Chapitre 0

Logique

Hugo Salou MPI*

Dernière mise à jour le 31 mars 2023

Table des matières

0 Motivation

Considérons la grille de Sudoku 2×2 ci-dessous.

3			2
	4	1	
	3	2	
4			1

Figure 1 – Grille de Sudoku 2×2

On modélise ce problème : on considère $P_{i,j,k}$ une variable booléenne, c'est à dire un élément de $\{\boldsymbol{V}, \boldsymbol{F}\}$, définie telle que

$$P_{i,j,k} : \text{“}m(i,j) \stackrel{?}{=} k\text{” avec } (i,j,k) \in [\![1,4]\!]^3 \text{ ..}$$

On peut définir des contraintes logiques (des expressions logiques) pour résoudre le Sudoku. Les opérateurs ci-dessous seront définis plus tard.

$$\begin{aligned} & P_{113} \\ \wedge\ & P_{1,4,2} \\ \wedge\ & P_{2,2,4} \\ \wedge\ & P_{2,3,1} \\ & \vdots \\ \wedge\ & P_{1,2,1} \to (\neg P_{1,2,2} \wedge \neg P_{1,2,3} \wedge \neg P_{1,2,4}) \\ & \vdots \end{aligned}$$

Pour résoudre le Sudoku, on peut essayer chaque cas possible. Mais, ces possibilités sont très nombreuses.

En mathématiques, on utilise une certaine logique. Il en existe d'autre, certaines où tout est vrai, certaines où il est plus facile de montrer des théorèmes, etc. On va définir une logique ayant le moins d'opérateurs possibles.

1 Syntaxe

Définition : On suppose donné un ensemble $\mathcal{P}$ de variables propositionnelles.

Définition : On définit alors l'ensemble des formules de la logique propositionnelle par induction nommée avec les règles :

- $\neg\big|^1$;
- $\wedge\big|^2$;
- $\vee\big|^2$;
- $\to\big|^2$;
- $\leftrightarrow\big|^2$;
- $\top\big|^0$;
- $\bot\big|^0$;
- $V\big|^0_{\mathcal{P}}$.

On nomme l'ensemble des formules $\mathcal{F}$.

Définition (taille d'une formule) **:** On définit, par induction, la taille notée "taille" comme

$$\begin{aligned}
\text{taille} : \mathscr{F} &\longrightarrow \mathbb{N} \\
p \in \mathscr{P} &\longmapsto 1 \\
\top &\longmapsto 1 \\
\bot &\longmapsto 1 \\
\neg G &\longmapsto 1 + \text{taille}(G) \\
G \to H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\
G \leftrightarrow H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\
G \wedge H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H) \\
G \vee H &\longmapsto 1 + \text{taille}(G) + \text{taille}(H).
\end{aligned}$$

Définition (Ensemble des variables propositionnelles) **:** On définit inductivement

$$\begin{aligned}
\text{vars} : \mathscr{F} &\longrightarrow \wp(\mathscr{P}) \quad {}^{1} \\
p \in \mathscr{P} &\longmapsto \{p\} \\
\top, \bot &\longmapsto \varnothing \\
\neg G &\longmapsto \text{vars}(G) \\
G \odot H &\longmapsto \text{vars}(G) \cup \text{vars}(H)
\end{aligned}$$

où $\odot$ correspond à $\cup$, $\cap$, $\to$ ou $\leftrightarrow$.

Définition : On appelle *substitution* une fonction de $\mathscr{P}$ dans $\mathscr{F}$ qui est l'identité partout sauf sur un ensemble fini de variables. On la note alors

$$(p_1 \mapsto H_1,\, p_2 \mapsto H_2, \ldots, p_n \mapsto H_n)$$

qui est la substitution

$$\begin{aligned}
\mathscr{P} &\longrightarrow \mathscr{F} \\
p &\longmapsto \begin{cases} H_i & \text{si } p = p_i \\ p & \text{sinon.} \end{cases}
\end{aligned}$$

Définition (Application d'une substitution à une formule) **:** Étant donné une formule $G \in \mathscr{F}$ et une substitution σ, on définit inductivement $G[\sigma]$ par

$$\begin{cases}
\top[\sigma] = \top \\
\bot[\sigma] = \bot \\
p[\sigma] = \sigma(p) \\
(\neg G)[\sigma] = \neg(G[\sigma]) \\
(G \odot H)[\sigma] = (G[\sigma]) \odot H[\sigma]
\end{cases}$$

où $\odot$ correspond à $\cup$, $\cap$, $\to$ ou $\leftrightarrow$.

1. Le $\wp(E)$ représente ici l'ensemble des parties de E.

Définition : On appelle parfois *clés* d'une substitution de σ, l'ensemble des variables propositionnelles sur lequel elle n'est pas l'identité.

Définition : On définit la *composée* de deux substitutions σ et σ' par

$$\begin{aligned} \sigma \cdot \sigma' : \mathscr{P} &\longrightarrow \mathscr{F} \\ p &\longmapsto (p[\sigma])[\sigma]. \end{aligned}$$

Remarque :
L'opération $\cdot$ est associative.

Propriété : Soient σ et σ' deux substitutions, on a, pour toute formule $H \in \mathscr{F}$,

$$(H[\sigma])[\sigma'] = H[\sigma' \cdot \sigma].$$

■

Définition : On appelle *relation sous-formule*, la relation $\diamond$ définie dans la section 3 du chapitre -1.

2 Sémantique

2.1 Algèbre de Boole

Définition : On note $\mathbb{B} = \{\boldsymbol{V}, \boldsymbol{F}\}$ l'*ensemble des booléens*.

Définition : Sur $\mathbb{B}$, on définit les opérateurs

a	b	$a \cdot b$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$

Table 1 – Opération $\cdot$ sur les booléens

a	b	$a+b$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$

Table 2 – Opération $+$ sur les booléens

a	$\bar{a}$
$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$

Table 3 – Opération $\bar{\square}$ sur les booléens

Remarque :

Nom	$\cdot$	$+$
Commutativité	$a \cdot b = b \cdot a$	$a + b = b + a$
Neutre	$\boldsymbol{V} \cdot a = a$	$\boldsymbol{F} + a = a$
Absorbant	$\boldsymbol{F} \cdot a = \boldsymbol{F}$	$\boldsymbol{V} \cdot a = \boldsymbol{V}$
Associativité	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$a + (b + c) = (a + b) + c$
Idempotence	$a \cdot a = a$	$a + a = a$
Distributivité	$a \cdot (b + c) = a \cdot b + a \cdot c$	$a + (b \cdot c) = (a + b) \cdot (a + c)$
Complémentaire	$a \cdot \bar{a} = \boldsymbol{F}$	$a + \bar{a} = \boldsymbol{V}$
Morgan	$\overline{a \cdot b} = \bar{a} + \bar{b}$	$\overline{a + b} = \bar{a} \cdot \bar{b}$

Table 4 – Règles dans $\mathbb{B}$

2.2 Fonctions booléennes

Définition (Environnement propositionnel) **:** On appelle *environnement propositionnel* une fonction de $\mathcal{P}$ dans $\mathbb{B}$.

Définition : On appelle *fonction booléenne* une fonction de $\mathbb{B}^{\mathcal{P}}$ dans $\mathbb{B}$. On note l'ensemble des fonctions booléennes $\mathbb{F}$.

Remarque :
Si $|\mathcal{P}| = n$, alors $\left|\mathbb{B}^{\mathcal{P}}\right| = 2^n$ et donc $|\mathbb{F}| = 2^{2^n}$.

2.3 Interprétation d'une formule comme une fonction booléenne

Définition (Interprétation) **:** Étant donné une formule $G \in \mathcal{F}$ et un environnement propositionnel $\rho \in \mathbb{B}^{\mathcal{P}}$, on définit l'*interprétation* de G dans l'environnement ρ par

— $[\![\top]\!]^{\rho} = \boldsymbol{V}$;

— $[\![\bot]\!]^\rho = \boldsymbol{F}$;
— $[\![p]\!]^\rho = \rho(p)$ où $p \in \mathscr{P}$;
— $[\![\neg G]\!]^\rho = \overline{[\![G]\!]^\rho}$;
— $[\![G \wedge H]\!]^\rho = [\![G]\!]^\rho \cdot [\![H]\!]^\rho$;
— $[\![G \vee H]\!]^\rho = [\![G]\!]^\rho + [\![H]\!]^\rho$;
— $[\![G \to H]\!]^\rho = \overline{[\![G]\!]^\rho} + [\![H]\!]^\rho$;
— $[\![G \leftrightarrow H]\!]^\rho = \left(\overline{[\![G]\!]^\rho} + [\![H]\!]^\rho\right) \cdot \left(\overline{[\![H]\!]^\rho} + [\![G]\!]^\rho\right)$.

Définition (Fonction booléenne associée à une formule) **:** Étant donné une formule G, on note

$$\begin{aligned} \mathbb{F} \ni [\![G]\!] : \mathbb{B}^{\mathscr{P}} &\longrightarrow \mathbb{B} \\ \rho &\longmapsto [\![G]\!]^\rho . \end{aligned}$$

2.4 Liens sémantiques

Définition : On dit que G et H sont *équivalents* si et seulement si $[\![G]\!] = [\![H]\!]$. On note alors $G \equiv H$.

Définition (Conséquence sémantique) **:** On dit que H est *conséquence sémantique* de G dès lors que

$$\forall \rho \in \mathbb{B}^{\mathscr{P}},\ \left([\![G]\!]^\rho = \boldsymbol{V}\right) \implies \left([\![H]\!]^\rho = \boldsymbol{V}\right).$$

On le note $G \models H$.

Propriété : On a

$$G \equiv H \iff (G \models H \text{ et } H \models G).$$

■

Remarque :
$\models$ n'est pas une relation d'ordre.

Remarque :
La relation $\equiv$ est une relation d'équivalence. De plus, si $G \equiv G'$ et $H \equiv H'$, alors

— $G \wedge H \equiv G' \wedge H'$;
— $G \vee H \equiv G' \vee H'$;
— $G \to H \equiv G' \to H'$;
— $G \leftrightarrow H \equiv G' \leftrightarrow H'$;
— $\neg G \equiv \neg G'$.

Une telle relation est parfois appelée une *congruence*.

Définition : On dit d'une formule $H \in \mathcal{F}$ qu'elle est

— *valide* ou *tautologique* dès lors que $\forall \rho \in \mathbb{B}^{\mathcal{P}}$, $[\![H]\!]^\rho = \boldsymbol{V}$;

— *satisfiable* dès lors qu'il existe $\rho \in \mathbb{B}^{\mathcal{P}}$, $[\![H]\!]^\rho = \boldsymbol{V}$;

— *insatisfiable* dès lors qu'il n'est pas satisfiable.

On dit de $\rho \in \mathbb{B}^{\mathcal{P}}$ tel que $[\![H]\!]^\rho = \boldsymbol{V}$ que ρ est un *modèle* de H.

Définition : Si Γ est un ensemble de formules, on écrit $\Gamma \models H$ pour dire que

$$\forall \rho \in \mathbb{B}^{\mathcal{P}}, (\forall G \in \Gamma, [\![G]\!]^\rho = \boldsymbol{V}) \implies [\![H]\!]^\rho = \boldsymbol{V}.$$

Remarque :
Si Γ est fini, alors on a

$$\Gamma \models H \iff \Big(\bigwedge_{G \in \Gamma} G \Big) \models H.$$

On doit faire la preuve, pour $n \geqslant 1$,

$$\{G_1, G_2, \dots, G_n\} \models H \iff (\cdots((G_1 \wedge G_2) \wedge G_3) \cdots \wedge G_n) \models H.$$

3 Le problème Sat – Le problème Validité

On définit le problème Sat comme ayant pour donnée une formule H et pour question "H est-elle satisfiable ?" et le problème Valide comme ayant pour donnée une formule H et pour question "H est-elle valide ?"

3.1 Résolution par tables de vérité

Le problème Sat lit la colonne résultat, on cherche un $\boldsymbol{V}$. Le problème Valide lit la colonne résultat et vérifie qu'il n'y a que des $\boldsymbol{V}$.

Remarque :
Deux formules sont équivalent si et seulement si elles ont la même colonne résultat.

On essaie d'énumérer toutes les possibilités : si $|\mathcal{P}| = n \in \mathbb{N}$, alors le nombre de classes d'équivalences pour $\equiv$ est au plus 2^{2^n}. On cherche donc un meilleur algorithme.

4 Représentation des fonction booléennes

4.1 Par des formules ?

p	q	r	S
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$	$\boldsymbol{F}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$	$\boldsymbol{V}$
$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{V}$	$\boldsymbol{F}$

Table 5 – Table de vérité d'une formule inconnue

On regarde les cas où la sortie est $\boldsymbol{V}$ et on crée une formule permettant de tester cette combinaison de p, q et r uniquement. On unie toutes ces formules par des $\vee$. Dans l'exemple ci-dessus, on obtient

$$(\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r).$$

Théorème : Soit $f : \mathbb{B}^{\mathscr{P}} \to \mathbb{B}$ une fonction booléenne avec $\mathscr{P}$ fini. Il existe une formule $H \in \mathscr{F}$ telle que $[\![H]\!] = f$.

Avant de prouver ce théorème, on démontre d'abord les deux lemme suivants et on définit lit_ρ.

Définition : Soit $\rho \in \mathbb{B}^{\mathscr{P}}$. On définit

$$\mathrm{lit}_\rho(p) = \begin{cases} p & \text{si } \rho(p) = \boldsymbol{V}; \\ \neg p & \text{sinon.} \end{cases}$$

Lemme :

$$\forall \rho \in \mathbb{B}^{\mathscr{P}},\ \exists G \in \mathscr{F},\ \left(\forall \rho' \in \mathbb{B}^{\mathscr{P}},\ [\![G]\!]^{\rho'} = \boldsymbol{V} \iff \rho = \rho'\right).$$

On prouve ce lemme : ■

On peut donc maintenant prouver le théorème :

Lemme : Considérons alors la formule

$$H = \bigvee_{\substack{\rho \in \mathbb{B}^{\mathscr{P}} \\ f(\rho) = \boldsymbol{V}}} H_\rho.$$

On a $[\![H]\!] = f$.

■

Le théorème est prouvé directement à l'aide des deux lemmes précédents.

On connaît donc la réponse à la question du nom de ce paragraphe, à savoir "peut-on repré-

senter les fonctions booléennes par des formules ?" Oui.

4.2 Par des formules sous formes normales ?

Définition : On dit d'une formule de la forme

- p ou $\neg p$ avec $p \in \mathcal{P}$, que c'est un *littéral* ;
- $\bigwedge_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux que c'est une *clause conjonctive* ;
- $\bigvee_{i=1}^{n} \ell_i$ où les ℓ_i sont des littéraux que c'est une *clause disjonctive* ;
- $\bigwedge_{i=1}^{n} D_i$ où les D_i qui sont des clauses disjonctives est appelée une *forme normale conjonctive* ;
- $\bigvee_{i=1}^{n} C_i$ où les C_i qui sont des clauses conjonctives est appelée une *forme normale disjonctive*.

Remarque :
On prend, comme convention, que $\bigwedge_{i=1}^{0} G_i = \top$ et $\bigvee_{i=1}^{0} G_i = \bot$.

Remarque :
On écrit FMD pour une forme normale disjonctive et FNC pour une forme normale conjonctive.

Théorème : Toute formule est équivalente à une formule sous FND et à une formule sous FNC.

■

Nous n'avons pas encore prouvé la deuxième partie du théorème mais, on essaie de trouver une formule sous FNC :

Remarque :
Il est en fait possible de transformer une formule en FND en appliquant les règles suivantes à toutes les sous-formules jusqu'à obtention d'un point fixe.

- $\neg\neg H \rightsquigarrow H$;
- $\neg(G \wedge H) \rightsquigarrow G \vee H$;
- $\neg(G \vee H) \rightsquigarrow G \wedge H$;
- $(G \vee H) \wedge I \rightsquigarrow (G \wedge I) \vee (H \wedge I)$;
- $I \wedge (G \vee H) \rightsquigarrow (I \wedge G) \vee (I \wedge H)$;

- $H \wedge \top \rightsquigarrow H$;
- $\top \wedge H \rightsquigarrow H$;
- $H \vee \bot \rightsquigarrow H$;
- $\bot \vee H \rightsquigarrow H$;
- $\neg\top \rightsquigarrow \bot$;
- $\bot \wedge H \rightsquigarrow \bot$;
- $H \wedge \bot \rightsquigarrow \bot$;
- $\neg\bot \rightsquigarrow \top$;
- $\top \vee H \rightsquigarrow \top$;
- $H \vee \top \rightsquigarrow \top$.

Propriété : Soit $n \geqslant 2$ et H_n la formule $H_n = (a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_n \vee b_n)$ avec $\mathscr{P}_n = \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}$. Alors, par application de l'algorithme précédent on obtient

$$\bigvee_{P \in \wp([\![1,n]\!])} \left(\bigwedge_{j=1}^{n} \begin{cases} a_j \text{ si } j \in P \\ b_j \text{ sinon} \end{cases} \right).$$

À faire : ■

Remarque :
Qu'en est-il du problème Sat ? Le problème est-il simplifié pour les fnd ou les fnc ?

Oui, pour les fnd, le problème se simplifie. On considère, par exemple, la formule

$$\begin{array}{ll} & (\ell_{11} \wedge \ell_{12} \wedge \cdots \wedge \ell_{1,n_1}) \\ \vee & (\ell_{21} \wedge \ell_{22} \wedge \cdots \wedge \ell_{2,n_2}) \\ \vdots & \vdots \\ \vee & (\ell_{m,1} \wedge \ell_{m,2} \cdots \ell_{m,n_m}). \end{array}$$

On procède en suivant l'algorithme suivant : (À faire : Mettre l'algorithme à part) Pour i fixé, je lis la ligne i, puis je fabrique un environnement ρ.

Par exemple, pour $(p \wedge \neg q \wedge r \wedge \neg p) \vee (q \wedge r \wedge \neg q) \vee (p \wedge r)$, on a $\rho = (p \mapsto \boldsymbol{V}, r \mapsto \boldsymbol{V})$.

On en conclut que Sat peut être résolu en temps linéaire dans le cas d'une forme normale disjonctive. Le problème est de construire cette fnd.

Remarque :
Après s'être intéressé au problème Sat, on s'intéresse au problème Valide.

Par exemple, on considère la formule $(p \vee q \vee \neg r \vee \neg p) \wedge (p \vee \neg r \vee p \vee r) \wedge (q \vee r)$. On peut construire $\rho = (q \mapsto \boldsymbol{F}, r \mapsto \boldsymbol{F})$ est tel que $[\![H]\!]^\rho = \boldsymbol{F}$.

Si on ne peut pas construire un tel environnement propositionnel, la formule vérifie le problème Valide.

On en conclut que Valide peut être résolu en temps linéaire dans le cas d'une forme normale conjonctive. Le problème est de construire cette fnc.

5 Algorithme de Quine

L'objectif de cet algorithme est de résoudre le problème Sat. On commence par poser quelques lemmes, puis on donne l'algorithme.

Remarque :
Une forme normale peut être vue comme un ensemble d'ensembles de littéraux (c'est la représentation que nous allons utiliser en OCamL).

Lemme : Pour toute formule H, pour tout variable propositionnelle p et pour tout

environnement propositionnel ρ, tel que $\rho(p) = \boldsymbol{V}$, alors

$$\left[\!\left[H[p \mapsto \top]\right]\!\right]^{\rho} = [\![H]\!]^{\rho}\,.$$

Remarque :
Le résultat reste vrai en remplaçant $\boldsymbol{V}$ par $\boldsymbol{F}$ et $\top$ par $\bot$.

Lemme : Pour toute formule H, et pour toute variable propositionnelle p, H est satisfiable si, et seulement si $H[p \mapsto \top]$ est satisfiable ou $H[p \mapsto \bot]$ est satisfiable.

Lemme : Une formule sans variables est

— satisfiable si et seulement si elle est équivalente à $\top$;

— insatisfiable si et seulement si elle est équivalente à $\bot$;

```
type formule =
  | Top | Bot
  | Var   of string
  | Not   of formule
  | And   of formule * formule
  | Or    of formule * formule
  | Imply of formule * formule
  | Equiv of formule * formule

let substitution (f: formule) (x: string) (g: string): formule =
  ⟨code déjà fait en TP⟩

exception Found of string (* arret de la boucle *)
let get_var (f: formule): string option =
  let rec aux (f: formule): unit =
    match f with
    | Top | Bot       -> ()
    | Var(y)          -> raise(Found(y))
    | Not(f1)         -> aux f1
    | And(f1, f2)
    | Or(f1, f2)
    | Imply(f1, f2)
    | Equiv(f1, f2)   -> (aux f1; aux f2)
  in try
    aux f;
    None
  with
  | Found(y) -> Some(y)

let rec test_equiv (f: formule): bool option =
  match f with
  | Var(_) -> None
  | Top    -> Some(true)
  | Bot    -> Some(false)
  | Not(h) ->
            begin
              match test_equiv h with
              | None    -> None
              | Some(b) -> Some(not b)
            end
  | And(h1, h2) ->
            begin
              match test_equiv h1, test_equiv h2 with
```

```
44              | Some(false), _ | _, Some(false) -> Some(false)
45              | Some(true), Some(true)          -> Some(true)
46              | _                               -> None
47            end
48    | Or(h1, h2) ->
49            begin
50              match test_equiv h1, test_equiv h2 with
51              | Some(true), _ | _, Some(true) -> Some(true)
52              | Some(false), Some(false)      -> Some(true)
53              | _                             -> None
54            end
55    | Imply(h1, h2) ->
56            begin
57              match test_equiv h1, test_equiv h2 with
58              | Some(false), _
59              | Some(true), Some(true)       -> Some(true)
60              | Some(true), Some(false)      -> Some(false)
61              | _                            -> None
62            end
63    | Equiv(h1, h2) ->
64            begin
65              match test_equiv h1, test_equiv h2 with
66              | Some(true), Some(true)
67              | Some(false), Some(false)     -> Some(true)
68              | Some(true), Some(false)
69              | Some(false), Some(true)      -> Some(false)
70              | _                            -> None
71            end
72
73  let rec quine (f: formule): bool =
74    match get_var f, test_equiv f with
75    | _, Some(b)    -> b
76    | None, _       -> failswith "cas impossible"
77    | Some(p), None -> ?
```

Code 1 – Algorithme de Quine *version zéro*

> Remarque :
> Dans la suite, on s'intéresse aux formules sous forme cnf. Une cnf (ou même une dnf) peut être représentée au moyen d'un ensemble d'ensemble de littéraux. Ces ensembles sont finis. Par exemple, la formule $(p \vee \neg q) \wedge (r \vee p \vee p)$ est représenté par
>
> $$\{\{p, \neg q\}, \{r, p\}\}.$$

Algorithme 1 Algorithme *Assume*

Entrée G une cnf, p une variable propositionnelle, et $b \in \mathbb{B}$.
Sortie Une cnf équivalente à $G[p \mapsto b]$.

```
1 : Soit ℓ_V le littéral p si b = V, ¬p sinon.
2 : Soit ℓ_F le littéral ¬p si b = V, p sinon.
3 : pour C ∈ G faire
4 :     si ℓ_V ∈ C alors
5 :         On retire C de G.
6 :     sinon
7 :         si ℓ_F ∈ C alors
8 :             On retire ℓ_F de C.
```

On peut donc donner l'algorithme de Quine final :

Algorithme 2 Algorithme de QUINE

Entrée Une CNF G

```
si G = ∅ alors
    retourner OUI
sinon
    si ∅ ∈ G alors
        retourner NON
    sinon si ∃{ℓ} ∈ G alors
        si ℓ = p, avec p ∈ vars(G) alors
            QUINE(Assume(G, p, V))
        sinon si ℓ = ¬p, avec p ∈ vars(G) alors
            QUINE(Assume(G, p, F))
    sinon
        p ← h(G)
        On essaie QUINE(Assume(G, p, V))
        On essaie QUINE(Assume(G, p, F))
```

6 Synthèse du chapitre

— Formules —

On définit l'ensemble des formules $\mathcal{F}$ par induction nommé avec les règles

- $\neg|^1$;
- $\wedge|^2$;
- $\vee|^2$;
- $\to|^2$;
- $\leftrightarrow|^2$;
- $\top|^0$;
- $\bot|^0$;
- $V|^0_{\mathcal{P}}$.

On définit inductivement $\text{taille}(F)$, pour $F \in \mathcal{F}$, la taille de cette formule, *i.e.* le nombre d'opérateurs dans cette formule. On définit également l'ensemble des variables $\text{vars}(F)$ d'une formule $F \in \mathcal{F}$.

Une *substitution* est une fonction de $\mathcal{P}$ dans $\mathcal{F}$, où elle est l'identité partout, sauf en nombre fini de variables, alors nommés *clés* de cette substitution. On note $F[\sigma]$ l'application d'une substitution σ à une formule $F \in \mathcal{F}$. On définit la composée de deux substitutions $\sigma \cdot \sigma'$, comme $\sigma \cdot \sigma' : p \mapsto (p[\sigma])[\sigma']$, cela ne correspond pas à la définition mathématique d'une composition de fonctions.

— Fonctions booléennes —

Un *environnement propositionnel* est une fonction de $\mathcal{P}$ dans $\mathbb{B}$. Une *fonction booléenne* est une fonction de $\mathbb{B}^{\mathcal{P}}$ dans $\mathbb{B}$. L'ensemble $\mathbb{F}$ est l'ensemble des fonctions booléennes.

On définit inductivement l'*interprétation* d'une formule $F \in \mathcal{F}$ dans un environment ρ. On note ce booléen $[\![F]\!]^\rho$. On note également $[\![F]\!]$ l'application $\rho \mapsto [\![F]\!]^\rho$.

— Liens sémantiques —

On note $G \equiv H$ si, et seulement si $[\![G]\!] = [\![H]\!]$. On note $G \models H$ dès lors que, pour $\rho \in \mathbb{B}^{\mathcal{P}}$, si $[\![G]\!]^\rho = \boldsymbol{V}$, alors $[\![H]\!]^\rho = \boldsymbol{V}$. On étend cette définition pour un ensemble Γ de formules G. On a $G \equiv H$ si, et seulement si $G \models H$ et $H \models G$.

Une formule *valide* ou *tautologique* est une formule dont l'interprétation vaut toujours $\boldsymbol{V}$, peu importe l'environnement propositionnel. Une formule satisfiable est une formule dont l'interprétation vaut $\boldsymbol{V}$, pour un certain environnement propositionnel. Si $\rho \in \mathbb{B}^{\mathcal{P}}$ vérifie $[\![F]\!]^\rho = \boldsymbol{V}$, on dit que ρ est un *modèle* de F.