Chapitre 3

Logique des propositions

la logique est une branche des mathématiques, de l'informatique et de la philosophe=ie, qui cherche à étudier la notion de vérité en mathématiques. La difficulté par rapport à d'autres branches est qu'il y existe deux niveaux de discours : le langage, qui décrit les objets mathématiques qu'on étudie, et le méta-langage, dont on se sert pour en parler. Mais certains des mots sont communs au langage et au méta-langage, la phrase suivante peut par exemple être ambigüe

Pour toutes formules a, b, c, a et $(b$ et $c) \Leftrightarrow (a$ et $b)$ et c et a et $b \Leftrightarrow b$ et a.

Le "et" souligné souligne-t-il qu'on a énoncé deux résultats (*i.e.* appartient au méta-langage), ou qu'on a énoncé un résultat qui est une conjonction logique (*i.e.* appartient au langage de la logique) ?

Il faut donc prendre soin de bien définir le langage de la logique, en s'efforçant de le différencier du méta-langage.

Il faut aussi faire attention à différencier la *syntaxe* du langage et sa *sémantique* : la syntaxe est l'ensemble de règles qu'on utilise pour construire des formules logiques, qui doivent être valides (par exemple, $\forall x \exists \wedge$ n'est pas valide), alors que la sémantique regarde leur sens, et éventuellement leur vérité ($\forall x,\ x \neq x$ est une formule valide, mais fausse).

3.1 Syntaxe

Commençons tout de suite par fixer un ensemble dénombrable $V = \{v_1, \ldots, v_n, \ldots\}$. Les éléments de V sont des

NOTA

Dans toute la suite, on ne manipulera qu'un nombre fini de formules, ne contenant chacune qu'un nombre fini de variables. On pourrait donc se limiter à un ensemble fini de variables, mais autant voir les choses en grand.

Une logique avec seulement des variables étant peu intéressante, ajoutons-y des symboles :

- unaire :
- binaires :
- et des parenthèses

L'alphabet qu'on utilisera pour construire les formule sera donc un alphabet infini

$$\Sigma =$$

On a alors deux façons (équivalentes) de définir les formules propositionnelles :

Définition 3.1.1 : Définition par le haut
Le langage des formules propositionnelles $\mathcal{L}$ est le plus petit langage sur Σ tel que

-
-
-

Définition 3.1.2 : Définition par le bas
On pose $\mathcal{L}_0 =$ et pour tout $n \in \mathbb{N}$,

$\mathcal{L}_{n+1} =$

On pose alors $\mathcal{L} =$.

Proposition 3.1.3
Les deux définitions sont équivalentes.

Démonstration. En exercice. □

On pourra alors, comme dans le cas des arbres, prouver des propriétés sur les formules et y définir des fonctions par induction :

Proposition 3.1.4 : Définition de fonction sur $\mathcal{L}$ par induction
Soient E un ensemble, $f_0 : V \to E$, $f_1 : E \to E$ et $f_2 : E \times \{\wedge, \vee, \to, \leftrightarrow\} \times E \to E$. Alors il existe une unique fonction $F : \mathcal{L} \to E$ telle que

-
-

Définition 3.1.5
Définissons la hauteur h d'une formule par induction :

-
-
-

NOTA

Pour "coller" au théorème, on définit la hauteur avec

Proposition 3.1.6 : Induction structurelle sur les formules propositionnelles
Soit P une propriété sur les mots de Σ^*. Si

-
-

alors pour toute formule $\chi \in \mathcal{L}$,

3.1.1 Représentation sous forme d'arbres

Tout ceci nous fait beaucoup penser au chapitre sur les arbres. En effet, on peut représenter les formules par des arbres, dont les feuilles seront les variables, et les nœuds les connecteurs.

```
type var =
type binop =

type formule=









let rec hauteur f =
```

EXERCICE

On définit par induction l'ensemble sous-formules d'une formule χ, noté $\mathrm{SF}(\chi)$:

- $\mathrm{SF}(v_i) = \{v_i\}$
- $\mathrm{SF}(\neg\chi) = \{\neg\chi\} \cup \mathrm{SF}(\chi)$
- $\mathrm{SF}(\chi \alpha \psi) = \{\chi \alpha \psi\} \cup \mathrm{SF}(\chi) \cup \mathrm{SF}(\psi)$

Écrire une fonction `sous_formules:formule -> formule list` donnant la liste des sous-formules d'une formule.

Définition 3.1.7 : Substitution
Soient v une variable et χ et ψ deux formules. On définit la substitution de x par ψ dans χ, notée $\chi\left[x/\psi\right]$ par induction sur ψ :

-
-
-

EXEMPLE

Considérons $\chi = \neg((a \wedge b) \rightarrow a)$ et $\psi = a \vee b$. Alors

$$\chi \left[a/\psi \right] =$$

NOTA

On peut aussi définit des substitutions multiples simultanées $\chi \left[v_i/\psi_i, v_j/\psi_j, v_k/\psi_k, \cdots/\ldots \right]$, qui ne correspondent pas du tout à la suite de substitution $\chi \left[v_i/\psi_i \right] \left[v_j/\psi_j \right] \left[v_k/\psi_k \right] \left[\cdots/\ldots \right]$.

EXERCICE

Écrire la fonction OCaml correspondante.

3.2 Sémantique

Maintenant qu'on sait comment former des formules valides d'un point de vue syntaxique, il convient de donner un sens à ces formules.

Définition 3.2.1
On appelle environnement *ou* valuation

On note que l'ensemble d'arrivée pourrait être n'importe quel ensemble à deux éléments, par exemple `bool`.

Une valuation nous donnera des valeurs pour les variables propositionnelles ; il reste donc à propager ces valeurs à travers les connecteurs.

Définition 3.2.2
On définit les fonctions suivantes :

- $F_\neg : \underset{x}{\{0,1\}} \underset{\longmapsto}{\longrightarrow} \{0,1\}$
- $F_\vee : \underset{x}{\{0,1\}^2} \underset{\longmapsto}{\longrightarrow} \{0,1\}$
- $F_\leftrightarrow : \underset{x}{\{0,1\}^2} \underset{\longmapsto}{\longrightarrow} \{0,1\}$
- $F_\wedge : \underset{x}{\{0,1\}^2} \underset{\longmapsto}{\longrightarrow} \{0,1\}$
- $F_\rightarrow : \underset{x}{\{0,1\}^2} \underset{\longmapsto}{\longrightarrow} \{0,1\}$

EXERCICE

Vérifier que les fonctions ci-dessus correspondent bien aux valeurs voulues, *e.g.* $F_\wedge(x,y) = 1$ si et seulement si $x = y = 1$.

On peut enfin évaluer notre formule :

Définition 3.2.3
On définit la fonction eval *qui prend une formule et une valuation en renvoie un élément de* $\{0,1\}$ *par induction :*

-
-
-

eval *est appelée*

En OCaml, nous utiliserons plutôt le type `bool` pour l'ensemble d'arrivée des valuations :

```
let rec eval f gamma =
```

EXERCICE

Réécrire cette fonction en prenant `int` comme ensemble d'arrivée.

Un moyen simple de représenter toutes les valuations possibles est de faire une *table de vérité*, ce qui est possible grâce à

Proposition 3.2.4
Soit χ une formule. Alors

Pour faire une table, on respectera les conventions suivantes :

- Sur la première ligne, on écrit :
  -
  -
  -
- Sur les lignes suivantes,

EXEMPLE

Regardons la table de vérité de $\chi = ((a \to b) \to a) \to a$:

a	b	$a \to b$	$(a \to b) \to a$	χ

Donnons maintenant un sens à nos formules :

Définition 3.2.5
Soit χ une formule. On dit qu'une valuation γ satisfait χ si

On dit alors que χ est satisfiable

Une formule non satisfiable s'appelle

Une formule satisfaite par toutes les valuations s'appelle une

EXERCICE

Montrer que χ est une tautologie si et seulement si $\neg\chi$ est une contradiction.

En fait, la table de vérité d'une formule nous donne directement sa nature : s'il n'y a que des 0 dans la dernière colonne, c'est ; s'il n'y a que des 1, c'est ; sinon, c'est

EXEMPLE

La formule $((a \to b) \to a) \to a$ est une tautologie : elle s'appelle *.

Maintenant, on aimerait pouvoir définir des classes de tautologies. On utilisera le théorème suivant.

Théorème 3.2.6
Soient $\chi, \psi_1, \ldots, \psi_n$ des formules, et $v_1 \ldots, v_n$ des variables distinctes.

Alors si χ est une tautologie, alors

est aussi une tautologie.

Démonstration. On le fait dans le cas d'une seule substitution $\chi\left[v/\psi\right]$. Soit γ une valuation. On définit une nouvelle valuation γ' définie par

$\gamma'(v) =$

On montre facilement par induction que $\text{eval}(\chi\left[v/\psi\right], \gamma) =$.

On a donc pour toute valuation γ une valuation γ' telle que $\text{eval}(\chi\left[v/\psi\right], \gamma) =$ $trouev(\chi, \gamma') = 1$ puisque χ est une tautologie. □

Avec une tautologie, on peut donc en créer des nouvelles en remplaçant les variables par des formules.

EXEMPLE

Pour toutes formules χ et ψ, $((\chi \to \psi) \to \chi) \to \chi$ est une tautologie.

Cherchons les tautologies en OCaml. Pour cela, on va modifier un peu la fonction d'évaluation, pour qu'elle puisse ne prendre en compte que les variables réellement présentes dans la formule.

Une valuation est alors une liste de couples (*variable*,*valeur*).

```
let rec eval f gamma =
```

On peut alors récupérer la liste des variables d'une formule.

```
let rec liste_var f =
```

On veut maintenant obtenir la liste des valuations satisfaisant une formule. Pour cela, on génère toutes les valuations possibles (complexité 2^n), puis on les teste une par une.

```
let rec liste_valuation (l:char list) =
```

```
let table_verite f =
```

EXERCICE

Écrire une fonction permettant d'afficher la table de façon lisible.

On peut maintenant créer la liste des valuations satisfaisant une formule ; si elle est vide, la formule est une contradiction. Pour tester une tautologie, on vérifie que sa négation est une contradiction.

```
let satisfiable_vals f =


let est_contradiction f =


let est_tautologie f =
```

Définition 3.2.7
On dit que deux formules χ et ψ sont équivalentes, et on note $\chi \equiv \psi$ si

On note que c'est complètement équivalent à dire que

Proposition 3.2.8
Pour toutes formules χ et ψ, on a

$$\chi \equiv \psi \text{ si et seulement si}$$

3.3 Exercices

Arbres binaires

Exercice 1. On appelle *système complet de connecteurs* tout ensemble de connecteur logique permettant d'exprimer n'importe quel formule.

(i) Montrer que $\{\neg, \wedge, \vee\}$ est un SCC.

(ii) Montrer que $\{\neg, \wedge\}$ est un SCC.

(iii) On appelle *connecteur de Sheffer* le connecteur défini par $v|w = \neg(v \wedge w)$. Montrer que $\{|\}$ est un SCC.

Exercice 2. Les expressions suivantes sont-elles des tautologies, des contradictions, ou juste satisfiables ?

(i) $A \to (B \to C) \leftrightarrow (A \wedge B) \to C$

(ii) $((A \to B) \to C) \leftrightarrow (A \to (B \to C))$

(iii) $(A \to (B \to A)) \to A$

Exercice 3. Sur une île, on distingue deux types de gens :

- Les Sincères, qui disent tout le temps la vérité ;
- Les Menteurs, qui ne disent jamais la vérité.

On prend deux personnes aux hasard sur l'île : Alice et Bob.

On note A la variable "Alice est Sincère", et B la variable "Bob est Sincère".

Dans les deux cas suivants, traduire l'énoncé en terme de formule logique, et dire si Alice et Bob sont Sincères ou Menteurs :

(i) Bob dit : "Nous sommes tous les deux des Menteurs".

(ii) Alice dit : "Je ne suis ni une Sincère, ni une menteuse". Bob ajoute "C'est vrai".

Exercice 4. Soit Σ un ensemble de formules (fini ou infini). On dit que Σ est satisfiable s'il existe une même valuation γ qui satisfait toutes les formules de Σ. On veut montrer le

Théorème 3.3.1 : de compacité
Σ est satisfiable si et seulement si toute partie finie de Σ est satisfiable.

(i) Montrer le sens direct.

(ii) Supposons maintenant que toute partie de Σ est satisfiable.
On veut construire par récurrence une suite $(\varepsilon_n) \in \{0,1\}^{\mathbb{N}^*}$ telle que pour tout $n \in \mathbb{N}$, on a la propriété P_n : "Pour toute partie finie de Σ, il existe une valuation γ qui la satisfait et telle que $\forall i \in [\![1, n]\!],\ \gamma(v_i) = \varepsilon_i$".
Commencer par montrer P_0.

(iii) Supposons que $(\varepsilon_1, \ldots, \varepsilon_n)$ sont construits et satisfont P_n.
Cas 1 : Pour toute partie finie de Σ, il existe une valuation γ qui la satisfait et telle que $\forall i \in [\![1, n]\!],\ \gamma(v_i) = \varepsilon_i$ et $\gamma(v_{i+1}) = 0$.
Conclure ce cas.

(iv) Conclure lorsque le cas 1 n'est pas vérifié.

(v) On a maintenant construit notre suite. Montrer que la valuation définie par γ_0 : $v_i \mapsto \varepsilon_i$ satisfait Σ.