$

$v(\alpha) = \overline{v(A \longrightarrow B)} + v(A \longrightarrow C) = \overline{\overline{v(A)} + v(B)} + \overline{v(A)} + v(C) = v(A) \cdot \overline{v(B)} + \overline{v(A)} + v(C)$.

Si $v(A) = 0, v(B) = 1$ et $v(C) = 1$ alors $v(\alpha) = 1$ donc α est satisfiable.

□

On dira qu'une formule φ est
- valide si toute interprétation est un modéle de φ (on notera alors $\models \varphi$)
- insatisfiable si aucune interprétation n'est un modéle de φ (on dit aussi que φ est contradictoire)

***exemple* 2.14** : (suite de l'exemple 2.13) α n'est pas valide puisque, si $v(A) = 1, v(B) = 1$ et $v(C) = 0$ on a $v(\alpha) = 0$.

□

remarque : dans $\mathbb{B}$, 0 et 1 représentent aussi les valeurs de vérité $Vrai$ et $Faux$ et les opérations $+$, $\cdot$,- représentent respectivement les connecteurs logiques $\wedge, \vee, \neg$ *i.e.* le *ou* logique, le *et* logique et la *négation*.

Pour éviter trop de parenthèses on admet les priorités suivantes (de la plus élevée à la plus faible) : $\neg \ > \ \wedge, \vee \ > \longrightarrow, \longleftrightarrow$, mais on peut toujours représenter les formules propositionnelles sous forme arborescente.

Il est important de noter pour les connecteurs $\vee, \wedge, \longrightarrow$ que leurs tables de vérité comportent toujours un cas particulier qui correspond à $Faux$ ou $Vrai$ selon le connecteur :

Propriété 2.7 *Soient φ et ψ deux formules et v une interprétation quelconque.*

- $v(\varphi \longrightarrow \psi) = 0$ *si et seulement si* $v(\varphi) = 1$ *et* $v(\psi) = 0$
- $v(\varphi \vee \psi) = 0$ *si et seulement si* $v(\varphi) = 0$ *et* $v(\psi) = 0$
- $v(\varphi \wedge \psi) = 1$ *si et seulement si* $v(\varphi) = 1$ *et* $v(\psi) = 1$
- $v(\varphi \longleftrightarrow \psi) = 1$ *si et seulement si* $v(\varphi) = v(\psi)$

remarque : en particulier, si $v(\varphi) = 0$ alors $v(\varphi \longrightarrow \psi) = 1$ quelque soit la valeur de $v(\psi)$.

2.5.6 Equivalence sémantique

Définition 2.6 *On dira que φ et ψ sont logiquement équivalentes, noté $\varphi \equiv \psi$, si pour toute interprétation v, $v(\varphi) = v(\psi)$.*

la relation $\equiv$ a la propriété suivante

Propriété 2.8 *$\equiv$ est une relation d'équivalence sur $\mathcal{P}$*

Les trois théorèmes suivants font le lien entre formules et sous-formules et la notion de substitution.

Théorème 2.9 *Soit φ une formule propositionnelle, $\varphi_1, \cdots, \varphi_n$ n formules propositionnelles et $A_1, \cdots, A_n$ n variables propositionnelles distinctes 2 à 2. Si φ est une tautologie alors $\varphi_{\varphi_1/A_1, \cdots \varphi_n/A_n}$ est aussi une tautologie.*

***exemple* 2.15** : Soit $\varphi = A \vee \neg A$. φ est clairement une tautologie donc $A \wedge (B \longrightarrow C) \vee \neg(A \wedge (B \longrightarrow C))$ est aussi une tautologie.

□

Théorème 2.10 *Soit ψ une sous-formule de φ et soit ψ' une formule équivalente à ψ. Alors la formule obtenue en substituant ψ' à ψ dans φ est logiquement équivalente à φ.*

***exemple* 2.16** : Soit $\varphi = (A \wedge (B \longrightarrow (C \wedge \neg A))) \vee \neg(A \wedge (B \longrightarrow C))$.

La formule $A \ \wedge \ (B \ \longrightarrow \ (C \ \wedge \ \neg \ A))$ est une sous-formule de φ et est logiquement équivalente à $A \wedge \neg B$ donc $\varphi \equiv (A \wedge \neg B) \vee \neg(A \wedge (B \longrightarrow C))$.

□

Théorème 2.11 *Soient $\varphi_1, \cdots, \varphi_n$ n formules propositionnelles et $A_1, \cdots, A_n$ n variables propositionnelles distinctes 2 à 2. Soit v une interprétation, on définit l'interprétation v' de la façon suivante*

$$\begin{aligned} v'(A) &= v(A) \text{ si } A \in \mathcal{V} \text{ et } A \notin \{A_1, \cdots, A_n\} \\ &= v(\varphi_i) \text{ si } A = A_i \end{aligned}$$

On a alors $v(\varphi_{\varphi_1/A_1,\cdots\varphi_n/A_n}) = v'(\varphi)$

preuve : on démontrera ce résultat par induction sur la construction des formules propositionnelles. Soit $\psi = \varphi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$

- si $\varphi = T$ alors $\psi = T$ et $v(\psi) = v'(\varphi) = 1$.
- si $\varphi = F$ alors $\psi = F$ et $v(\psi) = v'(\varphi) = 0$.
- si $\varphi = A_i$ alors $\psi = \varphi_i$ et $v(\psi) = v(\varphi_i) = v'(A_i) = v'(\varphi)$.
- si $\varphi = A$ et $A \notin \{A_1, \cdots, A_n\}$ alors $\psi = \varphi$ et $v(\psi) = v(A) = v'(A) = v'(\varphi)$.
- supposons que la propriété soit vraie pour deux formules quelconques ϕ et ϕ'.
  • si $\varphi = \neg\phi$ alors $\psi = \neg\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n}$. Donc $v(\psi) = \overline{v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n})} = \overline{v'(\phi)}$ par hypothèse sur ϕ. Puisque $v'(\varphi) = \overline{v'(\phi)}$ on a $v(\psi) = v'(\varphi)$.
  • si $\varphi = \phi \wedge \phi'$ alors $\psi = \phi_{\varphi_1/A_1,\cdots\varphi_n/A_n} \wedge \phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ et on a $v(\psi) = v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n}) \cdot v(\phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}) = v'(\phi) \cdot v'(\phi')$ par hypothèse sur ϕ et ϕ'. On en déduit que $v(\psi) = v'(\phi \wedge \phi') = v'(\varphi)$.
  • si $\varphi = \phi \vee \phi'$ alors $\psi = \phi_{\varphi_1/A_1,\cdots\varphi_n/A_n} \vee \phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ et on a $v(\psi) = v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n}) + v(\phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}) = v'(\phi) + v'(\phi')$ par hypothèse sur ϕ et ϕ'. On en déduit que $v(\psi) = v'(\phi \vee \phi') = v'(\varphi)$
  • si $\varphi = \phi \longrightarrow \phi'$ alors $\psi = \phi_{\varphi_1/A_1,\cdots\varphi_n/A_n} \longrightarrow \phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ et on a $v(\psi) = \overline{v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n})} + v(\phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}) = \overline{v'(\phi)} + v'(\phi')$ par hypothèse sur ϕ et ϕ'. On en déduit que $v(\psi) = v'(\phi \longrightarrow \phi') = v'(\varphi)$
  • si $\varphi = \phi \longleftrightarrow \phi'$ alors $\psi = \phi_{\varphi_1/A_1,\cdots\varphi_n/A_n} \longleftrightarrow \phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}$ et on a $v(\psi) = \overline{v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n})} \cdot \overline{v(\phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n})} + v(\phi_{\varphi_1/A_1,\cdots\varphi_n/A_n}) \cdot v(\phi'_{\varphi_1/A_1,\cdots\varphi_n/A_n}) = \overline{v'(\phi)} \cdot \overline{v'(\phi')} + v'(\phi) \cdot v'(\phi')$ par hypothèse sur ϕ et ϕ'. On en déduit que $v(\psi) = v'(\phi \longleftrightarrow \phi') = v'(\varphi)$.

***exemple* 2.17** : on veut calculer la table de vérité de la formule $\varphi = ((A \longrightarrow B) \longrightarrow C) \longrightarrow (A \longrightarrow C)$. Pour toute interprétation v, $v(\varphi) = v'(K \longrightarrow L)$ où $v'(K) = v((A \longrightarrow B) \longrightarrow C)$ et $v'(L) = v(A \longrightarrow C)$. Cela permet d'établir le type de table de vérité suivant :

A	B	C	$(A \longrightarrow B) \longrightarrow C$	$A \longrightarrow C$	φ
0	0	0	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	1	1	1

□

2.5.7 Des tables de vérité aux formules

En établissant une table de vérité on décrit en fait le graphe d'une fonction booléenne c'est-à-dire une fonction de $\mathbb{B}^n$ dans $\mathbb{B}$.

Donc à chaque formule φ correspond une fonction $f : \mathbb{B}^n \longrightarrow \mathbb{B}$.

On peut alors se demander si la correspondance inverse est vraie et combien y-a-t-il de telles fonctions pour une valeur de n fixée.

On sait que $|\mathbb{B}^n| = 2^n$ donc il y a 2^{2^n} applications de $\mathbb{B}^n$ dans $\mathbb{B}$.

On remarque ensuite que, si $\varphi \equiv \psi$, alors φ et ψ ont la même table de vérité et donc correspondent à la même fonction. On peut donc dire qu'une classe d'équivalence de formules détermine une fonction.

Mais si on choisit une fonction au hasard peut-on trouver une telle classe ou du moins un de ses représentants ?

Théorème 2.12 *Pour toute application $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ il existe au moins une formule dont la table de vérité est le graphe de l'application f.*

preuve : on donnera un exemple ; si l'application f vaut 1 pour les n-uplets suivants et seulement pour ceux là :

$$(0, \cdots, 0, 1), (1, 0, \cdots, 0), (1, 1, 0, \cdots, 0), (0, \cdots, 0, 1, 1)$$

alors une formule φ dont la table de vérité est le graphe de l'application f est $\varphi =$

$(\neg A_1 \wedge \neg A_2 \wedge \cdots \wedge \neg A_{n-1} \wedge A_n) \vee (A_1 \wedge \neg A_2 \wedge \cdots \wedge \neg A_n) \vee (A_1 \wedge A_2 \wedge \neg A_3 \wedge \cdots \wedge \neg A_n) \vee (\neg A_1 \wedge \neg A_2 \wedge \cdots \wedge \neg A_{n-2} \wedge A_{n-1} \wedge A_n)\square$

Parce que cette preuve est généralisable, cela signifie que toute formule est équivalente à une formule de la même forme que φ. C'est ce qui nous amène à la notion de forme normale.

2.5.8 Formes normales

D'après l'exemple précédent, les formules s'expriment en fonction des variables atomiques ou de leur négation et des connecteurs $\wedge$ et $\vee$.

Définition 2.7 *un* littéral *est une variable atomique ou la négation d'une variable atomique*

Soit n le nombre total de variables atomiques. Soit $\mathcal{V} = \{A_1 \cdots A_n\}$ l'ensemble des variables atomiques. On notera $l_i = A_i$ ou $l_i = \neg A_i$ et $\widetilde{\mathcal{V}}$ l'ensemble des n-uplets $(l_1, \cdots, l_n)$.

Définition 2.8 *une formule φ est sous* forme normale disjonctive canonique *(FNDC) si et seulement si* $\varphi = \bigvee_{(l_1,\cdots,l_n)\in\widetilde{\mathcal{V}}} (l_1 \wedge \cdots \wedge l_n)$

Définition 2.9 *une formule φ est sous* forme normale conjonctive canonique *(FNCC) si et seulement si* $\varphi = \bigwedge_{(l_1,\cdots l_n)\in\widetilde{\mathcal{V}}} (l_1 \vee \cdots \vee l_n)$

remarque : on parle de FND ou de FNC lorsque toutes les variables atomiques ne sont pas nécessairement présentes.

***exemple* 2.18** : Soit $\mathcal{V} = \{A, B, C\}$. On définit φ et φ' :

$\varphi = (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge \neg C)$ est en FNDC.

$\varphi' = (\neg A \wedge \neg B) \vee (A \wedge B \wedge \neg C)$ est en FND.

On a $\varphi \equiv \varphi'$

□

Toute formule est équivalente à une formule en FNDC ou en FNCC comme on l'a vu dans l'introduction de ce paragraphe. Mais comment déterminer une FNDC ou une FNCC équivalente à une formule donnée ?

***exemple* 2.19** : Soit $\varphi = (A \longrightarrow (((B \wedge \neg A) \vee (\neg C \wedge A)) \longleftrightarrow (A \vee (A \longrightarrow B))))$. On peut établir la table de vérité de φ :

A	B	C	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

φ est satisfait par les interprétations suivantes : $\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{matrix}$ et $\neg\varphi$ par : $\begin{matrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$

□

On peut alors écrire

$\varphi \equiv (\neg A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge \neg C)$,

et $\neg\varphi \equiv (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$.

On en déduit par les lois de De Morgan que $\varphi \equiv (\neg A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg C)$.

On a ainsi les formes FNDC et FNCC de φ.

□

La méthode employée dans cet exemple est basée sur les remarques suivantes

- si $L = (l_1, \cdots l_n) \in \widetilde{\mathcal{V}}$ alors la formule $l_1 \wedge \cdots \wedge l_n$ est satisfaite par l'interprétation v_L telle que $v_L(A_i) = 1$ si $l_i = A_i$ et $v_L(A_i) = 0$ si $l_i = \neg A_i$, et seulement par cette interprétation.
- si $L_k = (l_{k_1}, \cdots l_{k_n}) \in \widetilde{\mathcal{V}}$ pour $k = 1, \cdots, p$ alors la formule $\bigvee_{1 \leq k \leq p} (l_{k_1} \wedge \cdots \wedge l_{k_n})$ est satisfaite par les interprétations v_{L_k}, $k = 1, \cdots, p$ et seulement par celles-ci.

2.5.9 Système complet de connecteurs

C'est un ensemble de connecteurs logiques qui permet d'engendrer tous les connecteurs de la logique propositionnelle.

***exemple* 2.20** : $\{\neg, \longrightarrow\}$ est complet et minimal, ainsi que $\{\neg, \vee\}$ et $\{\neg, \wedge\}$.
$\{\neg, \wedge, \vee\}$ est complet mais pas minimal et $\{\wedge, \vee\}$ n'est pas complet.

□

Pour les preuves inductives, on peut se contenter de deux étapes en choisissant bien les connecteurs.

2.5.10 Clauses de Horn

Définition 2.10 *une* clause *est une disjonction de littéraux*

Définition 2.11 *une* clause de Horn *est une clause dont au plus un littéral est "positif" c'est-à-dire est une variable atomique.*

***exemple* 2.21** : si $\mathcal{V} = \{A, B, C\}$ alors $A \vee \neg B \vee \neg C$, $\neg A \vee B$ sont des clauses de Horn.

□

remarque : une clause de Horn (avec un littéral positif) est en fait équivalente à une formule du type $(A_1 \wedge \cdots \wedge A_n) \longrightarrow B$ où $A_1, \cdots, A_n, B$ sont des variables atomiques.

S'il n'y a pas de littéral positif alors la clause de Horn est en fait équivalente à une formule du type $(A_1 \wedge \cdots \wedge A_n) \longrightarrow F$ et s'il n'y a qu'un littéral positif et pas de littéraux négatifs alors la clause de Horn est en fait équivalente à une formule du type $T \longrightarrow B$.

remarque : une clause de Horn peut être vide, elle est alors équivalente à $True$.

On dira qu'une formule est une *formule de Horn* si elle est en FNC et si chaque disjonction contient au plus une variable atomique *i.e.* une formule de Horn est une conjonction de clauses de Horn.

Il existe un algorithme pour tester la satisfiabilité d'une formule de Horn φ.

- s'il existe une sous-formule $T \longrightarrow A$ alors marquer chaque occurrence de A dans φ
- appliquer les règles suivantes jusqu'à ce que l'on ne puisse plus
  - si $(A_1 \wedge \cdots \wedge A_n) \longrightarrow B$ et si $A_1, \cdots, A_n$ sont marqués et B n'est pas marqué alors marquer chaque occurrence de B dans φ
  - si $(A_1 \wedge \cdots \wedge A_n) \longrightarrow F$ et si $A_1, \cdots, A_n$ sont marqués alors **stop** : φ est insatisfiable
- **stop** : φ est satisfiable par l'interprétation v définie par $v(A) = 1$ *si et seulement si* A est marqué dans φ.

Cet algorithme termine en un temps au plus n, où n est le nombre de variables propositionnelles.

***exemple* 2.22** : soit $\varphi = (A \vee \neg B) \wedge (\neg A \vee \neg B \vee C) \wedge C$. Alors $\varphi \equiv (B \longrightarrow A) \wedge (A \wedge B \longrightarrow C) \wedge (T \longrightarrow C)$ et on marque chaque occurrence de C dans φ

$\varphi \equiv (B \longrightarrow A) \wedge (A \wedge B \longrightarrow \mathbf{C}) \wedge (T \longrightarrow \mathbf{C})$

L'algorithme s'arrête et φ est satisfiable par l'interprétation v avec $v(A) = v(B) = 0, v(C) = 1$

□ *remarque* : toute formule n'est pas équivalente à une clause de Horn. Par exemple $A \vee B$.

Chapitre 3

Concepts d'algorithmes

Avant d'aborder la notion d'algorithme dont un des buts est de pouvoir être programmé et exécuté par un ordinateur on doit d'abord comprendre que les objets que l'on manipulera seront codés dans l'ordinateur.

3.1 Information digitale

Toute information traitée dans un ordinateur est codée et ce codage consiste théoriquement en une suite de 0 et de 1. En pratique, le dispositif physique permettant de garder une information binaire sera appelée *bit* pour **b***inary dig***it**. Ces bits sont regroupés par N où $N \in \mathbb{N}$ et une telle suite de N bits sera appelée *mot*; ces mots seront considérés comme une unité d'information et traités par la machine comme un seul bloc aussi bien au niveau de l'accès mémoire et des transports qu'au niveau du traitement logique ou arithmétique. La plupart du temps N est égal à un multiple de 8 et un groupement de huit bits se nomme *octet* (*byte* en anglais).

Un mot de N bits peut avoir 2^N états possibles, on peut donc représenter 2^N informations différentes.

***exemple* 3.1** : avec $N = 3$, 000, 001, 010, 011, 100, 101, 110, 111 sont les 8 états possibles pour un mot de 3 bits.

□

Dans cette section on s'intéressera à la façon de coder les nombres. Le codage consiste à établir une loi de correspondance (code) entre les informations à représenter et les configurations binaires de telle sorte qu'à chaque information corresponde une et une seule information binaire. Auparavent, on introduira les notions de base sur l'écriture des nombres en base 2.

3.1.1 Arithmétique binaire

Ecriture des entiers en base 2

Tout nombre entier naturel peut s'écrire en *base* 2 c'est-à-dire comme une suite finie de 0 et de 1 représentant des coefficients de puissances de 2 selon le résultat suivant :

théorème : pour tout $n \in \mathbb{N}$, il existe $k \in \mathbb{N}$ et une suite $a_0, \cdots, a_k$ avec $a_i \in \{0, 1\}$ pour $i \in \{0, \cdots, k\}$ tels que $n = \sum_{i=0}^{i=k} a_i 2^i = a_0 + a_1 2 + a_2 2^2 + \cdots + a_k 2^k$.

définition : n s'écrit en *base* 2 de la façon suivante : $n = a_k a_{k-1} \cdots a_1 a_{0_{deux}}$.

exemple : $7 = 111_{deux}$ car $7 = 2^2 + 2^1 + 2^0$ $\quad$ $25 = 11001_{deux}$ car $25 = 2^4 + 2^3 + 2^0$.

Pour trouver la décomposition d'un entier n en base 2, on peut appliquer l'algorithme suivant :

répéter

diviser n par 2 ; écrire le reste de la division à gauche des écritures précédentes

remplacer n par le quotient de la division

jusqu'à ce que $n = 0$

exemple : $n = 175$

$n : 2 = 87$ - reste 1 - écrire **1**

$87 : 2 = 43$ - reste 1 - écrire **1**1

$43 : 2 = 21$ - reste 1 - écrire **1**11

$21 : 2 = 10$ - reste 1 - écrire **1**111

$10 : 2 = 5$ - reste 0 - écrire **0**1111

$5 : 2 = 2$ - reste 1 - écrire **1**01111

$2 : 2 = 1$ - reste 0 - écrire **0**101111

$1 : 2 = 0$ - reste 1 - écrire **1**0101111

arrêt et $n = 10101111_{deux}$.

Réciproquement il est facile de passer de la base 2 à l'écriture décimale en commençant la lecture du nombre écrit en base deux de la droite vers la gauche.

exemple : $110010010_{deux} = 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 + 1 \times 2^8 = 2 + 16 + 128 + 512 = 658$

additionner en base 2

La règle d'addition en base 2 est basée sur la remarque suivante : $2^n + 2^n = 2^{n+1}$. On posera donc une addition en base 2 comme une addition d'entiers habituelle avec les règles de retenue : $0 + 0 = 0$ - retenue 0 — $0 + 1 = 1 + 0 = 1$ - retenue 0 — $1 + 1 = 0$ - retenue 1.

exemples :

$$\begin{array}{cccccccc} & & & 1^1 & 1^1 & 0^1 & 0^1 & 1 \\ & + & & & 1 & 1 & 1 & 1 \\ \hline = & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

$(25 + 15 = 40)$

$$\begin{array}{cccccc} & & & & 1 & 0 & 1 \\ + & 1 & 0^1 & 1 & 0 & 0 \\ \hline = & 1 & 1 & 0 & 0 & 1 \end{array}$$

$(5 + 20 = 25)$

multiplier en base 2

On posera une multiplication en base 2 de la même façon qu'avec des entiers écrits en base 10.

exemple :

$$\begin{array}{cccccc} & & & 1 & 1 & 0 & 0 \\ & & \times & & 1 & 0 & 1 \\ \hline & & & 1 & 1 & 0 & 0 \\ & & 0 & 0 & 0 & 0 & \cdot \\ & 1 & 1 & 0 & 0 & \cdot & \\ \hline = & 1 & 1 & 1 & 1 & 0 & 0 \end{array}$$

$(12 \times 5 = 60)$

$$\begin{array}{cccccccc} & & & & 1 & 0 & 1 & 1 \\ & & & \times & 1 & 0 & 1 & 1 \\ \hline & & & & 1^1 & 0^1 & 1 & 1 \\ & & & 1^1 & 0 & 1 & 1 & \cdot \\ & & 0^1 & 0 & 0 & 0 & \cdot & \cdot \\ & 1 & 0 & 1 & 1 & \cdot & \cdot & \cdot \\ \hline = & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}$$

$(11^2 = 121)$

remarque : la multiplication par 2 revient à *décaler* les bits d'un rang vers la gauche et à ajouter un 0 à droite des autres bits.

soustraire en base 2

On posera une soustraction $a - b$ avec $a \geq b$ comme pour des entiers.

***exemple* 3.2** :

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & \\ - & & 1 & 1 & 1 & \\ \hline = & & 1 & 0 & 0 & \end{array}$$

(cette soustraction représente $11 - 7 = 4$)

□

Le cas où $a < b$ sera envisagé lorsque l'on présentera la représentation des entiers relatifs.

diviser en base 2

On effectue la division entière au moyen de soustractions répétées avec décalage à droite.

***exemple* 3.3** :

$$\begin{array}{cccc|ccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & & 1 & 0 & \\ \cline{1-3} & & 1 & 0 & & & \\ & & 0 & 0 & & & \\ \cline{3-4} & & 1 & 0 & & & \end{array}$$

(cette division représente $10 : 4 = 2$ reste 2)

□

Ecriture des rationnels en base 2

Parmi les nombres rationnels, on distingue les décimaux et les autres qui n'ont pas une écriture décimale finie. Par exemple, $\frac{2}{3} = 0,6666\cdots$ mais $\frac{5}{4} = 1,25$; cette écriture signifie en fait $\frac{2}{3} = \sum_{i\geq 1} 6.10^{-i} = 0,6 + 0,06 + 0,006 + \cdots$ et $\frac{5}{4} = 10^0 + 2.10^{-1} + 5.10^{-2} = 1 + 0,2 + 0,05$.

En base 2, la *décomposition* se fera selon les puissances positives et négatives de 2. Comme dans le cas de la base décimale certains nombres auront une écriture finie, d'autres non.

exemples : $\frac{2}{3} = \sum_{i\geq 0} 2^{-2i-1}$ donc $\frac{2}{3} = 0,010101010\cdots_{deux}$ $\quad \frac{5}{4} = 1 + 2^{-2}$ donc $\frac{5}{4} = 1,01_{deux}$.

remarque : Attention, contrairement à ce que l'exemple pourrait suggérer, les rationnels qui ont une écriture décimale finie n'ont pas nécessairement une écriture en base 2 finie. Par exemple, $\frac{1}{5} = \frac{1}{15} + \frac{2}{15} = \sum_{i\geq 0} 2^{-4-4i} + 2^{-3-4i}$ donc $\frac{1}{5} = 0,001100110011\cdots_{deux}$, or $\frac{1}{5} = 0,2$.

Pour trouver la décomposition d'un nombre décimal x en base 2, on peut appliquer l'algorithme suivant :

séparer la partie entière n de la partie décimale y ($y < 1$)

répéter

diviser n par 2 ; écrire le reste de la division à gauche des écritures précédentes

remplacer n par le quotient de la division

jusqu'à ce que $n = 0$

placer la virgule derrière n

répéter

multiplier y par 2 ; écrire le chiffre des unités à droite des écritures précédentes

remplacer y par la partie décimale de la multiplication

jusqu'à ce que $y = 0$ (il se peut que cela ne se produise jamais)

***exemple* 3.4** : $x = 125,625$

$n : 2 = 62$ - reste 1 - écrire **1**

$62 : 2 = 31$ - reste 0 - écrire **0**1

$31 : 2 = 15$ - reste 1 - écrire **1**01

$15 : 2 = 7$ - reste 1 - écrire **1**101

$7 : 2 = 3$ - reste 1 - écrire **1**1101

$3 : 2 = 1$ - reste 1 - écrire **1**11101

$1 : 2 = 0$ - reste 1 - écrire **1**111101

arrêt et écrire 1111101,

$0,625 \times 2 = 1,25$ - écrire 1111101,**1**

$0,25 \times 2 = 0,5$ - écrire 1111101,1**0**

$0,5 \times 2 = 1,0$ - écrire 1111101,10**1**

arrêt et $\overline{x}^2 = 1111101,101_{deux}$

□

***exemple* 3.5** : $x = 1,2$

$1 : 2 = 0$ - reste 1 - écrire **1**

arrêt et écrire 1,

$0,2 \times 2 = 0,4$ - écrire 1,**0**

$0,4 \times 2 = 0,8$ - écrire 1,0**0**

$0,8 \times 2 = 1,6$ - écrire 1,00**1**

$0,6 \times 2 = 1,2$ - écrire 1,001**1**

$0,2 \times 2 = 0,4$ - écrire 1,0011**0**

$\cdots$ (l'itération est infinie)

$\overline{x}^2 = 1,0011001100110011\cdots_{deux}$

□

Réciproquement, pour passer de l'écriture en base 2 à l'écriture en base 10, il suffit de suivre l'exemple ci-dessous :

***exemple* 3.6** : $1101,11_{deux} = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2} = 8+4+0+1+0,5+0,25 = 13,75$.

□

3.1.2 Représentation des entiers naturels

On s'intéresse maintenant à la représentation en machine des entiers naturels. On utilisera donc des *mots* de N bits pour coder ces entiers. Il est facile de calculer que l'on peut alors coder 2^N informations différentes sur N bits donc pour des entiers naturels, la plage représentée va de 0 à $2^N - 1$ (en effet $\underbrace{11\cdots1_{deux}}_{N\text{bits}} = \sum_{i=0}^{i=N-1} 2^i = 2^N - 1$).

***exemple* 3.7** : $N = 6$, 3 est représenté par 000011 et 011011 représente 51.

$N = 8$, 11111111 représente 255 et 179 est représenté par 10110011.

□

3.1.3 Représentation des entiers relatifs

différents codages

Pour représenter les entiers relatifs (positifs et négatifs) (ou entiers signés), la première idée est de réserver un bit pour coder le signe.

***exemple* 3.8** : $N = 6$, 011011 représente 27
111011 représente -27

□

Un inconvénient est qu'alors l'addition bit à bit de ces deux nombres ne donne pas 0. Il faut donc traiter différemment les additions de nombres positifs et les additions d'un positif et d'un négatif.

Une deuxième idée est d'alors de représenter un négatif à partir de la représentation de sa valeur absolue en changeant tous les bits. On dit que l'on prend son complément restreint.

***exemple* 3.9** : $N = 4$, 0011 représente 3
1100 représente -3

□

Mais alors 0 est codé de deux façons 0000 et 1111.

Une troisième méthode pour coder les relatifs permet d'éviter ces types de problèmes : ***le complément à deux***.

L'idée est toujours de réserver un bit pour représenter le signe − mais de façon à ce que la somme de deux entiers opposés donnent toujours le mot $000\ldots00$ sur les N bits.

Pour ce faire on peut noter que si un bit est pris pour le signe alors il reste $N-1$ bits pour représenter la valeur et donc le plus grand entier positif sera $2^{N-1} - 1$.

Dans ce cas, on peut avoir tous les entiers négatifs de -2^{N-1} à -1 en ajoutant -2^{N-1} respectivement à 0,... , $2^{N-1} - 1$.

Le principe est alors d'interpréter le bit le plus à gauche (appelé aussi *bit de plus fort poids*) comme le coefficient de -2^{N-1} alors que les autres bits ont leur signification habituelle.

***exemple* 3.10** : $N = 8$, 00101111 représente $1 + 2 + 4 + 8 + 32 = 47$
10101111 représente $1 + 2 + 4 + 8 + 32 - 2^7 = 47 - 128 = -81$

□

Le complément à deux a plusieurs avantages :
- 0 a un unique codage
- tous les nombres négatifs ont un 1 comme bit de plus fort poids et les positifs ont tous 0 comme bit de plus fort poids. Cela permet de déterminer facilement le signe d'un entier codé en complément à deux.
- l'addition d'un entier positif et d'un entier négatif peut se faire bit à bit.

***exemple* 3.11** : $N = 8$,

$$\begin{array}{llllll} 01000101 & \text{représente} & 1+4+64 & = & 69 & \\ 10100100 & \text{représente} & 4+32-2^7 & = & 36-128 & = -92 \end{array}$$

$$\begin{array}{rcccccccc} & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ + & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline = & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}$$

et 11101001 représente $1 + 8 + 32 + 64 - 128 = -23$.

□

Le tableau suivant montre les différents codages possibles avec $N = 4$.

entier	valeur absolue	complément restreint	complément à deux
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
+0	0000	0000	0000
−0	1000	1111	
−1	1001	1110	1111
−2	1010	1101	1110
···	···	···	···
−6	1110	1001	1010
−7	1111	1000	1001
−8	—	—	1000

***exemple* 3.12** : avec $N = 32$, quelle est la valeur représentée en complément à deux par

1111 1111 1111 1111 1111 1111 1111 1100 ?

Le bit de plus fort poids est le coefficient de -2^{31} alors que les autres sont les coefficients des puissances 2^{30} à 2^0 en lisant de gauche à droite. Donc le nombre représenté est

$$\begin{aligned} 1 \times -2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \cdots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 &= -2^{31} + 2^{30} + 2^{29} + \cdots + 2^2 \\ &= -2^{31} + \frac{2^{31} - 2^2}{2 - 1} \\ &= -4 \end{aligned}$$

□

coder en complément à deux

Pour trouver la représentation d'un entier négatif, on peut appliquer deux méthodes :
- prendre le complément restreint du codage de la valeur absolue puis lui ajouter 1.
- coder la valeur absolue puis lire de la droite vers la gauche : après le premier bit valant 1 changer tous les bits (toujours de la droite vers la gauche)

***exemple* 3.13** : avec $N = 8$, coder -97
- $1^{ère}$ méthode
  - 97 est codé par 0110 0001
  - on prend le complément restreint et on obtient 1001 1110
  - on ajoute 1
  - (-97) est codé par 1001 1111
- $2^{ème}$ méthode
  - 97 est codé par 0110 0001
  - le premier bit égal à 1 depuis la droite est 0110 000**1**
  - (-97) est codé par 1001 1111

□

Remarque : ces deux méthodes sont réversibles *i.e.* on peut passer du codage d'un entier négatif au codage de son opposé (soit un entier positif) par les mêmes moyens.

***exemple* 3.14** : avec $N = 8$, (-112) est codé par 1001 0000. On retrouve le codage de 112.
- $1^{ère}$ méthode
  - (-112) est codé par 1001 0000
  - on prend le complément restreint et on obtient 0110 1111
  - on ajoute 1
  - 112 est codé par 0111 0000
- $2^{ème}$ méthode
  - (-112) est codé par 1001 0000
  - le premier bit égal à 1 depuis la droite est 100**1** 0000
  - 112 est codé par 0111 0000

□

Il est facile de passer de codages binaires sur N à des codages binaires sur N' avec $N' > N$ en recopiant le bit de plus fort poids sur la gauche autant de fois que nécessaire.

***exemple* 3.15** : : la version binaire de 12 sur 8 bits est 0000 1100. On la convertit en un nombre binaire 16 bits en effectuant 8 copies du bit de poids fort (0) sur la gauche. On obtient 0000 0000 0000 1100.

□

***exemple* 3.16** : : la version binaire de -10 sur 8 bits est 1111 0110. On la convertit en un nombre binaire 16 bits en effectuant 8 copies du bit de poids fort (1) sur la gauche. On obtient 1111 1111 1111 0110.

□

opérations avec le complément à deux

En travaillant en complément à deux, la soustraction sera obtenue par addition de l'opposé en ne tenant pas compte de l'éventuelle retenue qui porterait sur un $(N+1)$-ième bit.

***exemple* 3.17** : $\begin{array}{cccccc} & 0 & 1 & 1 & 1 \\ + & 1 & 1 & 0 & 1 \\ \hline = & 0 & 1 & 0 & 0 \end{array}$ représente la soustraction $\begin{array}{cc} & 7 \\ - & 3 \\ \hline = & 4 \end{array}$ sur 4 bits.

□

***exemple* 3.18** : : Sur 16 bits, effectuer l'addition $6+7$ la soustraction $7-6$ puis la soustraction $6-7$.

$$\begin{array}{ccccc} & 0000 & 0000 & 0000 & 0111 \\ + & 0000 & 0000 & 0000 & 0110 \\ \hline = & 0000 & 0000 & 0000 & 1101 \end{array} \qquad \begin{array}{ccccc} & 0000 & 0000 & 0000 & 0111 \\ + & 1111 & 1111 & 1111 & 1010 \\ \hline = & 0000 & 0000 & 0000 & 0001 \end{array}$$

$$\begin{array}{ccccc} & 0000 & 0000 & 0000 & 0110 \\ + & 1111 & 1111 & 1111 & 1001 \\ \hline = & 1111 & 1111 & 1111 & 1111 \end{array}$$

(on ne tient pas compte de la dernière retenue sur le bit de poids fort).

□

débordement de capacité

Comme on l'a vu, avec N bits le complément à deux permet de coder tous les entiers de -2^{N-1} à $2^{N-1}-1$.

En additionnant deux nombres de N bits, le résultat peut être trop grand pour être représenté sur N bits.

***exemple* 3.19** : avec $N = 8$, l'addition $54 + 100 = 154$ donnerait $\begin{array}{ccc} & 0011 & 0110 \\ + & 0110 & 0010 \\ \hline = & 1001 & 1001 \end{array}$ alors que 1001 1001 représente $-128 + 16 + 8 + 1 = -103$ en complément à deux. Il aurait fallu 9 bits pour représenter 154 en complément à deux.

□

Un tel débordement de capacité peut survenir également lors d'une soustraction.

***exemple* 3.20** : avec $N = 8$, la soustraction $-65 - 74 = -139$ donne $\begin{array}{ccc} & 1011 & 1111 \\ + & 1011 & 0111 \\ \hline = & 0111 & 0110 \end{array}$ alors que 0111 0110 représente $64 + 32 + 16 + 4 + 2 = 118$.

□

Un débordement ne peut survenir que lorsqu'on ajoute des opérandes de même signe.

3.1.4 Représentation des réels

On représentera en fait des nombres décimaux ou même des nombres entiers supérieurs à la capacité de représentation des entiers signés.

***exemple* 3.21** : : $3,15576 \times 10^9$ (nombre de secondes dans un siècle) est plus grand que 2147483647 plus grand entier signé représentable sur 32 bits.

□

flottant

On cherchera à écrire tout d'abord les nombres binaires en notation scientifique normalisée *i.e.* sous la forme $1,xxxxxx_{deux} \times 2^{yyyy_{deux}}$. On dira que l'on représente ainsi un *flottant*.

***exemple* 3.22** : $101,01_{deux} = 1,0101_{deux} \times 2^{10_{deux}}$ signifie $5,25 = 1,3125 \times 4$.

□

$1,0101_{deux} \times 2^{10_{deux}}$ est en notation scientifique normalisée car c'est le produit d'un nombre compris entre 1 et 2 (exclu) et d'un puissance de deux. Ce nombre et l'exposant de la puissance de 2 sont codés en binaire. La partie derrière la virgule – partie fractionnaire – est appelée *mantisse*.

Un tel nombre en notation scientifique normalisée sera codé en trois parties

- signe : + est représenté par sur un bit par 0 et − par le bit 1
- exposant : l'exposant est codé en complément à deux ; s'il occupe k bits (par exemple k=8), on a d'une part les exposants négatifs variant de -2^{k-1} à 0 et d'autre part les exposants positifs variant de 0 à $2^{k-1}-1$
- mantisse : la mantisse est codée sur les $N-k-1$ bits restants. La mantisse est codée par $m_1 m_2 \cdots m_{N-k-1}$ avec $1 + mantisse = 1 + m_1 2^{-1} + m_2 2^{-2} + \cdots m_{N-k-1} 2^{N-k-1}$

***exemple* 3.23** : $5,25 = 101,01_{deux} = 1,0101_{deux} \times 2^{10_{deux}}$, on a alors $signe = 0$, $exposant = 10_{deux}$ et $mantisse = 0101\ 0000 \cdots\ 0000_{deux}$.

□

Donc en numérotant les bits par ordre décroissant à partir du bit de signe on a la représentation suivante sur 32 bits avec 8 bits pour l'exposant :

numéro du bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	···	0
valeur du bit	0	0	0	0	0	0	0	1	0	0	1	0	1	0	···	0

***exemple* 3.24** : $0,5 = \dfrac{1}{2} = 2^{-1} = 1,0 \times 2^{-1}$ donc $signe = 0$, $exposant = -1$ et $mantisse = 0$. Donc en numérotant les bits par ordre décroissant à partir du bit de signe on a la représentation suivante sur 32 bits :

numéro du bit	31	30	29	28	27	26	25	24	23	22	21	20	···	0
valeur du bit	0	1	1	1	1	1	1	1	1	0	0	0	···	0

□

cas particulier : 0 ne peut pas être écrit en notation scientifique normalisée. On lui attribue alors la valeur réservée 0 pour l'exposant.

En résumé, pour $N = 32$, 0 est représenté par

0000 0000 0000 0000 0000 0000 0000 0000

et tous les autres nombres par

$$s \quad exposant_{deux} \quad mantisse_{deux}$$

avec $nombre = (-1)^s \times (1 + mantisse_{deux}) \times 2^{exposant_{deux}}$.

***exemple* 3.25** : donner la représentation binaire de $-0,75$ sur 32 bits avec 8 bits pour l'exposant.

On utilise l'algorithme pour coder les décimaux

$0,75 \times 2 = 1,5$ - écrire $0,\mathbf{1}$

$0,5 \times 2 = 1,0$ - écrire $0,1\mathbf{1}$

puis on en déduit que

$-0,75 = -0,11_{deux} = -\ 1,1_{deux} \times 2^{-1}$ en notation scientifique normalisée.

Le signe moins est codé par 1 pour le bit de plus fort poids. L'exposant vaut -1 et est codé en complément à deux sur les 8 bits 30 à 23. La mantisse vaut 1 en base deux et est codée sur les bits 22 à 0.

Donc $-0,75$ est représenté par

31	30	29	28	27	26	25	24	23	22	21	20	$\cdots$	0
1	1	1	1	1	1	1	1	1	1	0	0	$\cdots$	0

□

En pratique, il faut trouver un juste équilibre entre la taille de l'exposant et la taille de la mantisse.

Accroître la taille de la mantisse *i.e.* augmenter le nombre de bits pour la représenter donne une précision plus élevée.

Accroître la taille de l'exposant augmente l'étendue des nombres pouvant être représentés.

Avec $N = 32$ et $k = 8$ on peut représenter de 2×10^{-38} à 2×10^{38} et de -2×10^{38} à -2×10^{-38} (environ).

La notation scientifique normalisée a trois avantages :

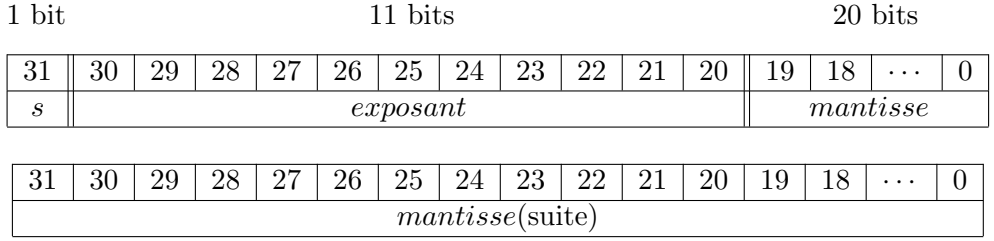
- l'échange de données contenant des nombres flottants est simplifiée
- savoir que les nombres seront toujours sous cette forme simplifie les algorithmes de calcul sur les flottants (arithmétique flottante)
- la précision des nombres que l'on peut stocker dans un mot est accrue car les 0 inutiles avant la virgule sont remplacés par des chiffres significatifs après celle-ci.

double précision

Il peut aussi y avoir des problèmes de débordement dans le cas où l'exposant est trop grand pour être représenté par les k bits fixés.

De façon duale, on parle de *sous-débordement* lorsque la valeur absolue d'un exposant négatif est trop grande pour être représentée par les k bits fixés.

Afin de réduire les risques de débordement et sous-débordement, la plupart des langages de programmation de haut niveau dispose d'une notation avec un plus grand exposant. En **C** on parle de *double précision*. Elle utilise deux mots de 32 bits. Cela correspond au schéma de représentation suivant :



3.1.5 Autres bases

On peut être amené à travailler en *octal* (base 8) ou en *hexadécimal* (base 16). Dans ce dernier cas, pour symboliser les nombres (en base dix) $10, 11, 12, 13, 14, 15$, la convention suivante a été adoptée : ils sont dans l'ordre représentés par les lettres a, b, c, d, e, f.

***exemple* 3.26** : $10 = 12_{huit}$; $10 = a_{seize}$.

Le tableau suivant donne les permiers nombres en base 2, 8, 16 et 10.

binaire	octal	hexadécimal	décimal
0	**0**	**0**	**0**
1	**1**	**1**	**1**
10	**2**	**2**	**2**
11	**3**	**3**	**3**
100	**4**	**4**	**4**
101	**5**	**5**	**5**
110	**6**	**6**	**6**
111	**7**	**7**	**7**
1000	10	**8**	**8**
1001	11	**9**	**9**
1010	12	**a**	10
1011	13	**b**	11
1100	14	**c**	12
1101	15	**d**	13
1110	16	**e**	14
1111	17	**f**	15
10000	20	10	16
10001	21	11	17

□

Suivant que la machine travaille sur 6 bits ou sur 8 bits, on conviendra de regrouper les bits par 3 ou 4 respectivement, ce qui revient à travailler en octal ou en hexadécimal.

***exemple* 3.27** : $10110110111,11101_{deux}$ s'écrira également $2667,72_{huit}$.

$eb7,e8_{seize}$ s'écrira également $10110110111,11101000_{deux}$.

□

En effet, en regroupant par trois (à partir de la virgule vers la gauche et vers la droite) on a :

$10110110111,11101_{deux} = \underbrace{10}_{2}\ \underbrace{110}_{6}\ \underbrace{110}_{6}\underbrace{111,}_{7,}\ \underbrace{111}_{7}\ \underbrace{010}_{2}$

et en regroupant par quatre on a :

$eb7,e8_{seize} = \underbrace{101}_{5}\ \underbrace{1011}_{b}\ \underbrace{0111,}_{7,}\ \underbrace{1110}_{e}\ \underbrace{1000}_{8}$

Pour convertir un entier ou un décimal d'une base quelconque vers la base 10, on procède comme en la base deux.

***exemple* 3.28** : : $652_{sept} = 6 \times 7^2 + 5 \times 7^1 + 2 \times 7^0 = 343$

$3,42_{cinq} = 3 \times 5^0 + 4 \times 5^{-1} + 2 \times 5^{-2} = 3,88$

□

Inversement, pour convertir un entier en base 10 vers son équivalent dans une base quelconque, on divise le nombre par la base puis le quotient ainsi obtenu par la base $\cdots$; le nombre cherché est donné par les restes successifs.

***exemple* 3.29** : $200 : 8 = 25$ - reste 0 - écrire **0**

$25 : 8 = 3$ - reste 1 - écrire **1**0

$3 : 8 = 0$ - reste 3 - écrire **3**10

Donc $200_{dix} = 310_{huit}$

□

Pour convertir un nombre décimal dans une base quelconque, il faut traiter séparément les parties à gauche et à droite de la virgule.

***exemple* 3.30** : on veut convertir $7,2$ en base 3.

$7 : 3 = 2$ - reste 1 - écrire **1**

$2 : 3 = 0$ - reste 2 - écrire **2**1

écrire 21,

$0,2 \times 3 = 0,6$ - écrire $21,$**0**

$0,6 \times 3 = 1,8$ - écrire $21,0$**1**

$0,8 \times 3 = 2,4$ - écrire $21,01$**2**

$0,4 \times 3 = 1,2$ - écrire $21,012$**1**

$0,2 \times 3 = 0,6$ - écrire $21,0121$**0**

$\cdots$

$7,2 = 21,0121012101\cdots_{trois}$

□

3.2 notion d'algorithme

La notion d'algorithme est très ancienne et est intimement liée aux mathématiques. Un algorithme est une méthode de résolution d'un problème décrivant toutes les étapes permettant d'aboutir à la solution.

On cherche généralement des procédés :

- *réalisables* c'est-à-dire dont les étapes sont consituées d'actions que l'on sait mener à bien
- *universels* c'est-à-dire qui s'appliquent à des données variant dans un ensemble potentiellement infini (toutefois, ces données ne seront pas quelconques, leur type dépendra du problème).

3.2.1 Quelques exemples

Exemple 1

Soient a, b deux entiers naturels. Pour calculer la différence $a - b$ avec $a > b$ on peut ajouter 1 à b jusqu'à atteindre a et la somme des 1 que l'on a ajoutés est $a - b$. Bien sûr, ce procédé est réalisable à condition que l'on sache additionner : précisons les différentes étapes.

Soient $a > b$ deux entiers naturels.

- on pose $d = 0$, $c = b$
- **repéter** remplacer c par $c + 1$ et d par $d + 1$ **jusqu'à ce** que $c = a$
- le résultat est d

□

Exemple 2

Soient a, b deux entiers naturels. Pour calculer le produit $a \times b$ on peut ajouter a à lui-même b fois. Cet énoncé est ambigü. Plus précisement

soient $a > b$ deux entiers naturels,

- on pose $p = 0$
- **repéter** b **fois** remplacer p par $p + a$
- le résultat est p

□

De nouveau, ce procédé est réalisable à condition que l'on sache additionner. La principale différence avec l'exemple précédent est dans la répétition : ici on a un nombre déterminé d'additions $p + a$ à effectuer alors que dans l'exemple précédent le nombre d'additions $c + 1$ n'est pas fixé : on sait seulement quand on devra s'arrêter.

Une première question sur ce type d'algorithme se pose : est-on sûr que l'on s'arrêtera ? Dans cet exemple, l'arrêt est certain parce que la structure de $\mathbb{N}$ nous permet de le montrer mais ce n'est pas toujours aussi facile à démontrer.

Un point commun à ces deux exemples est l'utilisation de l'action "remplacer x par $f(x)$" où f est une fonction. Cette action est une **affectation** et consiste à calculer la valeur de $f(x)$ puis changer la valeur de x en lui donnant le résultat de ce calcul. x devient alors une sorte de *contenant* pour des valeurs variables mais la notation x peut aussi signifier le *contenu* courant du contenant x :

on notera l'affectation "remplacer x par $f(x)$" par $x \leftarrow f(x)$ sachant que l'occurrence x de gauche désigne le contenant x et l'occurrence x de droite (dans $f(x)$) désigne le contenu de x soit sa valeur actuelle avant remplacement.

x sera alors défini comme une **variable**.

De façon plus générale les actions effectuées par un algorithme – dont font partie les affectations – seront appelées des **instructions**.

L'action de répétition est une instruction différente d'une affectation.

On peut aussi considérer que les instructions du type "on pose $p = 0$" sont des affectations, la valeur affectée étant une valeur constante et non pas calculée à partir d'une variable.

Il est important de noter que cette affectation est indispensable au bon fonctionnement de l'algorithme. Sans elle, la première instruction $p \leftarrow p + a$ n'a pas de sens puisque la valeur actuelle de p n'est pas connue. On dira que la première affectation d'une variable est une **initialisation**.

Exemple 3

Soit une liste L (finie) d'entiers. On cherche quel est le plus grand parmi eux. Il suffirait de tous les examiner un par un et de garder le plus grand mais cela reste très vague comme solution. Précisons les étapes :

soient $x_1 \dots x_n$ n entiers ,

- on pose $m \leftarrow x_1$
- **repéter pour** i **variant de 2 à** n, **si** $x_i > m$ **alors** $m \leftarrow x_i$
- le résultat est m

De nouveau dans cet exemple on a une répétition un nombre déterminé de fois mais cette répétition est *indexée* par une variable i qui prendra les valeurs successives $2, \dots, n$.

Pour chaque valeur prise par i un *test* de comparaison entre x_i et m (**si** $x_i > m$) est effectué puis dans le cas où x_i est effectivement supérieur à m l'affectation $m \leftarrow x_i$ est exécutée.

On peut réécrire cet algorithme ainsi

soient $x_1 \dots x_n$ n entiers ,

- on pose $m \leftarrow x_1$, $i \leftarrow 2$
- **repéter**
  - **si** $x_i > m$ **alors** $m \leftarrow x_i$
  - $i \leftarrow i + 1$

  jusqu'à ce que $i = n + 1$
- le résultat est m

Dans cette version, l'évolution de la variable i est dirigée par l'instruction $i \leftarrow i + 1$. □

Exemple 4

On va maintenant utiliser nos connaissances du chapitre 2 pour envisager le problème suivant dit de satisfiabilité.

Soit f une fonction booléenne d'arité n de $\mathbb{B}^n$ dans $\mathbb{B}$. Le problème est de déterminer s'il existe un n-uplet $(x_1 \dots x_n)$ de $\mathbb{B}^n$ tel que $f(x_1 \dots x_n) = 1$. Un algorithme simple consiste à calculer $f(x_1 \dots x_n)$ pour tous les n-uplets $(x_1 \dots x_n) \in \mathbb{B}^n$ pour répondre.

Mais le cardinal de $\mathbb{B}^n$ est 2^n ...

Supposons que nous puissions calculer $f(x_1 \dots x_n)$ pour une valeur du n-uplet $(x_1 \dots x_n)$ en 1 μs soit $10^{-6} s$. Dans la table suivante on donne alors une approximation de la durée en s de l'exécution de l'algorithme.

$n =$	10	20	30	60
$2^n \approx$	$1,02 \cdot 10^3$	$1,05 \cdot 10^6$	$1,07 \cdot 10^9$	$1,15 \cdot 10^{18}$
durée $\approx$	$1,02 \cdot 10^{-3}$	$1,05$	$1,07 \cdot 10^3$	$1,15 \cdot 10^{12}$

Notons que $1,15 \cdot 10^{12}$ est supérieur à 36000 ans. □

Une deuxième question se pose maintenant : un tel algorithme est-il efficace ?

Dans ce cas il clair que non.

3.2.2 Efficacité des algorithmes

Ce dernier exemple nous montre qu'une solution algorithmique peut être réalisable théoriquement mais pas en pratique.

Il existe de nombreux problèmes pour lesquels la réalisation d'un algorithme n'est pas "raisonnable".

Exemple 5

Supposons que l'on a placé divers commutateurs téléphoniques en divers points d'une région et que l'on veuille maintenant les relier par des câbles de façon la plus économique possible : il faut alors tenir compte du coût de construction pour relier chaque paire de commutateurs.

Par exemple, si on 3 commutateurs A,B,C tels que les coûts de pose des câbles soient
- pour relier A et B de 12 unités,
- pour relier A et C de 7 unités,
- pour relier B et C de 3 unités.

alors il est clair que que l'on choisira de câbler A et C puis B et C pour un coût total de 10 unités (le câblage de A et B est inutile puis que A et B sont reliés par l'intermédiaire de C).

La problème se complique vite avec le nombre de commutateurs ; si on a 4 commutateurs A,B,C,D tels que les coûts de pose des câbles soient
- pour relier A et B de 12 unités,
- pour relier A et C de 7 unités,
- pour relier A et D de 5 unités,
- pour relier B et C de 6 unités,
- pour relier B et D de 10 unités,
- pour relier C et D de 5 unités,

alors on choisira de câbler A et D puis D et C puis C et B pour un coût total de 5+5+6=11 unités (on peut vérifier que les commutateurs sont reliés).

Une solution consiste donc à faire la liste de toutes les paires de commutateurs et d'en choisir un nombre nécessaire et suffisant de façon à ce que tous les commutateurs soient reliés et que le coût soit minimal.

Sans entrer dans les détails techniques, s'il y a n commutateurs et donc $n(n-1)/2$ paires, il faudra choisir $n-1$ paires qui relient tous les commutateurs : un tel choix est appelé un *arbre couvrant*.

Or il y a n^{n-2} arbres couvrants.

Par conséquent il ne reste qu'à examiner tous ces arbres couvrants un par un pour choisir le moins coûteux. □

Voyons maintenant combien de temps (en s) prend cet algorithme en supposant qu'il faut $1\mu s$ pour déterminer un arbre couvrant.

$n =$	10	20	30
$n^{n-2} \approx$	10^8	20^{18}	30^{28}
durée $\approx$	10^2	$2,6 \cdot 10^{17}$	$2,3 \cdot 10^{63}$

On dépasse largement les 36 000 ans précédent. □

Pour ce problème particulier dit de l'*arbre couvrant minimal* il existe plusieurs algorithmes tout à fait réalisables.

De façon plus générale, on parle de la *complexité* d'un algorithme lorsque l'on cherche à évaluer les ressources nécessaires à l'exécution de l'algorithme. Cette complexité sera déterminée en fonction d'une donnée initiale dont la taille sera considérée comme une variable (entière, réelle $\cdots$).

Le terme ressource peut recouvrir plusieurs sens :
- on peut chercher à déterminer le temps de calcul qui est alors lié au nombre d'opérations,
- on peut chercher à déterminer la capacité de mémoire nécessaire pour stocker toutes les données intermédiaires au cours de l'exécution.

Cela mène à un classement des problèmes dans des classes dont les plus célèbres sont **P** et **NP**.

3.2.3 Limitation des algorithmes

Comme nous venons de le voir tous les algorithmes ne sont pas réalisables en pratique. Il existe des problèmes pour lesquels on ne connaît pas d'algorithme de complexité raisonnable.

Il existe aussi des problèmes pour lesquels on sait qu'il n'existe pas d'algorithme : on parle alors de problème *indécidable*.

Donc confronté à un problème,
- on cherche à savoir s'il est décidable,
- puis on cherche une solution (un algorithme) dont on essaiera de montrer qu'elle est correcte (en particulier que le calcul termine dans tous les cas et avec une réponse correcte)
- enfin on cherche à calculer la complexité de l'algorithme.

Finalement, ayant passé toutes ces étapes, on peut enfin *implémenter* l'algorithme sous la forme d'un *programme* dans son langage de programmation favori.