Introduction

Nous avons vu que les expressions arithmétiques, peuvent s'écrire sous la forme d'arbres. Dans ce chapitre, on étudie les expressions booléennes. On étudiera tout d'abord leur syntaxe, c'est à dire les constructeurs que l'on utilise pour les définir, et de manière plus importante leur sémantique. On parlera de **formules logiques** plutôt que d'expressions.

1 Syntaxe et sémantique

A Syntaxe

On se donne un ensemble infini $\mathcal{Q}$ de variables. On notera ses éléments $Q, P, Q', Q_1 \dots$.

Définition 1

On définit inductivement l'ensemble des formules propositionnelles $\mathcal{F}$, dont les éléments seront notés $\varphi, \psi, \varphi', \dots$, comme suit :

$$\varphi ::= Q \mid \top \mid \bot \mid \neg\psi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \to \varphi_2$$

On a :

- $\top$ s'appelle "top" ou "true", et $\bot$ s'appelle "bottom" ou "false" ;
- $\neg$ s'appelle "négation" ($\neg\varphi$ se lira "non φ") ;
- $\wedge$ s'appelle "conjonction" ($\varphi \wedge \psi$ se lira "φ et ψ") ;
- $\vee$ s'appelle "disjonction" ($\varphi \vee \psi$ se lira "φ ou ψ") ;
- $\to$ s'appelle "implication" ($\varphi \vee \psi$ se lira "φ implique ψ").

Exercice 1

Dessiner les arbres syntaxiques correspondant aux formules suivantes :

1. $Q \vee \neg Q$
2. $(P \wedge Q) \vee (\neg P \wedge \neg Q)$
3. $(P \wedge \top) \to (\neg P \vee Q)$
4. $(\bot \vee \neg Q) \wedge (P \wedge \neg Q)$

Remarque 1

En tant qu'arbre, on pourra parler de la hauteur et de la taille d'une formule.

Définition 2

Soit φ une formule propositionnelle. On définit inductivement l'ensemble des variables libres de φ, que l'on notera $\mathcal{V}(\varphi)$, comme suit :

— $\mathcal{V}(\top) = \mathcal{V}(\bot) = \emptyset$
— Pour $Q \in \mathcal{Q}$, $\mathcal{V}(Q) = \{Q\}$
— Pour $\varphi \in \mathcal{F}$, $\mathcal{V}(\neg\varphi) = \mathcal{V}(\varphi)$
— Pour $\varphi, \psi \in \mathcal{F}$, $\mathcal{V}(\varphi \wedge \psi) = \mathcal{V}(\varphi \vee \psi) = \mathcal{V}(\varphi \to \psi) = \mathcal{V}(\varphi) \cup \mathcal{V}(\psi)$

Par exemple, $\mathcal{V}(P \wedge (\neg Q \vee P)) = \{P, Q\}$.

B Algèbre des booléens

Les formules propositionnelles sont des objets **syntaxiques**, n'ayant pas de sens par elles-même. Par exemple, rien ne dit pour l'instant que "Vrai OU Faux" et "Vrai" sont égaux, comme ce serait le cas dans une condition booléenne en C ou en OCaml. L'algèbre des booléens va permettre d'introduire une sémantique simple.

Définition 3

L'**algèbre des booléens**, notée $\mathbb{B}$, est l'ensemble $\{0, 1\}$ muni des opérations $+, \times$ et $\bar{(\cdot)}$ définies comme suit :

- $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, $1 + 1 = 1$
- $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$, $1 \times 1 = 1$
- $\bar{0} = 1$, $\bar{1} = 0$

L'algèbre des booléens sert à représenter la sémantique des formules : 0 et 1 correspondent à faux et vrai, et les opérateurs $+, \times$ et $\bar{(\cdot)}$ correspondent au ET, OU et NON logique. On souligne à nouveau la différence entre syntaxe et sémantique : les formules $\top \wedge \bot$ et $\bot \wedge \top$ sont distinctes, mais $1 \times 0 = 0 \times 1 = 0$.

Proposition 1

0 est un élément neutre pour $+$ et absorbant pour $\times$. 1 est un élément absorbant pour $\times$ et neutre pour $+$.

C Table de vérité

On peut représenter une fonction $f : \mathbb{B}^n \to \mathbb{B}$ par sa **table de vérité**. Cette table contient 2^n lignes, une pour chaque élément de $\mathbb{B}^n$. Sur chaque ligne, on fait figurer la valeur de chaque entrée de la fonction, ainsi que la valeur de sortie pour cette entrée.

Exemple 1

On considère la fonction $f(x, y, z) = \bar{x} + (y \times x) + (\bar{z} \times y)$. Sa table de vérité est :

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

On remarque que c'est la même table de vérité que $g(x, y, z) = \bar{x} + y$

En pratique, on se forcera à énumérer les lignes des tables de vérité comme si on comptait en binaire.

Exercice 2

Q1. Calculer la table de vérité de la fonction $f(x,y,z) = ((x \times y) + (\bar{z} \times x)) \times (\bar{x} + \bar{y} + \bar{z})$, puis en proposer une expression plus simple.

Q2. Si $f(x_1, \ldots, x_n)$ est une fonction n-aire, expliquer comment obtenir la table de vérité de $g(x_1, \ldots, x_n) = \overline{f(x_1, \ldots, x_n)}$ à partir de celle de f.

Q3. Montrer que $+$ et $\times$ sont associatives, et qu'elles sont distributives l'une sur l'autre, en utilisant des tables de vérité.

D Sémantique

Nous pouvons maintenant définir la sémantique des formules propositionnelles :

Définition 4

Soit $\sigma : \mathcal{Q} \to \mathbb{B}$ une fonction partielle, appelée **valuation**. L'interprétation d'une formule $\varphi \in \mathcal{F}$ dans σ, notée $[\![\varphi]\!]^\sigma$, est définie si le domaine de σ contient $\mathcal{V}(\varphi)$, et se définit inductivement comme suit :

— $[\![\top]\!]^\sigma = 1$
— $[\![\bot]\!]^\sigma = 0$
— $[\![\varphi \wedge \psi]\!]^\sigma = [\![\varphi]\!]^\sigma \times [\![\psi]\!]^\sigma$
— $[\![\varphi \vee \psi]\!]^\sigma = [\![\varphi]\!]^\sigma + [\![\psi]\!]^\sigma$
— $[\![\neg\varphi]\!]^\sigma = \overline{[\![\varphi]\!]^\sigma}$
— $[\![\varphi \to \psi]\!]^\sigma = \overline{[\![\varphi]\!]^\sigma} + [\![\psi]\!]^\sigma$

Lorsque $[\![\varphi]\!]^\sigma = 1$, on dit que σ satisfait φ. On dit que φ est **satisfiable** s'il existe une valuation la satisfaisant. On dit que c'est une **tautologie** si toutes les valuations la satisfont.

Exercice 3

Q1. Pour la valuation $\sigma = [P \mapsto 1, Q \mapsto 0, R \mapsto 1]$, calculer l'interprétation des formules suivantes :

1. $Q \vee \neg Q$
2. $(P \wedge Q) \vee (\neg P \wedge \neg Q)$
3. $P \wedge \top$
4. $(\bot \vee \neg Q) \wedge (P \wedge \neg Q)$
5. $(P \wedge Q) \vee (R \wedge \neg Q)$
6. $P \wedge (Q \wedge R)$
7. $(P \wedge Q) \wedge R$

Q2. Pour chacune des formules précédentes, dire si elle est satisfiable, et si c'est une tautologie.

Ainsi, pour σ une valuation fixée, on peut regarder son effet sur les formules et considérer la fonction

$$[\![\cdot]\!]^\sigma : \mathcal{F} \to \mathbb{B}$$

Cependant, il est plus intéressant de fixer une formule φ et regarder son interprétation dans différentes valuation.

Remarque 2

L'interprétation d'une formule φ avec une valuation σ ne dépend que de la restriction de σ à $\mathcal{V}(\varphi)$.

Si l'on numérote les éléments de $\mathcal{V}(\varphi) = \{X_1, X_2, \ldots, X_n\}$, alors on peut voir φ comme une fonction de $\mathbb{B}^n$ dans $\mathbb{B}$:

$$[\![\varphi]\!]^{(\cdot)} : \begin{array}{ccc} \mathbb{B}^n & \longrightarrow & \mathbb{B} \\ (a_1, \ldots, a_n) & \longmapsto & [\![\varphi]\!]^{[X_1 \mapsto a_1, \ldots, X_n \mapsto a_n]} \end{array}$$

Exemple 2

La formule $X \wedge (\neg Y \vee Z)$ peut se voir comme la fonction $f : (x, y, z) \mapsto x \times (\bar{y} + z)$

Il faut à nouveau faire la différence entre syntaxe et sémantique. Au niveau syntaxique, les formules $X \wedge (Y \wedge \top)$ et $X \wedge Y$ sont distinctes, alors qu'au niveau sémantique, elles donnent lieu à la même fonction de $\mathbb{B}^2$ dans $\mathbb{B}$: $f(x, y) = x \times y$.

Considérons les formules $P \wedge (Q \wedge R)$ et $(P \wedge Q) \wedge R$. Formellement, les deux sont bien deux objets différents. Cependant, leur interprétation est la même, car l'opérateur $\times$ dans $\mathbb{B}$ est associatif. En pratique, on s'autorisera à écrire $P \wedge Q \wedge R$ sans parenthèses lorsque cela ne crée pas d'ambiguïté. Idem pour $P \vee Q \vee R$.

Définition 5

Si $(\varphi_i)_{i \in [\![1,n]\!]}$ est une famille finie de formules, on introduit les notations $\bigvee_{i=1}^n \varphi_i$ et $\bigwedge_{i=1}^n \varphi_i$ qui correspondent aux conjonctions / disjonctions des φ_i/ On les définit comme suit :

— $\bigvee_{i=1}^1 \varphi_i = \varphi_1$ et $\bigvee_{i=1}^n \varphi_i = (\bigvee_{i=1}^{n-1} \varphi_i) \vee \varphi_n$ pour $n \geq 2$

— $\bigwedge_{i=1}^1 \varphi_i = \varphi_1$ et $\bigwedge_{i=1}^n \varphi_i = (\bigwedge_{i=1}^{n-1} \varphi_i) \wedge \varphi_n$ pour $n \geq 2$

De manière générale, pour I ensemble fini, et $(\varphi_i)_{i \in I}$ une famille de formules indexée par I, on pourra considérer les formules $\bigvee_{i \in I} \varphi_i$ et $\bigwedge_{i \in I} \varphi_i$. Par convention, une conjonction vide est $\top$ et une disjonction vide est $\bot$ (car ce sont les éléments neutres).

Remarque 3

On définit la double-implication $\leftrightarrow$ comme suit :

— $\varphi \leftrightarrow \psi = (\varphi \to \psi) \wedge (\psi \to \varphi)$

Exercice 4

Dresser les tables de vérité de $\to$ et $\leftrightarrow$:

X	Y	$X \to Y$	$X \leftrightarrow Y$
0	0		
0	1		
1	0		
1	1		

On remarque que 0 implique 0 et 1. Cette propriété correspond au fait qu'en supposant une hypothèse fausse, on peut déduire n'importe quelle conclusion (ce que l'on appelle principe d'explosion).

Définition 6

On dit que deux formules φ, ψ sont logiquement équivalentes, ce que l'on notera $\varphi \equiv \psi$, si $[\![\varphi]\!]^\sigma = [\![\psi]\!]^\sigma$ pour toute valuation σ.

Proposition 2

$\equiv$ est une relation d'équivalence sur $\mathcal{F}$.

Proposition 3

Soient $\varphi, \psi, \varphi', \psi' \in \mathcal{F}$ telles que $\varphi \equiv \varphi'$ et $\psi \equiv \psi'$. Alors :

- $\varphi \wedge \psi \equiv \varphi' \wedge \psi'$
- $\varphi \vee \psi \equiv \varphi' \vee \psi'$
- $\varphi \to \psi \equiv \varphi' \to \psi'$
- $\overline{\varphi} \equiv \overline{\varphi}'$

Exercice 5

Q1. Pour chaque couple de formules (φ, ψ), déterminer si elles sont équivalentes, et si ce n'est pas le cas, donner une valuation permettant de les différencier.

- $\varphi = X \wedge Y, \psi = X \vee Y$
- $\varphi = X \vee (Y \wedge Z), \psi = (X \vee Y) \wedge (X \vee Z)$
- $\varphi = (\neg x \vee (Y \wedge Z)) \wedge ((X \wedge Y) \vee (X \wedge Z)), \psi = \bot$
- $\varphi = X \to (\neg X \wedge Y), \psi = \neg X$

Q2. Montrer le principe de contraposée :

$$\forall \varphi, \psi \in \mathcal{F}, \varphi \to \psi \equiv \neg\psi \to \neg\varphi$$

Proposition 4

Soient $\varphi, \psi \in \mathcal{F}$. Alors :

- $\varphi \to \psi \equiv \neg\varphi \vee \psi$
- $\neg\neg\varphi \equiv \varphi$
- $\neg(\varphi \wedge \psi) \equiv (\neg\varphi) \vee (\neg\psi)$
- $\neg(\varphi \vee \psi) \equiv (\neg\varphi) \wedge (\neg\psi)$

Les deux dernières identités sont les **lois de De Morgan**.

Démonstration. Exercice (*indication : passer par les tables de vérité).* □

On peut également s'intéresser à une relation plus faible que l'équivalence, appelée la conséquence logique :

Définition 7

Soient $\varphi, \psi \in \mathcal{F}$. On dit que φ est conséquence logique de ψ si toute valuation qui satisfait ψ satisfait aussi φ. On note $\psi \models \varphi$:

$$\psi \models \varphi \iff \forall \sigma : \mathcal{Q} \to \mathbb{B}, [\![\psi]\!]^\sigma = 1 \Rightarrow [\![\varphi]\!]^\sigma = 1$$

Cette définition signifie que si l'on suppose ψ vraie, alors φ est forcément vraie. L'année prochaine, vous étudierez un système de preuve appelé la déduction naturelle, qui permet de formaliser la notion de preuve mathématique. Dans ce système, on ne dira plus que les formules sont vraies mais qu'elles sont prouvables. Alors, la conséquence logique pourra se lire : "Si l'on suppose ψ, alors on peut prouver φ". ψ est donc une hypothèse et φ la conclusion.

Exemple 3

Si $X, Y \in \mathcal{Q}$, alors $X \wedge Y \models X$ car toute valuation qui satisfait $X \wedge Y$ satisfait en particulier X. Informellement, si je suppose $X \wedge Y$ alors je peux prouver X.

On peut étendre la notion de conséquence logique afin d'avoir plusieurs hypothèses :

Définition 8

Soit $\Gamma \subseteq \mathcal{F}$ un ensemble de formules, et $\varphi \in \mathcal{F}$. On dit que φ est conséquence logique de Γ si pour toute valuation σ qui satisfait chaque formule de Γ, σ satisfait aussi φ :

$$\Gamma \models \varphi \iff \forall \sigma : \mathcal{Q} \to \mathbb{B}, (\forall \psi \in \Gamma, [\![\psi]\!]^\sigma = 1) \Rightarrow [\![\varphi]\!]^\sigma = 1$$

On lira "φ est conséquence de Γ."

Exercice 6

Soient $X, Y, Z \in \mathcal{Q}$. Pour chacune des conséquences logiques suivantes, dire si elles sont vraies ou fausse. Si elles sont vraies, en faire une preuve. Sinon, donner une valuation constituant un contre-exemple adéquat.

- $X \models X \vee Y$
- $X \vee Y \models X$
- $X \to Y \models Y \to X$
- $X \to Y \models \neg Y \to \neg X$
- $X \to Y \to Z \models (X \wedge Y) \to Z$
- $X, X \to Y \models Y$
- $\neg X, X \models Y$
- $X \to Y, Y \to Z \models X \to Z$

Lorsqu'une formule φ est conséquence logique de l'ensemble vide, i.e. $\emptyset \models \varphi$, on notera simplement $\models \varphi$. Remarque : $\models \varphi$ signifie exactement que φ est une tautologie, i.e. que $\varphi \equiv \top$.

Exercice 7

L'effaceur fétiche des MP2I a été volé pendant la nuit. Les deux suspects potentiels sont Adrien et Béatrice. Après avoir enquêté, la MP2I a récolté les informations suivantes :

1. Adrien ou Béatrice a volé l'effaceur
2. Si Béatrice l'a volé, elle l'a fait avant minuit
3. Si le vol a eu lieu **après** minuit, alors Adrien est coupable
4. Le vol a eu lieu avant minuit.

Q1. On considère trois variables propositionnelles A, B, M correspondant respectivement à "Adrien est coupable", "Béatrice est coupable" et "le crime a eu lieu après minuit". Donnez 4 formules $\varphi_1, \dots \varphi_4$ correspondant aux informations récoltées par les enquêteurs.

Q2. On note $\Gamma = \{\varphi_1, \dots, \varphi_4\}$. Quel est le sens naturel de $\Gamma \models A$ et $\Gamma \models B$? A t'on l'une (ou les deux) de ces deux conséquences logiques ?

Q3. On rajoute l'information suivante : Si Adrien est coupable, alors le vol a eu lieu après minuit. Conclure.

D'un point de vue théorique, Γ peut être infini, mais en pratique de nombreux problèmes peuvent se modéliser avec Γ fini.

Proposition 5

Soit $\Gamma \subseteq \mathcal{F}$ un ensemble fini de formules et $\varphi, \psi \in \mathcal{F}$. Alors :

1. Si $\varphi \equiv \psi$ et $\Gamma \models \varphi$, alors $\Gamma \models \psi$.
2. $\Gamma \models \varphi$ si et seulement si $\bigwedge_{\psi \in \Gamma} \psi \models \varphi$.
3. $\psi \models \varphi$ si et seulement si $\models \psi \to \varphi$.

Démonstration. Les points 1 et 2 sont laissés en exercice. Pour le point 3, commençons par le sens direct. Supposons que $\psi \models \varphi$. Montrons qu'alors $\models \psi \to \varphi$. Par le point 1, il suffit de montrer que $\models \neg\psi \vee \varphi$.

Soit σ une valuation. Montrons que σ satisfait $\neg\psi \vee \varphi$. On a :

$$[\![\neg\psi \vee \varphi]\!]^\sigma = 1 \iff [\![\psi]\!]^\sigma = 0 \text{ ou } [\![\varphi]\!]^\sigma = 1$$

Faisons une disjonction de cas selon la valeur de $[\![\psi]\!]^\sigma$:

— Si $[\![\psi]\!]^\sigma = 0$, alors $[\![\neg\psi \vee \varphi]\!]^\sigma = 1$.

— Sinon, alors $[\![\psi]\!]^\sigma = 1$. Or comme $\psi \models \varphi$, et que σ satisfait ψ, σ satisfait aussi φ. Ainsi, $[\![\varphi]\!]^\sigma = 1$, et donc $[\![\neg\psi \vee \varphi]\!]^\sigma = 1$.

Le sens indirect est analogue.

□

Les trois points de la proposition précédente montrent que toute conséquence logique peut se mettre sous la forme $\models \phi$. Ainsi, lorsqu'un problème peut se modéliser par une conséquence logique, on peut même se ramener à une question de la forme "est ce que φ est une tautologie ?". Dans la partie suivante, on s'intéresse aux problèmes de la forme "est ce que φ est satisfiable".

2 Satisfiabilité

A Le problème SAT

Rappels Un problème de décision est une question à laquelle on peut répondre par oui ou non. Formellement, un problème de décision est un couple d'ensembles $(\mathcal{I}, P)$ avec $P \subseteq \mathcal{I}$: on dit que $\mathcal{I}$ est l'ensemble des instances du problème, et P l'ensemble des instances positives (celles pour qui la réponse est oui.

Un problème de décision fondamental en informatique est celui de la satisfiabilité des formules propositionnelles. On appelle ce problème **SAT** :

Définition 9

Le problème **SAT** est de savoir, étant donné une formule propositionnelle φ, s'il existe σ une valuation satisfaisant φ. Formellement, $\textbf{SAT} = (\mathcal{F}, S)$ avec $S \subseteq \mathcal{F}$ l'ensemble des formules satisfiable.

Les formules propositionnelles peuvent servir à encoder de nombreux problèmes, de telle sorte que résoudre **SAT** équivaut à résoudre le problème initial.

Exemple 4

Le club du frigo est un club très sélectif, dont les règles d'admissions sont comme suit :

1. Tout membre qui a une écharpe doit avoir un chapeau et des lunettes
2. Aucun membre ne peut avoir un chapeau et une écharpe
3. Si un membre n'a pas d'écharpe, il doit avoir des lunettes
4. Si un membre a un chapeau ou une écharpe, alors il a aussi des lunettes

On se demande si ce club admet des membres. Pour cela, on considère des variables propositionnelles C, E, L, dont le sens sera "avoir un chapeau", "avoir une écharpe" et "avoir des lunettes". Les règles du club se modélisent alors par les formules suivantes :

— $\varphi_1 = E \to (C \wedge L)$
— $\varphi_2 = \neg(C \wedge E)$
— $\varphi_3 = \neg E \to L$
— $\varphi_4 = (C \vee E) \to \neg L)$

Alors, le club admet au moins un membre si et seulement si la formule $\Phi = \bigwedge_{i=1}^{4} \varphi_i$ est satisfiable. De plus, une valuation satisfaisant Φ donne des conditions précises permettant d'intégrer le club.

Exercice 8

Tracer la table de vérité de Φ, et conclure sur la sélectivité du club du frigo.

Exemple 5

On considère une grille de Sudoku 4×4. On encode cette grille en une formule comme suit :

— Pour $(i, j) \in [\![1, 4]\!]$ et $k \in [\![1, 4]\!]$, on considère une variable propositionnelle $M_{i,j,k}$. On veut faire en sorte que cette variable corresponde à l'assertion "La case (i, j) contient le chiffre k.

— Pour chaque case (i, j) on considère la formule :

$$A_{i,j} = (\bigvee_{k=1}^{4} M_{i,j,k}) \wedge \neg \left[\bigvee_{1 \leq k \neq k' \leq 4} (M_{i,j,k} \wedge M_{i,j,k'}) \right]$$

Cette formule exprime qu'une case a au moins un chiffre écrit, mais pas plus.

— Pour chaque ligne $i \in [\![1, 4]\!]$ et chaque chiffre $k \in [\![1, 4]\!]$, on considère la formule :

$$L_{i,k} = \neg \bigvee_{1 \leq j \neq j' \leq 4} (M_{i,j,k} \wedge M_{i,j',k})$$

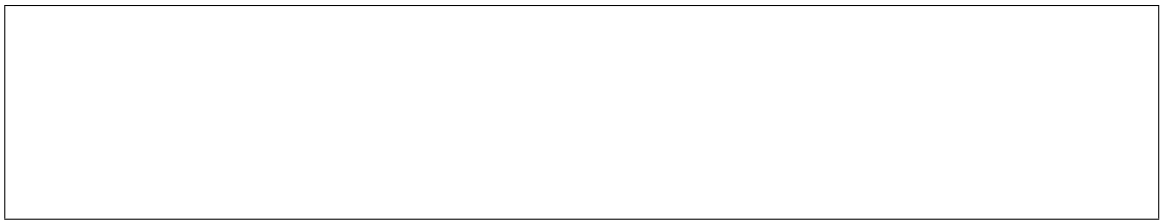
Cette formule exprime que sur la ligne i, la couleur k ne figure qu'une seule fois.

— De même, pour chaque colonne $j \in [\![1, 4]\!]$ et chaque chiffre $k \in [\![1, 4]\!]$, on considère la formule :

$$C_{j,k} = \neg \bigvee_{1 \leq i \neq i' \leq 4} (M_{i,j,k} \wedge M_{i',j,k})$$

Cette formule exprime que sur la colonne j, le chiffre k ne figure qu'une seule fois.

— Pour $i_0, j_0 \in [\![1, 2]\!]$ et $k \in [\![1, 4]\!]$, on considère une formule $S_{i_0,j_0,k}$ exprimant le fait qu'au sein du grand carré i_0, j_0, le chiffre k ne figure qu'une seule fois :



— Enfin, on numérote les cases déjà remplies $(i_1, j_1), (i_2, j_2), \ldots, (i_p, j_p)$ et on note $k_1, k_2, \ldots, k_p$ les chiffres inscrits. On considère la formule :

$$P = \bigwedge_{t=1}^{p} M_{i_t,j_t,k_t}$$

Cette formule exprime que les cases en questions contiennent un chiffre fixé à l'avance.

Enfin, on considère la grande formule composée de la conjonction de toutes les formules précédentes.

$$\Phi = \bigwedge_{i,j} A_{i,j} \wedge \bigwedge_{i,k} L_{i,k} \wedge \bigwedge_{j,k} C_{j,k} \wedge \bigwedge_{i_0,j_0,k} S_{i_0,j_0,k} \wedge P$$

Alors, Φ est satisfiable si et seulement la grille peut être résolue. De plus, une valuation satisfaisant Φ donne directement une solution de la grille, et inversement. En effet :

— Si la grille peut être résolue, on note $G = (G[i,j])_{1\leq i,j\leq 4}$ la grille complétée. On considère alors la valuation σ_G suivante :

$$\forall i,j,k \in [\![1,4]\!], \sigma(M_{i,j,k}) = \begin{cases} 1 & \text{si } G[i,j] = k \\ 0 & \text{sinon} \end{cases}$$

Remarquons qu'alors, les variables propositionnelles de la forme $M_{i,j,G[i,j]}$ sont mises à 1 par σ_G, et toutes les autres à 0.

Montrons que σ_G satisfait Φ. Il suffit de montrer que σ_G satisfait chacune des formules $A_{i,j}, L_{i,k}, C_{j,k}, S_{i_0,j_0,k}$ et P. Chacune se vérifie facilement car elles découlent des règles du Sudoku. Par exemple :

— Pour $i,j \in [\![1,4]\!]$, $A_{i,j}$ est satisfaite car $\sigma(M_{i,j,G[i,j]}) = 1$ et $\sigma(M_{i,j,k}) = 0$ pour les autres valeurs de k. Donc $[\![\bigvee_{k=1}^{4} M_{i,j,k}]\!]^{\sigma_G} = 1$ et $[\![\bigvee_{1\leq k\neq k'\leq 4}(M_{i,j,k} \wedge M_{i,j,k'})]\!]^{\sigma_G} = 0$, et ainsi $[\![A_{i,j}]\!]^{\sigma_G} = 1$.

— P est satisfaite car si la grille G est une solution de la grille initiale, alors en particulier les cases qui étaient déjà présentes n'ont pas changé. Autrement dit, en reprenant les notations utilisées pour définir P, pour $t \in [\![1,p]\!]$, $G[i_t,j_t] = k_t$, et donc $\sigma(M_{i_t,j_t,k_t}) = 1$, et donc $[\![P]\!]^{\sigma_G} = 1$.

— A l'inverse, si σ est une valuation satisfaisant Φ, montrons que l'on peut en extraire une solution de la grille. Tout d'abord, remarquons que σ satisfait chacune des $A_{i,j}$. Or, cette formule est construite précisément de telle sorte que cela implique qu'il existe un unique $k_{i,j}$ tel que $M_{i,j,k} = 1$. On considère alors la grille $G = (G[i,j])_{1\leq i,j\leq 4}$ avec $G[i,j] = k_{i,j}$. Il reste à vérifier que cette grille est une solution de la grille initiale, en vérifiant que chaque nombre apparaît une seule fois dans chaque ligne, colonne, et carré, et que les cases pré-remplies n'ont pas changé. Pour cela, on repasse par les sous-formules de Φ, qui expriment précisément ces contraintes. Par exemple :

— Vérifions que chaque ligne i contient une seule fois chaque nombre. Pour $k \in [\![1,4]\!]$, la formule $L_{i,k}$ exprime qu'il n'existe aucun couple de colonnes $j \neq j'$ tels que $\sigma(M_{i,j,k}) = 1$ et $\sigma(M_{i,j',k}) = 1$, i.e. tels que $G[i,j] = G[i,j'] = 1$. Autrement dit, le chiffre k n'apparaît pas deux fois sur la ligne.

Le problème **SAT** est un problème fondamental en informatique : on ne connait pas à ce jour d'algorithme polynomial pour le résoudre, et s'il en existe un, alors $\mathbf{P} = \mathbf{NP}$ (ce qui résoudrait l'un des (si ce n'est le) plus célèbres problèmes en informatique théorique). **P** est la classe des problèmes résoluble en temps polynomial, et **NP**, que vous étudierez l'année prochaine, est la classe des problèmes dont on peut vérifier une solution en temps polynomial.

B Forme normale

Étant donne une formule φ, il existe une infinité de formules équivalentes. Les formes normales vont donner des formes canoniques pour les formules.

Définition 10

Soit $\varphi \in \mathcal{F}$. Alors :

- Si φ est de la forme Q ou $\neg Q$, avec $Q \in \mathcal{Q}$, on dit que φ est un **littéral**
- Si φ est de la forme $L_1 \vee L_2 \vee \cdots \vee L_n$ avec $L_1, \ldots, L_n$ des littéraux, on dit que c'est une **clause disjonctive** de taille n
- Si φ est de la forme $L_1 \wedge L_2 \wedge \cdots \wedge L_n$ avec $L_1, \ldots, L_n$ des littéraux, on dit que c'est une **clause conjonctive** de taille n
- Si φ est de la forme $C_1 \wedge \cdots \wedge C_p$ avec $C_1, \ldots, C_p$ des clauses disjonctive, on dit que φ est sous **forme normale conjonctive** (FNC)
- Si φ est de la forme $C_1 \vee \cdots \vee C_p$ avec $C_1, \ldots, C_p$ des clauses conjonctive, on dit que φ est sous **forme normale disjonctive** (FND)

Une formule sous FNC est donc une conjonction de disjonctions de littéraux.

Exemple 6

Quelques exemples sur ces notions :

- $X \vee Y \vee Z \vee \neg T$ est une clause disjonctive
- $(X \vee Y \vee \neg T) \wedge (Y \vee \neg Y \vee \neg Z) \wedge (Y \vee Z \vee T)$ est une formule sous FNC.
- $X \wedge (Y \vee (T \wedge Z) \vee (\neg Y \wedge Z))$ n'est pas sous FNC

Remarque 4

Soit $\varphi = \bigwedge_{i=1}^{p} \bigvee_{j=1}^{m_i} L_{ij}$ une formule sous forme normale **conjonctive**. Alors une valuation σ satisfait φ si et seulement pour tout $i \in [\![1, p]\!]$, il existe $j \in [\![1, m_i]\!]$ tel que σ satisfait L_{ij}.

Exercice 9

Proposer une remarque similaire pour les formules sous FND.

Proposition 6

Toute formule est équivalente à une formule sous FNC, et à une formule sous FND.

Démonstration. Soit φ une formule. Commençons par trouver une FND pour φ. Notons $V = \{X_1, \ldots, X_n\}$ l'ensemble fini des variables apparaissant dans φ. On trace la table de vérité de φ sur V. On obtient alors 2^n lignes :

X_1	$\ldots$	X_{n-1}	X_n	$[\![\varphi]\!]$
0	$\ldots$	0	0	s_0
0	$\ldots$	0	1	s_1
0	$\ldots$	1	0	s_2
0	$\ldots$	1	1	s_3
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
1	$\ldots$	1	1	s_{2^n-1}

avec $s_i = 0$ ou 1 pour $i \in [\![0, 2^n - 1]\!]$.

On note $K \subseteq [\![0, 2^n - 1]\!]$ l'ensemble des k telle que $S_k = 1$. Pour $k \in K$, on note C_k la clause constitué des conjonction des X_i valués à 1 sur la ligne k et des $\neg X_i$ pour X_i valué à 0 sur la ligne k. Par exemple, si la ligne k contient $1, 0, 1, 1$ pour X_1, X_2, X_3, X_4, alors la clause correspondante sera $X_1 \wedge \neg X_2 \wedge X_3 \wedge X_4$.

Alors (la justification formelle est en exercice dans le TD) :

$$\varphi \equiv \bigvee_{k \in K} C_k$$

Pour déterminer une FNC de φ, on détermine une FND de $\neg\varphi$ de la forme $\bigvee_{i=1}^{p} C_i$ avec les C_i clauses conjonctives. Donc, φ est équivalente logiquement à $\neg \bigvee_{i=1}^{p} C_i$. En appliquant les lois de De Morgan, cette formule est équivalente à $\bigwedge_{i=1}^{p} \neg C_i$. En appliquant à nouveau les lois de De Morgan sur les C_i, qui sont des conjonctions, on transforme chaque $\neg C_i$ en une clause disjonctive D_i, et donc φ est équivalente à $\bigwedge_{i=1}^{p} D_i$. □

Exercice 10

Q1. Pour $\varphi = (X \to (Y \wedge \neg Z)) \wedge (\neg(Y \wedge Z) \vee X)$, déterminer une FND et une FNC de φ.

Pour $n \in \mathbb{N}$, on considère la formule φ_n suivante sur les variables $X_1^1, X_1^2, X_2^1, X_2^2, \ldots, X_n^1, X_n^2$:

$$\varphi_n = \bigwedge_{k=1}^{n} (X_k^1 \vee X_k^2)$$

Q2. φ_n est-elle sous une forme normale ? Quelle est la taille de φ_n en fonction de n ?

Q3. Déterminer la formule sous FNC obtenue à l'aide de la table de vérité de φ_n. Quelle est sa taille ?

On voit donc que le procédé décrit pour calculer une FND et une FNC à partir d'une formule ne donne pas nécessairement une formule de taille minimale. Au contraire, elle peut rallonger la formule initiale.

Définition 11

On définit les problèmes $\mathbf{FNC-SAT}$ et $\mathbf{FND-SAT}$ comme suit :

- $\mathbf{FNC-SAT}$: étant donné φ une variable sous FNC, φ est-elle satisfiable ?
- $\mathbf{FND-SAT}$: étant donné φ une variable sous FND, φ est-elle satisfiable ?
- $3-\mathbf{SAT}$: étant donné φ une variable sous FNC donc chaque clause contient au plus trois littéraux, φ est-elle satisfiable ?

$\mathbf{FNC-SAT}$ et $3-\mathbf{SAT}$ sont aussi durs que $\mathbf{SAT}$: ils font partie de la classe de problèmes appelée $\mathbf{NP-complets}$, pour lesquels on ne connaît aucun algorithme en temps polynomial. En revanche, $\mathbf{FND-SAT}$ est très simple à résoudre.

Exercice 11

Q1. Montrer que pour φ et ψ deux formules quelconques, $\varphi \vee \psi$ est satisfiable si et seulement si φ est satisfiable ou ψ est satisfiable.

Q2. Est-ce vrai pour $\varphi \wedge \psi$?

Q3. Déterminer un algorithme polynomial permettant de déterminer si une formule sous FND est satisfiable. Cet algorithme prendra en entrée une liste de listes de

littéraux.

Q4. Rillaume Gousseau, jeune chercheur en informatique, pense avoir résolu **P** = **NP**. En effet, il a trouvé un algorithme polynomial pour déterminer si une formule quelconque est satisfiable :

Algorithme 1 : Satisfiabilité

Entrée(s) : φ une formule propositionnelle
Sortie(s) : φ est-elle satisfiable ?
1 $\psi \leftarrow$ forme normale disjonctive de φ ;
2 Résoudre $\mathbf{FND-SAT}$ sur ψ en temps polynomial ;
3 **retourner** le résultat obtenu pour $\mathbf{FND-SAT}$

Où se situe son erreur ?

Exercice 12

Modélisons un petit jeu sous la forme d'une formule sous FNC. On considère le jeu du solitaire, dont le principe est le suivant : On se donne un damier de $m \times m$ cases, dont chacune contient soit une pierre bleue soit une pierre rouge, soit rien. Le but est d'enlever des pierres du damier, afin d'obtenir une situation où :

— Sur chaque ligne, il y a au moins une pierre
— Sur chaque colonne, toutes les pierres sont de la même couleur.

Formellement, on considère le problème de décision suivant, appelé **SOLITAIRE** :

— Entrée : (m, B_0, R_0) avec $m \in \mathbb{N}^*$, $B_0 \subseteq [\![1,m]\!]^2$, $R_0 \subseteq [\![1,m]\!]^2$ et $R_0 \cap B_0 = \emptyset$
— Sortie : Vrai s'il existe $B \subseteq [\![1,m]\!]^2$ et $R \subseteq [\![1,m]\!]^2$ tels que :
  1. $B \subseteq B_0$
  2. $R \subseteq R_0$
  3. $\forall i \in [\![1,m]\!], \{i\} \times [\![1,m]\!] \cap (B \cup R) \neq \emptyset$
  4. $\forall j \in [\![1,m]\!], [\![1,m]\!] \times \{i\} \cap B = \emptyset$ ou $[\![1,m]\!] \times \{i\} \cap R = \emptyset$

On souhaite trouver une procédure qui, étant donnée (m, B_0, R_0) un instance de **SOLITAIRE** calcule une formule propositionnelle Φ_{m,B_0,R_0} telle que (m, B_0, R_0) est une instance positive de **SOLITAIRE** si et seulement si Φ_{m,B_0,R_0} est satisfiable. Par ailleurs, on souhaite que la formule construite soit sous FNC.

On utilisera les variables propositionnelles $R_{i,j}$ et $B_{i,j}$ (pour $(i,j) \in [\![1,m]\!]^2$) qui expriment respectivement que $(i,j) \in B$ et que $(i,j) \in R$, autrement dit qui expriment la couleur de la pierre que contient la case (i,j).

Q1. (entraînement) Écrire une formule signifiant "La case $(2,3)$ ne contient ni pierre bleue ni pierre rouge".

Q2. Déterminer des formules ϕ^1_{m,B_0,R_0}, ϕ^2_{m,B_0,R_0}, ϕ^3_{m,B_0,R_0} et ϕ^4_{m,B_0,R_0} exprimant les contraintes 1 à 4. Les mettre sous FNC.

Q3. Exprimer Φ_{m,B_0,R_0}.

Montrons maintenant que Φ_{m,B_0,R_0} est satisfiable si et seulement si (m, B_0, R_0) est une instance positive.

Q4. On suppose que (m, B_0, R_0) est une instance positive. On peut donc considérer B et R qui satisfont les conditions 1 à 4. Donner une valuation σ qui satisfait Φ_{m,B_0,R_0}.

Q5. On suppose maintenant que Φ_{m,B_0,R_0} est satisfiable. Soit σ satisfaisant Φ_{m,B_0,R_0}. Décrire des ensembles B et R permettant de conclure que (m, B_0, R_0) est une instance positive de **SOLITAIRE** (et montrer que B et R sont bien définis et vérifient bien les conditions 1 à 4).

Un point important de la formule $\Phi_{m,B,R}$ est qu'elle peut être construite en temps polynomial en la taille de l'entrée. Vous verrez l'année prochaine que cette propriété est fondamentale dans l'étude des classes de complexité.

Proposition 7

2 – SAT est résoluble en temps polynomial.

Démonstration. L'année prochaine ! □

C Résolution de SAT

Notons tout d'abord qu'il existe un algorithme très simple pour résoudre **SAT** : étant donné une formule φ sur n variables libres, il suffit de tester les 2^n valuations possibles :

Algorithme 2 : SAT-solver simple

Entrée(s) : φ une formule propositionnelle
Sortie(s) : "Oui" si φ est satisfiable, "Non" sinon

```
1 Noter X_1, ..., X_n les variables de φ ;
2 pour toute valuation σ sur X_1, ..., X_n faire
3     si σ satisfait φ alors
4         retourner "Oui"
5 retourner "Non"
```

Cet algorithme s'exécute dans le pire cas en $\mathcal{O}(m2^n)$ avec m la taille de la formule et n le nombre de variables libres, car il y a 2^n valuations à tester, et calculer l'interprétation de φ dans une valuation σ se fait en temps linéaire en la taille de φ, i.e. en $\mathcal{O}(m)$.
On ne connaît pas à l'heure actuelle d'algorithme polynomial pour résoudre **SAT** [1]. Cependant, il existe de nombreuses optimisations et heuristiques permettant d'accélérer l'algorithme naïf. Nous étudions ici une de ces méthodes, appelée l'algorithme de Quine. Cet algorithme fonctionne selon le principe de **retour sur trace**, ou **backtracking** et consiste à construire la valuation recherchée variable par variable.

1. L'existence ou non d'un tel algorithme fait même l'objet d'un des problèmes du prix du millénaire !

Définition 12

Soit $\varphi, \psi \in \mathcal{F}$ des formules propositionnelles, et $Q \in \mathcal{Q}$ une variable propositionnelle. La substitution de Q par ψ dans φ, notée $\varphi[\psi/Q]$, est la formule φ dans laquelle chaque occurrence de Q a été remplacée par ψ. On peut définir une telle substitution par induction comme suit :

- $\top[\psi/Q] = \top$
- $\bot[\psi/Q] = \bot$
- $Q[\psi/Q] = \psi$
- $P[\psi/Q] = P$ si $P \neq Q$
- $(\varphi_1 \wedge \varphi_2)[\psi/Q] = \varphi_1[\psi/Q] \wedge \varphi_2[\psi/Q]$
- $(\varphi_1 \vee \varphi_2)[\psi/Q] = \varphi_1[\psi/Q] \vee \varphi_2[\psi/Q]$
- $(\neg\varphi)[\psi/Q] = \neg(\varphi[\psi/Q])$

On lira $\varphi[\psi/Q]$ "φ dans laquelle ψ remplace Q".

Le principe de l'algorithme de Quine est de tester, comme dans la méthode naïve, les 2^n possibilités (où n est le nombre de variables distinctes), mais de couper immédiatement les cas qui ne mèneront nulle part en appliquant des simplifications aux formules. On choisit donc une variable X aléatoire, on essaye de substituer X par $\top$, on simplifie puis on teste récursivement la satisfiabilité, et si l'on n'a pas trouvé de valuation, on recommence en substituant X par $\bot$.

Pour simplifier une formule, on la remplace par une formule logiquement équivalente selon les règles suivantes :

1. $\top \wedge \varphi \equiv \varphi \wedge \top \equiv \varphi$
2. $\bot \wedge \varphi \equiv \varphi \wedge \bot \equiv \bot$
3. $\top \vee \varphi \equiv \varphi \vee \top \equiv \top$
4. $\bot \vee \varphi \equiv \varphi \vee \bot \equiv \varphi$
5. $\neg\neg\varphi \equiv \varphi$
6. $\neg\top \equiv \bot$
7. $\neg\bot \equiv \top$

Plus précisément, on applique ces règles jusqu'à obtenir un point fixe.

Exercice 13

On note $\varphi_1 = (X \wedge \neg Z \wedge (\neg Y \vee Z) \vee (\neg X \wedge Y \wedge \neg Z)$.

Q1. Simplifier $\varphi_1[\top/Z]$.

Q2. Simplifier $\varphi_1[\bot/Z]$. On notera φ_2 la formule obtenue.

Q3. Simplifier $\varphi_2[\bot/X]$. On notera φ_3 la formule obtenue

Q4. Simplifier $\varphi_3[\top/Y]$.

Décrivons maintenant l'algorithme de Quine :

Algorithme 3 : Algorithme de Quine

Entrée(s) : φ une formule propositionnelle
Sortie(s) : Une valuation σ satisfaisant φ, ou IMPOSSIBLE si ce n'est pas possible

si $\varphi = \top$ **alors**
  retourner $\emptyset$ ***la valuation vide***
si $\varphi = \bot$ **alors**
  retourner ***IMPOSSIBLE***
$X \leftarrow$ une variable quelconque de φ ;
$\varphi_\top \leftarrow \varphi[\top/X]$;
Simplifier $\varphi_\top$;
si $\varphi_\top$ ***est satisfiable par une valuation*** σ **alors**
  retourner $\sigma' = \sigma[X \mapsto 1]$
sinon
  $\varphi_\bot \leftarrow \varphi[\bot/X]$;
  Simplifier $\varphi_\bot$;
  si $\varphi_\bot$ ***est satisfiable par une valuation*** σ **alors**
    retourner $\sigma' = \sigma[X \mapsto 0]$
  sinon
    retourner ***IMPOSSIBLE***

Appliquons l'algorithme sur quelques formules pour voir comment il marche. On représentera l'exécution de l'algorithme par un arbre : chaque noeud correspondra à un appel, et les enfants d'un noeud interne correspondront aux appels récursifs de l'algorithme, c'est à dire aux formules obtenues en remplaçant une variable par $\top$ puis par $\bot$.

1. $\varphi = ((X \to Y) \wedge (Y \to Z)) \to (X \to Z)$
2. $\varphi = (((X \vee Y) \wedge (Y \to \neg Z)) \vee (\neg X \wedge \neg Z)) \wedge Z$

D Représentation de formules en code

En OCaml, pour représenter des formules propositionnelles quelconques, on pourra utiliser le type suivant :

```
type formula =
  | Top | Bot
  | And of formula * formula
  | Or of formula * formula
  | Var of string
```

Il faut ensuite déterminer une manière de représenter les valuations. La structure adaptée est celle de dictionnaire : à chaque variable on associe une valeur.

```
type valuation = ...

(* get sigma x renvoie la valeur de x dans sigma *)
let get (sigma: valuation) (x: string) : bool = ...

(* renvoie la valuation sigma mise àjour avec l'association [x -> b] *)
let set (sigma: valuation) (x: string) (b: bool) : valuation = ...
```

On peut par exemple stocker les couples (nom de variable, valeur assignée) dans une `(string*bool)list` :

```
type valuation = (string * bool) list

(* cherche un couple de la forme (v, b) dans l, et renvoie b.
   Lève une exception si aucun tel couple n'est trouvé. *)
let rec get (l: valuation) (v:string) : bool =
  match l with
  | (x, b)::q when x = v -> b
  | _::q -> get q v
  | [] -> failwith "variable non présente"
```

Remarque : en OCaml, la fonction `List.assoc: ('a -> ('a * 'b)list -> 'b)` réalise exactement cette opération (mais l'ordre des opérandes est inversé).

On pourrait aussi utiliser des ABR, ou des hashtables, ce qui donnerait de meilleures complexités. Voyons maintenant la fonction permettant d'interpréter une formule dans une valuation :

```
let rec interpreter (f: formula) (sigma: valuation) : bool =
  match f with
  | Top -> true
  | Var v -> get sigma v
  | And(f1, f2) -> interpreter f1 sigma && interpreter f2 sigma
  ...
```

E Spécialisation aux formules sous FNC

Voyons comment adapter l'algorithme de Quine pour les formules sous FNC. On peut représenter une formule sous FNC comme une liste de listes de littéraux. [2]

```
type litteral =
  | Var of string
  | NotVar of string

type clause = litteral list
type fnc = clause list
```

Sur ce type, intéressons nous à l'interprétation d'une formule. Si σ est une valuation, et $\varphi = \bigwedge_{i=1}^{p} \bigvee_{j=1}^{k_i} l_{i,j}$ est une formule sous FNC, alors σ satisfait φ si et seulement si σ satisfait chaque clause, si et seulement si pour tout $i \in [\![1, p]\!]$, il existe $j \in [\![1, k_i]\!]$ tel que $[\![l_{i,j}]\!]^\sigma = 1$.

On rappelle l'existence des fonctions `List.for_all` et `List.exists` :

```
List.forall : ('a -> bool) -> 'a list -> bool
  forall p l renvoie true si pour tout élément x de l, p x = true
  et false sinon

List.exists: ('a -> bool) -> 'a list -> bool
  exists p l renvoie true si il existe un élément x de l tel p x = true
  et false sinon
```

Ces fonctions permettent d'implémenter simplement la fonction d'interprétation d'une formule sous FNC :

```
(* interprète le littéral l dans la valuation sigma *)
let interp_litt (sigma: valuation) (l:litteral) : bool = match l with
  | Var v -> get sigma v
  | NotVar v -> not (get sigma v)

```

2. En réalité, l'ordre des clauses et des littéraux n'ayant pas d'importance, on pourrait ici aussi utiliser des ABR, ou des hashtables...

```
(* interprète la clause c dans la valuation sigma. *)
let interp_clause (sigma: valuation) (c: clause) : bool =
  (* vérifie si au moins un des littéraux de c s'interprète àvrai *)
  List.exists (interp_litt sigma) c

(* interprète la formule f sous FNC dans sigma. *)
let interp_fnc (sigma: valuation) (f: fnc) : bool =
  (* vérifie si chaque clause de f s'interprète àvrai *)
  List.for_all (interp_clause sigma) f
```

Regardons maintenant l'algorithme de Quine spécialisé aux formules sous FNC. Si $\varphi = \bigwedge_{i=1}^{p} \bigvee_{j=1}^{k_i} l_{i,j}$ est une formule sous FNC, et X une variable apparaissant dans φ, alors pour obtenir une formule ψ sous FNC équivalente à $\varphi[\top/X]$, il suffit de supprimer les clauses qui contiennent le littéral X, et de supprimer de chaque clause restante le littéral $\neg X$.

Exemple 7

On considère $\varphi = (X \vee \neg Y \vee Z) \wedge (\neg X \vee Z) \wedge (X \vee \neg Z)$. Alors, si l'on remplace X par $\top$ dans φ, on obtient la formule $(\top \vee \neg Y \vee Z) \wedge (\neg \top \vee Z) \wedge (\top \vee \neg Z)$. Mais on sait que $\top \vee \theta \equiv \top$ pour toute formule θ et que $\bot \vee \theta \equiv \theta$ pour toute formule θ. Donc, cette formule est équivalente à Z.

L'algorithme de Quine peut donc s'exprimer ainsi sur les FNC :

Algorithme 4 : `Quine`(F)

Entrée(s) : F une conjonction de clauses $C_1, \dots, C_p$, avec chaque C_i une liste de littéraux $l_{i_1}, \dots, l_{i,k_i}$

Sortie(s) : σ une valuation satisfaisant F, IMPOSSIBLE si une telle valuation n'existe pas

si F ***est vide*** **alors**
  retourner ***valuation vide*** `// une conjonction vide est équivalente à` $\top$
si F ***contient une clause vide*** **alors**
  retourner ***IMPOSSIBLE*** `// une disjonction vide est équivalente à` $\bot$
$X \leftarrow$ une variable quelconque de F;
pour $i = 1$ **à** ***longueur***(F) **faire**
  si C_i ***contient*** X **alors**
    Retirer C_i de F;
  sinon
    si C_i ***contient*** $\neg X$ **alors**
      Retirer $\neg X$ de C_i;
$\sigma \leftarrow$ `Quine`(F);
si σ ***n'est pas IMPOSSIBLE*** **alors**
  Mettre à jour σ avec $X \mapsto 1$;
  retourner σ
sinon
  Répéter tout le procédé en échangeant les rôles de X et $\neg X$.;
  Renvoyer **IMPOSSIBLE** si cela échoue aussi;

En pratique, lorsque l'on modélise un problème par une formule logique, on la met sous FNC, car les algorithmes permettant de résoudre SAT (dont celui de Quine) sont plus efficaces lorsqu'ils travaillent sur des FNC que sur des formules sans structure particulière.