DE LA LOGIQUE AVANT TOUTE CHOSE

À la fin de ce chapitre, je sais :

- ☞ formuler des propositions logiques à partir du langage naturel
- ☞ utiliser les connecteurs logiques pour relier des variables
- ☞ établir une table de vérité
- ☞ utiliser les lois de Morgan, le tiers exclu et la décomposition de l'implication
- ☞ mettre une formule propositionnelle sous une forme normale
- ☞ étudier la satisfaisabilité d'une formule propositionnelle

A Logique et applications

Quelque soit le sujet, les êtres humains ont tendance à chercher s'assurer qu'ils ont raison ou que les autres ont tord, ne serait-ce que pour des raisons d'ego [1] ou de commerce [2]. Dans le cadre de la science et de l'ingénierie, il est fondamental de pouvoir apporter la preuve que le raisonnement tenu est correct, car c'est ainsi que la science progresse collectivement et ainsi que l'ingénierie garantit le bon fonctionnement de l'objet produit. Aujourd'hui,tous les domaines de l'industrie sont dépendants de la logique. On peut citer par exemple : la planification (logistique, organisation des tâches), la satisfaction de contraintes multiples, l'élaboration de diagnostiques, la vérification formelle de modèles de systèmes complexes ou l'élaboration et vérification des circuits électroniques.

Une démarche scientifique exige un raisonnement formalisé : c'est l'objet de la logique et du calcul propositionnel qui est présenté dans ce chapitre. Ce formalisme est né au milieu du XIX^{e} siècle grâce au travaux de Boole et a été finalisé au cours de la première moitié du XX^{e} siècle par Frege, Russel et Gödel.

■ **Définition 1 — Logique.** Du grec *logos*, la raison. La logique en tant que discipline scientifique fournit les outils nécessaires à la construction d'un raisonnement : elle permet de manipuler des concepts et d'enchaîner les déductions. Les vérités logiques ne concernent aucun domaine de connaissance en particulier : elles sont valides en amont de toute vérité

1. D'abord, j'ai toujours raison!
2. Mon collègue est-il en train de m'arnaquer?

scientifique particulière empirique [a].

a. c'est-à-dire issue de l'expérience du monde réel

■ **Exemple 1 — Structure d'un raisonnement.** Considérons l'énoncé suivant :

> Si un professeur est un super-héros et que je suis un professeur, alors je suis un super-héros.

Cet énoncé est vrai et inspire une observation fondamentale : la notion de vérité que l'on peut lui associer ne repose que sur sa structure, sa forme, c'est-à-dire Si ... et ..., alors .... On pourrait en effet remplacer le terme *professeur* par *élève*, le terme *super-héros* par *star* et le terme *je* par *tu* et malgré tout, l'énoncé serait vrai.

> Si un élève est une star et que tu es un élève, alors tu es une star.

On peut donc **construire** des énoncés logiques formels à l'aide de connecteurs : Si, et, ou, alors, ne ... pas,donc.... Par exemple, on peut combiner :

p_1 le ciel est bleu

et

p_2 je me promène au bord de l'océan

ainsi :

p_3 Si le ciel est bleu, alors je me promène au bord de l'océan.

Formellement, on peut modéliser cet énoncé logique ainsi : $p_3 = (p_1 \longrightarrow p_2)$.

■ **Définition 2 — Proposition simple ou atomique.** Une proposition simple est une expression vraie ou fausse.

La logique des propositions est limitée : elle ne peut pas prendre en compte des énoncés quantifiés [3] comme dans l'exemple 2.

■ **Exemple 2 — Prédicat et raisonnement quantifié** ⇢ **Hors Programme** . Considérons l'énoncé suivant :

> Si tous les professeurs sont des super-héros et que je suis un professeur, alors je suis un super-héros.

La logique des propositions ne peut pas modéliser cet énoncé à cause du quantificateur universel *tous les*. En effet, la valeur de vérité de cette proposition dépend de ce quantificateur. Que se passe-t-il si un professeur n'est pas un super-héros ? Ce n'est plus la même proposition. Une proposition simple ne contient donc pas de quantificateur. Pour modéliser ce type d'énoncé, on utilise les prédicats, ceux que vous manipulez en mathématiques : supposons que S est le prédicat unaire *super-héros*, P le prédicat unaire *professeur*. On peut

3. On parle alors de logique du premier ordre, logique des prédicats ⇢ HORS PROGRAMME

écrire le prédicat suivant :

$$(\forall x, P(x) \longrightarrow S(x)) \wedge P(x) \longrightarrow S(x)$$

La suite de ce chapitre expose donc la logique des propositions. Elle débute par la définition des briques de bases que sont les constantes, les variables propositionnelles et les opérateurs logiques. Puis elle énonce la syntaxe de la logique propositionnelle en donnant la définition inductive des formules logiques. La définition de la valuation d'une formule permet de donner un sens à la logique propositionnelle et de définir une sémantique. Enfin, le chapitre aborde les problèmes de satisfaisabilité d'une formule logique.

B Constantes, variables propositionnelles et opérateurs logiques

Dans cette section est détaillé les éléments de l'ensemble qui forme la base des formules logiques.

■ **Définition 3 — Constante universelle $\top$.** La constante $\top$ désigne le vrai. On peut la voir comme un opérateur d'arité nulle.

■ **Définition 4 — Constante vide $\bot$.** La constante $\bot$ désigne la contradiction. On peut la voir comme un opérateur d'arité nulle.

Vocabulary 1 — Top et Bottom ⇜⇝ En anglais, la constante universelle $\top$ se dit *Top* et la constante vide $\bot$ Bottom.

■ **Définition 5 — Variable propositionnelle.** Une variable propositionnelle est une proposition atomique (cf. définition 2).

C Opérateurs logique et notations

L'étude des structures des propositions permet de définir opérateurs qui relient les formules logiques : la négation *non*, le *et*, le *ou*, l'*implication* et l'*équivalence*.

■ **Définition 6 — Opérateur.** Un opérateur est une fonction qui réalise une opération primitive dans un langage.

■ **Exemple 3 — Opérateurs.** Les symboles mathématiques $+, -, \times$ sont des opérateurs pour l'arithmétique des entiers naturels : $2+3\times5$. On utilise également ces symboles dans le cas de l'arithmétique des polynômes : $P+Q$. Cependant, il faut bien observer que ce ne sont pas les mêmes opérateurs.

De la même manière, en informatique, les symboles `+`, `-`, `*` représentent les opérateurs arithmétiques sur les entiers en Python. Même si on utilise les mêmes symboles pour faire les opérations de base sur les flottants, il faut bien remarquer que le langage offre ici ce

qu'on appelle du sucre syntaxique, une facilité. Car ce sont en fait des opérateurs différents. D'ailleurs, en OCaml, les opérateurs arithmétiques sur les entiers `+`, $-$, `*` ne sont pas les opérateurs sur les flottants `+.`, `-.`, `*.`.

■ **Définition 7 — Arité d'un opérateur.** L'arité d'un opérateur est le nombre de paramètres que prend sa fonction sous-jacente. On distingue les opérateurs unaires (une seule opérande) et les opérateurs binaires (deux opérandes).

■ **Exemple 4 — Arités des opérateurs arithmétiques.** On dit que l'addition ou la multiplication sont des opérateurs binaires car ils prennent deux opérandes en entrée.

En informatique, on distingue l'opérateur binaire $-$ de l'opérateur unaire $-$. Le premier réalise la soustraction des deux opérandes `a - b`, le second change le signe de l'opérande $-$`a`.

■ **Définition 8 — Notation infixe des opérateurs (connecteurs).** Dans le cadre d'une notation infixe des opérateurs binaires, l'opérateur est placé entre ses opérandes.

■ **Définition 9 — Notation préfixe des opérateurs (constructeurs).** Dans le cadre d'une notation préfixe des opérateurs binaires, l'opérateur est placé devant ses opérandes.

■ **Définition 10 — Négation (NON).** L'opérateur négation consiste à calculer la valeur opposée de son opérande.

- Connecteur : $\neg$
- Constructeur : `not`
- Notation informatique fréquente : not ou !

■ **Définition 11 — Conjonction (ET).** L'opérateur conjonction a pour résultat vrai si ses deux opérandes sont vraies.

- Connecteur : $\wedge$
- Constructeur : `and`
- Notation informatique fréquente : `&&` ou `and`

■ **Définition 12 — Disjonction (OU).** L'opérateur disjonction a pour résultat vrai si une des deux opérandes est vraies.

- Connecteur : $\vee$
- Constructeur : `or`
- Notation informatique fréquente : `||` ou `or`

R Les opérateurs négation, conjonction et disjonction sont les opérateurs premiers : ils permettent d'exprimer tous les autres.

■ **Définition 13 — Implication matérielle ($\Longrightarrow$).** L'opérateur implication logique a pour résultat faux seulement si la seconde opérande est fausse.

- Connecteur : $\Longrightarrow$
- Constructeur : `imp`

R L'implication matérielle est le seul opérateur de base en logique minimale.

■ **Définition 14 — Équivalence matérielle ($\Longleftrightarrow$).** L'opérateur équivalence matérielle est équivalent à une implication dans les deux sens.

- Connecteur : $\Longleftrightarrow$
- Constructeur : `eq`

■ **Exemple 5 — Notation infixe et préfixe d'une même formule logique.** Voici une même formule logique décrite à l'aide de variables propositionnelles et d'opérateurs infixes ou préfixes.

- avec les connecteurs : $(c \Longrightarrow b) \vee (\neg c \wedge b)$
- avec les constructeurs : or(imp(c,b),et(not(c), b))

Même s'il est possible de programmer des connecteurs, les informaticiens préfèrent les constructeurs pour plusieurs raisons :

- ils sont plus faciles à implémenter,
- ils donnent à l'ensemble des formules une structure arborescente facile à analyser.

R Les opérateurs possèdent un ordre de priorité, de la plus forte à la plus faible :

$$\neg > \wedge > \vee$$

Cela signifie qu'en cas d'ambiguïté sur une formule sans parenthèses, cet ordre permet trancher l'interprétation : on effectue d'abord la négation puis la conjonction et enfin la disjonction.

D Formules logiques : définition inductive et syntaxe

On considère un ensemble des variables propositionnelles $\mathcal{V}$ utilisé pour écrire un ensemble de formules $\mathcal{F}$ en logique propositionnelle.

■ **Définition 15 — Ensemble des formules propositionnelles $\mathcal{F}$ (défini inductivement).** L'ensemble $\mathcal{F}$ des formules propositionnelles sur $\mathcal{V}$ est défini inductivement comme suit :

$$\bot \in \mathcal{F} \quad \text{(Base)} \tag{1}$$
$$\top \in \mathcal{F} \quad \text{(Base)} \tag{2}$$
$$\forall x \in \mathcal{V}, x \in \mathcal{F} \quad \text{(Base)} \tag{3}$$
$$\forall \phi \in \mathcal{F}, \text{not}(\phi) \in \mathcal{F} \quad \text{(Constructeur négation)} \tag{4}$$
$$\forall \phi \in \mathcal{F}, \forall \psi \in \mathcal{F}, \text{and}(\phi, \psi) \in \mathcal{F} \quad \text{(Constructeur conjonction)} \tag{5}$$
$$\forall \phi \in \mathcal{F}, \forall \psi \in \mathcal{F}, \text{or}(\phi, \psi) \in \mathcal{F} \quad \text{(Constructeur disjonction)} \tag{6}$$

Cela signifie qu'une formule logique est soit :

- une constante universelle ou vide,
- une variable propositionnelle,
- une négation d'une formule logique,
- une conjonction ou une disjonction de formules logiques.

R L'intérêt principal de la définition inductive est qu'elle permet de **construire des formules logiques qui sont des objets informatiques avec lesquels on peut calculer**.

■ **Définition 16 — Formule atomique.** Une formule ϕ est atomique si ϕ est $\bot$, $\top$ ou une variable propositionnelle.

R Une formule logique de $\mathcal{F}$ peut être représentée par un arbre comme l'illustre la figure 1 : cette forme est très importante pour les compilateurs et les outils d'analyse de code en général. On la désigne par le terme arbre syntaxique. En parcourant un tel arbre, on peut calculer une formule logique. C'est pourquoi, la structure d'arbre est au programme et sera détaillée dans les prochains chapitres.

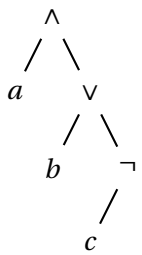

FIGURE 1 – Arbre représentant la formule logique $a \wedge (b \vee \neg c)$

■ **Définition 17 — Ensemble des sous-formules.** Soit ϕ une formule logique et $\mathcal{V}$ l'ensemble des ses variables propositionnelles. On définit l'ensemble des sous-formules de ϕ par la fonction sf :

$$\forall \phi \in \{\bot, \top, \} \cup \mathcal{V}, sf(\phi) = \phi \quad \text{(Base)} \tag{7}$$

$$\forall \phi \in \mathcal{F}, \exists \psi \in \mathcal{F}, \phi = \neg\psi \Longrightarrow sf(\phi) = \{\phi\} \cup sf(\psi) \quad \text{(Constructeur négation)} \tag{8}$$

$$\forall \phi \in \mathcal{F}, \exists \psi, \xi \in \mathcal{F}, \phi = \psi \wedge \xi \Longrightarrow sf(\phi) = \{\phi\} \cup sf(\psi) \cup sf(\xi) \quad \text{(Constructeur conjonction)} \tag{9}$$

$$\forall \phi \in \mathcal{F}, \exists \psi, \xi \in \mathcal{F}, \phi = \psi \vee \xi \Longrightarrow sf(\phi) = \{\phi\} \cup sf(\psi) \cup sf(\xi) \quad \text{(Constructeur disjonction)} \tag{10}$$

Une constante ou une variable propositionnelle est une sous-formule. Toutes les formules qui permettent de construire une formule logique sont des sous-formules.

■ **Définition 18 — Taille d'une formule.** La taille d'une formule logique est le nombre d'opérateurs qui construisent la formule. On la note $|\phi|$ et on la définit inductivement par :

$$|\bot| = 0 \quad \text{(Base)} \tag{11}$$

$$|\top| = 0 \quad \text{(Base)} \tag{12}$$

$$\forall x \in \mathcal{V}, |x| = 0 \quad \text{(Base)} \tag{13}$$

$$|\neg\phi| = 1 + |\phi| \quad \text{(Constructeur négation)} \tag{14}$$

$$|\phi \diamond \psi| = 1 + |\phi| + |\psi| \quad \text{(Constructeur conjonction ou disjonction } \diamond\text{)} \tag{15}$$

■ **Définition 19 — Hauteur d'une formule.** La hauteur d'une formule logique est la hauteur de son arbre syntaxique. On la note $h(\phi)$ et elle est définie inductivement :

$$h(\bot) = 0 \quad \text{(Base)} \tag{16}$$

$$h(\top) = 0 \quad \text{(Base)} \tag{17}$$

$$\forall x \in \mathcal{V}, h(x) = 0 \quad \text{(Base)} \tag{18}$$

$$h(\neg\phi) = 1 + h(\phi) \quad \text{(Constructeur négation)} \tag{19}$$

$$h(\phi \diamond \psi) = 1 + \max(h(\phi, h(\psi)) \quad \text{(Constructeur conjonction ou disjonction } \diamond\text{)} \tag{20}$$

E Sémantique et valuation des formules logiques

■ **Définition 20 — Ensemble des valeurs de vérité.** L'ensemble des valeurs de vérité est un ensemble à deux éléments que l'on peut noter de différentes manières :

$$\mathbb{B} = \{0,1\} = \{F,V\} = \{\text{Faux},\text{Vrai}\} = \{\text{F},\text{T}\} \tag{21}$$

■ **Définition 21 — Valuation ou interprétation.** Une valuation de $\mathcal{V}$ est une distribution des valeurs de vérité sur l'ensemble des variables propositionnelles $\mathcal{V}$, soit une fonction $v : \mathcal{V} \longrightarrow \mathbb{B}$.

■ **Définition 22 — Évaluation d'une formule logique (définie inductivement).** Soit v une valuation de $\mathcal{V}$: $v : \mathcal{V} \longrightarrow \mathbb{B}$. L'évaluation d'une formule logique d'après v est notée $[\![\phi]\!]_v$. Elle est définie inductivement par :

$$[\![\bot]\!]_v = \text{F} \qquad \text{(Base)} \tag{22}$$
$$[\![\top]\!]_v = \text{V} \qquad \text{(Base)} \tag{23}$$
$$\forall x \in \mathcal{V}, [\![x]\!]_v = v(x) \qquad \text{(Base)} \tag{24}$$
$$\forall \phi \in \mathcal{F}, [\![\neg\phi]\!]_v = \neg [\![\phi]\!]_v \qquad \text{(Constructeur négation)} \tag{25}$$
$$\forall \phi \in \mathcal{F}, \forall \psi \in \mathcal{F}, [\![\phi \wedge \psi]\!]_v = [\![\phi]\!]_v \wedge [\![\psi]\!]_v \qquad \text{(Constructeur conjonction)} \tag{26}$$
$$\forall \phi \in \mathcal{F}, \forall \psi \in \mathcal{F}, [\![\phi \vee \psi]\!]_v = [\![\phi]\!]_v \vee [\![\psi]\!]_v \qquad \text{(Constructeur disjonction)} \tag{27}$$
$$\tag{28}$$

R Pour être capable d'évaluer une formule logique, il faut donc pouvoir évaluer des négations, des conjonctions et des disjonctions sur des valeurs de vérité. Les tables de vérités des opérateurs premiers sont données sur les tableaux 1, 2, 3, 4 et 5.

a	$\neg a$
F	V
V	F

TABLE 1 – Table de vérité de l'opérateur négation

a	b	$a \wedge b$
F	F	F
F	V	F
V	F	F
V	V	V

TABLE 2 – Table de vérité de l'opérateur conjonction

a	b	$a \vee b$
F	F	F
F	V	V
V	F	V
V	V	V

TABLE 3 – Table de vérité de l'opérateur disjonction

a	b	$a \Longrightarrow b$
F	F	V
F	V	V
V	F	F
V	V	V

TABLE 4 – Table de vérité de l'opérateur implication matérielle

a	b	$a \Longleftrightarrow b$
F	F	V
F	V	F
V	F	F
V	V	V

TABLE 5 – Table de vérité de l'opérateur équivalence matérielle

■ **Définition 23 — Modèle.** Un modèle pour une formule logique ϕ est une valuation ν telle que :

$$[\![\phi]\!]_\nu = \mathrm{V} \tag{29}$$

■ **Définition 24 — Conséquence sémantique $\models$.** Soit ϕ et ψ deux formules de $\mathcal{F}$. On dit que ψ est une conséquence sémantique de ϕ si tout modèle de ϕ est un modèle de ψ. On note alors : $\phi \models \psi$

Une formule ψ peut également être la conséquence sémantique d'un ensemble de formules Γ. On note alors : $\Gamma \models \psi$.

■ **Définition 25 — Équivalence sémantique $\equiv$.** Deux formules logiques ϕ et ψ sont équivalentes sémantiquement si quelle que soit la valuation choisie, l'évaluation des deux formules produit le même résultat. On note cette équivalence $\phi \equiv \psi$.

Plus formellement,

$$\phi \equiv \psi \iff \forall \nu : \mathcal{V} \longrightarrow \mathbb{B}, [\![\phi]\!]_\nu = [\![\psi]\!]_\nu \tag{30}$$

■ **Définition 26 — Tautologie.** Une formule ϕ toujours vraie quel que soit le modèle d'interprétation est une tautologie. On la note $\top$ ou $\models \phi$.

R Deux formules sont donc équivalentes d'un point de vue sémantique si et seulement si leur équivalence matérielle est une tautologie : $\models (\phi \iff \psi)$.

■ **Définition 27 — Antilogie.** Une formule a toujours fausse quel que soit le modèle d'interprétation est une antilogie. On la note $\bot$.

■ **Définition 28 — Formule satisfaisable.** Soit ϕ une formule logique de $\mathcal{F}$. S'il existe une valuation ν de $\mathcal{V}$ qui satisfait ϕ, alors ϕ est dite satisfaisable.

Plus formellement :

$$\phi \text{ est une formule satisfaisable} \iff \exists \nu : \mathcal{V} \longrightarrow \mathbb{B}, [\![\phi]\!]_\nu = \mathrm{V} \tag{31}$$

F Lois de la logique

Les lois de la logique sont des équivalences sémantiques.

a Éléments neutres et absorbants, idempotence

$$a \wedge \top \equiv a \tag{32}$$
$$a \wedge \bot \equiv \bot \tag{33}$$
$$a \vee \top \equiv \top \tag{34}$$
$$a \vee \bot \equiv a \tag{35}$$
$$a \wedge a \equiv a \tag{36}$$
$$a \vee a \equiv a \tag{37}$$

b Commutativité, distributivité, associativité

$$a \wedge b \equiv b \wedge a \tag{38}$$
$$a \vee b \equiv b \vee a \tag{39}$$
$$a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c) \tag{40}$$
$$a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c) \tag{41}$$
$$a \wedge (b \wedge c) \equiv (a \wedge b) \wedge c \tag{42}$$
$$a \vee (b \vee c) \equiv (a \vee b) \vee c \tag{43}$$

c Lois de De Morgan

$$\neg(a \wedge b) \equiv \neg a \vee \neg b \tag{44}$$
$$\neg(a \vee b) \equiv \neg a \wedge \neg b \tag{45}$$

d Décomposition des opérateurs

$a \Longrightarrow b \equiv \neg a \vee b$	Implication et opérateurs premiers	(46)
$a \Longrightarrow b \equiv \neg b \Longrightarrow \neg a$	Contraposition	(47)
$(a \wedge b) \Longrightarrow c \equiv a \Longrightarrow (b \Longrightarrow c)$	Curryfication	(48)

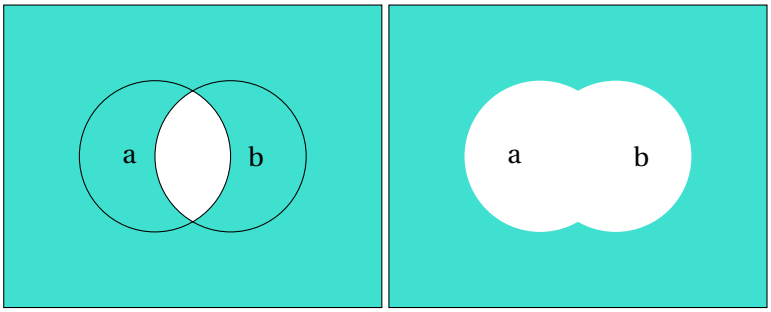

FIGURE 2 – Illustration des lois de De Morgan

e Démonstrations

Pour démontrer les lois précédentes, on peut produire les tables de vérités correspondantes puis utiliser les lois déjà prouvées pour démontrer les autres. Ces démonstrations constituent d'excellents exercices.

G Principes et logique classique

$a \vee \neg a \equiv \top$	Principe du tiers exclu	(49)
$a \wedge \neg a \equiv \bot$	Principe de non-contradiction	(50)
$\neg\neg a \equiv a$	Double négation	(51)
$\bot \Longrightarrow a$	Principe d'explosion	(52)
		(53)

R Il existe plusieurs logiques qui se différencient principalement par la manière de gérer les déductions de l'absurde :

- La logique minimale n'utilise qu'un seul connecteur : l'implication. Elle a pour caractéristique de ne rien déduire de $\bot$. Cela signifie qu'elle n'inclut ni le principe du tiers exclu, ni le principe d'explosion.
- La logique classique utilise les opérateurs définis dans ce cours. Elle a pour caractéristique d'inclure les principes ci-dessus et permet donc de conduire des raisonnements par l'absurde. La critique faite à ce type de raisonnement, c'est qu'il permet d'accepter l'existence d'un concept sans pouvoir le construire explicitement.
- La logique intuitionniste est constructive : la notion de preuve constructive remplace la notion de vérité. Constuire un concept, c'est exhiber sa preuve d'existence. Si un concept ne peut pas être établi par une preuve constructive, cela signifie qu'il n'existe pas. La

logique intuitionniste distingue le *être vrai* du *ne pas être faux*. C'est pourquoi elle n'inclut ni le principe du tiers exclu, ni le raisonnement par l'absurde, ni la double négation. Elle inclut par contre le principe d'explosion.

H Formes normales

■ **Définition 29 — Littéral.** Un littéral est une variable propositionnelle ou sa négation.

■ **Définition 30 — Clause conjonctive.** Une clause conjonctive est une conjonction de littéraux.

■ **Définition 31 — Forme normale disjonctive(FND).** Une forme normale disjonctive d'une formule logique est une disjonction de clauses conjonctives.

Théorème 1 — Toute formule logique est équivalente à une forme normale disjonctive.

Démonstration. En fait, il suffit d'écrire que cette formule logique est la disjonction de toutes ses valuations vraies. Plus formellement :

$$\phi \equiv \bigvee_{\substack{\nu, \\ [\![\phi]\!]_\nu = \mathrm{V}}} \bigwedge_{x \in \mathcal{V}} x \tag{54}$$

■

■ **Exemple 6 — Lien entre la table de vérité et la forme disjonctive complète.** On considère la formule logique $\phi = (a \vee b) \wedge ((c \Longrightarrow b) \vee a)$. Sa table de vérité contient l'ensemble possible de ses valuations vraies et fausses :

a	b	c	$a \vee b$	$c \Longrightarrow b$	$(c \Longrightarrow b) \vee a$	ϕ
F	F	F	F	F	F	F
F	F	V	F	F	F	F
F	V	F	V	V	V	V
F	V	V	V	V	V	V
V	F	F	V	F	V	V
V	F	V	V	F	V	V
V	V	F	V	V	V	V
V	V	V	V	V	V	V

Par conséquence, en prenant la disjonction de toutes les modèles, on obtient la FND :

$$\phi \equiv (a \wedge b \wedge c) \vee (a \wedge b \wedge \bar{c}) \vee (a \wedge \bar{b} \wedge c) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (\bar{a} \wedge b \wedge c) \vee (\bar{a} \wedge b \wedge \bar{c})$$

■ **Définition 32 — Clause disjonctive.** Une clause disjonctive est une disjonction de littéraux.

■ **Définition 33 — Forme normale conjonctive (FNC).** Une forme normale conjonctive d'une formule logique est une conjonction de clauses disjonctives.

Théorème 2 — Toute formule logique est équivalente à une forme normale conjonctive.

Démonstration. Soit ϕ une formule logique. On considère sa négation $\neg\phi$. D'après la question précédente, on peut mettre $\neg\phi$ sous une forme normale disjonctive, c'est à dire

$$\neg\phi \equiv c_1 \vee c_2 \ldots \vee c_n \tag{55}$$

où les $c_i = l_1 \wedge l_2 \wedge \ldots \wedge l_m$ sont des conjonctions de littéraux. En appliquant la loi de Morgan, on trouve que :

$$\begin{aligned}
\neg\neg\phi &\equiv (\neg c_1) \wedge (\neg c_2) \wedge \ldots \wedge (\neg c_n) && (56)\\
&\equiv (l_1 \vee l_2 \ldots \vee l_m) \wedge (\neg c_2) \wedge \ldots \wedge (\neg c_n) && (57)\\
&\equiv d_1 \wedge d_2 \wedge \ldots \wedge d_n && (58)\\
&\equiv \phi && (59)
\end{aligned}$$

où les d_i sont des disjonctions. Donc ϕ peut s'écrire sous une forme normale conjonctive. ■

■ **Exemple 7 — FNC équivalente.** Soit la formule logique $\phi = (a \vee \neg b) \wedge \neg(c \wedge \neg(d \wedge e))$. Une forme normale équivalente est $(a \vee \neg b) \wedge (\neg c \vee d) \wedge (\neg c \vee e)$. Celle-ci est obtenue en utilisant les lois logiques pour changer la forme de la formule.

■ **Exemple 8 — Table de vérité et forme normale conjonctive.** On considère la formule logique $\phi = (a \vee b) \wedge ((c \Longrightarrow b) \vee a)$. Sa table de vérité est toujours :

a	b	c	$a \vee b$	$c \Longrightarrow b$	$(c \Longrightarrow b) \vee a$	ϕ
F	F	F	F	F	F	F
F	F	V	F	F	F	F
F	V	F	V	V	V	V
F	V	V	V	V	V	V
V	F	F	V	F	V	V
V	F	V	V	F	V	V
V	V	F	V	V	V	V
V	V	V	V	V	V	V

Pour trouver la forme normale conjonctive, il faut sélectionner les lignes des contre-modèles de la formule. Puis, pour chaque littéral d'un contre-modèle, prendre la négation de sa valeur et construire une clause disjonctive. Enfin, prendre la conjonction de ces clauses. Par conséquence, on a donc la FNC :

$$\phi \equiv (a \vee b \vee \neg c) \wedge (a \vee b \vee c)$$

En utilisant la distributivité, on obtient :

$$\phi \equiv (a \vee b) \vee (\neg c \vee c) \equiv a \vee b$$

I Problème SAT

■ **Définition 34 — Problème de décision.** Un problème de décision est un problème dont la réponse est binaire : soit le on peut le décider, soit on ne peut pas.

■ **Définition 35 — Problème SAT.** Le problème de satisfaisabilité booléenne (SAT) est un problème de décision lié à une formule de logique propositionnelle et dont l'objectif est de déterminer s'il existe une valuation qui rend la formule vraie.

On note $\text{SAT}(\psi) = V$ si ψ est satisfaisable, et F sinon.

■ **Exemple 9 — Exemples de problème SAT.** Pour définir la date d'une réunion, on considère les contraintes suivantes :

- Johann est obligé d'assister à ses cours lundi, mercredi ou jeudi.
- Cécile ne peut pas se libérer mercredi,
- Annaïg est prise le vendredi
- Prosper n'est là ni le mardi ni le jeudi.

Est-il possible de trouver un jour pour fixer la réunion ? Il s'agit d'un problème de satisfaisabilité de la formule logique :

$$(\neg L \vee Ma \vee \neg Me \vee \neg J \vee V) \wedge (L \vee Ma \vee \neg Me \vee J \vee V) \wedge (L \vee Ma \vee Me \vee J \vee \neg V) \wedge (L \vee \neg Ma \vee Me \vee \neg J \vee V)$$

La méthode de résolution d'un problème SAT par la force brute (cf. figure 3) est de complexité exponentielle : on explore toutes les valuations possibles. Si la formule possède n variables, alors la complexité est en $\Theta(2^n)$. L'algorithme de Quine (cf. algorithme 1) permet d'éviter de tester des valuations qui ne sont pas solution.

J Algorithme de Quine

a Principe

L'algorithme de Quine (cf algorithme 1) explore l'ensemble des valeurs possibles pour chaque variable propositionnelle. Cette exploration se fait de manière arborescente [4].

La racine de l'arbre d'exploration est la formule. À chaque niveau de l'arbre, l'hypothèse est faite qu'une variable est vraie ou fausse et l'algorithme simplifie la formule en conséquence en remplaçant la variable par sa valuation. Des règles de simplifications permettent de propager

4. L'algorithme de Quine, c'est le retour sur trace ou backtracking appliqué à au problème SAT, cf. le cours de deuxième année, chapitre **??**).

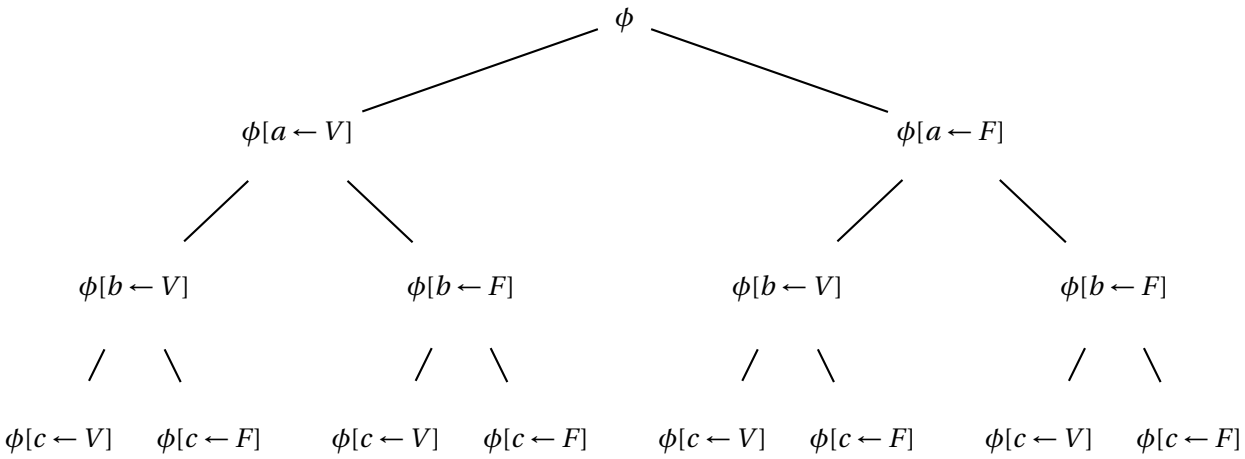

FIGURE 3 – Exemple d'arbre d'exploration de toutes les valuations possibles pour une formule logique simple $\phi = (a \wedge b) \vee c$. Selon la valuation des feuilles de l'arbre, on conclue sur la satisfaisabilité de la formule. Si au moins une feuille est évaluée V, alors la formule est satisfaisable.

la valeur et de conclure sur la possibilité de satisfaire la formule sous cette hypothèse ou non. Si c'est le cas, on étiquette la feuille de l'arbre avec un V et l'algorithme renvoie V.

Si la formule simplifiée est non satisfaisable, alors on ne poursuit pas l'exploration des solutions dans cette branche de l'arbre : on la marque F et on effectue un retour sur trace pour continuer l'exploration des autres branches.

Lorsque l'exploration est terminée, si les feuilles sont toutes marquées F, alors la formule n'est pas satisfaisable.

Algorithme 1 Algorithme Quine (SAT)

```
1 : Fonction QUINE_SAT(f)                                   ▷ f est une formule logique
2 :     SIMPLIFIER(f)
3 :     si f ≡ ⊤ alors
4 :         renvoyer Vrai
5 :     sinon si f ≡ ⊥ alors
6 :         renvoyer Faux
7 :     sinon
8 :         Choisir une variable x parmi les variables propositionnelles restantes de f
9 :         renvoyer QUINE(f[x ← Vrai]) ou QUINE(f[x ← Faux])
```

■ **Exemple 10 — Quine appliqué.** On considère la formule $\phi = (a \wedge b) \vee c$. La figure 4 détaille les étapes de l'algorithme de Quine sur l'arbre d'exploration. La branche de gauche a été raccourcie par les simplifications.

On en déduit la FND équivalente :

$$\phi \equiv c \vee (a \wedge b \wedge \bar{c}) \equiv c \vee (a \wedge b)$$

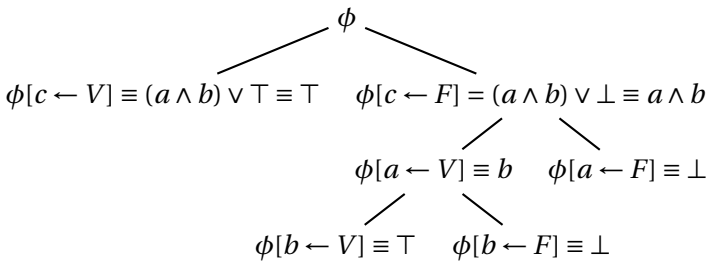

FIGURE 4 – Exemple d'arbre d'exploration structurant l'algorithme de Quine.

R Dans le pire des cas, l'algorithme de Quine nécessite d'explorer toutes les branches de l'arbre. Sa complexité est donc exponentielle en fonction du nombre de variables de la formule.

b Règles de simplification

Chaque nœud de l'arbre d'exploration est une formule créée en **remplaçant** une variable par $\top$ ou $\bot$. Il est alors possible de simplifier cette formule,c'est-à-dire de réduire sa taille, c'est-à-dire son nombre d'opérateurs, en observant les équivalences suivantes :

$$\neg\bot \equiv \top \quad \text{smart not} \tag{60}$$
$$\neg\top \equiv \bot \quad \text{smart not} \tag{61}$$
$$(\top \vee a) \equiv \top \quad \text{smart or} \tag{62}$$
$$(\bot \vee a) \equiv a \quad \text{smart or} \tag{63}$$
$$(\top \wedge a) \equiv a \quad \text{smart and} \tag{64}$$
$$(\bot \wedge a) \equiv \bot \quad \text{smart and} \tag{65}$$
$$(\top \Longrightarrow a) \equiv a \quad \text{smart imp} \tag{66}$$
$$(a \Longrightarrow \top) \equiv \top \quad \text{smart imp} \tag{67}$$
$$(\bot \Longrightarrow a) \equiv \top \quad \text{smart imp} \tag{68}$$
$$(a \Longrightarrow \bot) \equiv \neg a \quad \text{smart imp} \tag{69}$$
$$(\top \Longleftrightarrow a) \equiv a \quad \text{smart eq} \tag{70}$$
$$(\bot \Longleftrightarrow a) \equiv \neg a \quad \text{smart eq} \tag{71}$$

On nomme ces opérations de simplification des constructeurs intelligents (*smart constructors*). De plus, si la formule est simplifiée en $\top$ alors l'algorithme de Quine renvoie Vrai. Sinon, il continue l'exploration de l'arbre.

K Exemple de démonstration par induction structurelle

On se propose d'illustrer les démonstration de type induction structurelle sur l'exemple suivant : on se donne les formules logiques $\mathcal{F}$ définies inductivement comme en 15. On définit une transformation τ des formules logiques de la manière suivante :

- tous les opérateurs de type conjonction sont remplacés par des disjonctions et inversement, tous les opérateurs de type disjonctions sont remplacés par des conjonctions,
- les constantes ou les variables propositionnelles sont remplacées par leur négation.

On cherche à démontrer la propriété $\mathcal{P} : \forall \phi \in \mathcal{F}, \tau(\phi) \equiv \neg\phi$.

Démonstration. On procède par induction structurelle sur l'ensemble des formules logiques définis inductivement 15.

(Cas de base : $\bot$) Comme $\bot$ est une constante, d'après la définition de τ on a immédiatement $\tau(\bot) \equiv \top \equiv \neg\bot$.

(Cas de base : $\top$) de la même manière, $\tau(\top) \equiv \bot \equiv \neg\top$.

(Cas de base : $x \in \mathcal{V}$) de la même manière, $\forall x \in \mathcal{V}, \tau(x) \equiv \neg x$.

(Constructeur : not) Soit $\phi = \neg\phi_1$, ϕ_1 étant une formule vérifiant la propriété $\mathcal{P}$. Alors, $\tau(\phi_1) \equiv \neg\phi_1$. Comme la transformation τ ne modifie pas l'opérateur $\neg$, on peut écrire : $\tau(\phi) \equiv \tau(\neg\phi_1) \equiv \neg\tau(\phi_1) \equiv \neg(\neg\phi_1) \equiv \neg\phi$. La propriété $\mathcal{P}$ est donc vérifiée sur une formule construit par not.

(Constructeur : and) Soit ϕ_1 et ϕ_2 deux formules vérifiant la propriété $\mathcal{P}$. On construit alors $\tau(\phi_1 \wedge \phi_2) \equiv \tau(\phi_1) \vee \tau(\phi_2) \equiv \neg\phi_1 \vee \neg\phi_2 \equiv \neg(\phi_1 \wedge \phi_2)$. La propriété $\mathcal{P}$ est donc vérifiée sur une formule construit par and.

(Constructeur : or) Soit ϕ_1 et ϕ_2 deux formules vérifiant la propriété $\mathcal{P}$. On construit alors $\tau(\phi 1 \vee \phi_2) \equiv \tau(\phi_1) \wedge \tau(\phi_2) \equiv \neg\phi_1 \wedge \neg\phi_2 \equiv \neg(\phi_1 \vee \phi_2)$. La propriété $\mathcal{P}$ est donc vérifiée sur une formule construit par or

(Conclusion) La propriété $\mathcal{P}$ est vérifiée pour tous les éléments de bases de formules logiques. Par ailleurs, toute formule logique construite à l'aide d'un constructeur vérifie également la propriété. La propriété $\mathcal{P}$ est donc vérifiée pour toutes les formules logiques.

■