

# Syntaxe des formules logiques

OPTION INFORMATIQUE - TP n° 1.1 - Olivier Reynet

## À la fin de ce chapitre, je sais :

- ✎ utiliser la définition inductive des formules logiques pour calculer la hauteur ou la taille
- ✎ construire une table de vérité
- ✎ calculer une forme normale conjonctive ou disjonctive associée à une formule logique
- ✎ modéliser un énoncé simple du langage naturel en propositions logiques

## A Définition inductive des formules logiques

En s'inspirant de la définition inductive des formules logiques, on se dote du type de données OCaml formule suivant :

```
1 type formule =  
2   | T (* true *)  
3   | F (* false *)  
4   | Var of int (* variable *)  
5   | Not of formule (* negation *)  
6   | And of formule * formule (* conjonction *)  
7   | Or of formule * formule (* disjonction *)
```

L'expression `Var of int` signifie qu'on représente les variables d'une formule logique par un numéro, ce qui est le cas dans les logiciels standard de logique. Par exemple, pour les variables des formules ci-dessous, on peut choisir la convention suivante :  $a \rightarrow 0$ ,  $b \rightarrow 1$ ,  $c \rightarrow 2$ .

A1. Représenter les arbres syntaxiques des formules logiques suivantes, puis les coder en OCaml.

(a)  $f_1 : a \vee (b \wedge c)$

**Solution :**

```
1 let f1 = Or ((Var 0), And ((Var 1),(Var 2))));;
```

(b)  $f_2 : (a \wedge \neg b) \vee (b \wedge \neg(c \vee a))$

**Solution :**

```
1 let f2 = Or (And ((Var 0),(Not (Var 1))), And ((Var 1), Not(Or ((Var 2)  
    ,(Var 0)))));;
```

(c)  $f_3 : \neg a \vee (a \implies b)$ **Solution :** On transforme l'implication en l'exprimant à l'aide des opérateurs premiers :

$$\neg a \vee (a \implies b) \equiv \neg a \vee (\neg a \vee b) \equiv \neg a \vee b$$

```
1 let f3 = Or (Not (Var 0), (Var 1));;
```

---

(d)  $f_4 : (a \wedge (b \iff c))$ **Solution :** On transforme l'équivalent matérielle en l'exprimant à l'aide des opérateurs premiers : il s'agit en fait d'une double implication.

$$a \wedge (b \iff c) = a \wedge (b \implies c) \wedge (c \implies b) = a \wedge (\neg b \vee c) \wedge (\neg c \vee b)$$

```
1 let f4 = And ((Var 0), And (Or (Not (Var 1), (Var 2)), Or (Not (Var 2), (Var 1))));;
```

---

- A2. Proposer une fonction de signature `valuation : int -> bool` qui implémente une valuation des variables propositionnelles  $a$ ,  $b$  et  $c$  telle que les formules  $f_1$  et  $f_2$  soient vraies et  $f_3$  et  $f_4$  fausses. On utilisera le filtrage de motif. Les variables sont représentées par leur numéro :  $a \longrightarrow 0$ ,  $b \longrightarrow 1$ ,  $c \longrightarrow 2$ . Si le numéro de variable n'est pas connu, la fonction échoue et renvoie le message "Unknown variable"!.

**Solution :**

```
1 let valuation i =
2   match i with
3   | 0 -> true  (* a valuation *)
4   | 1 -> false (* b valuation *)
5   | 2 -> true  (* c valuation *)
6   | _ -> failwith "Unknown variable !";;
```

---

- A3. Écrire une fonction **récursive** de signature `evaluation : (int -> bool) -> formule -> bool` qui évalue une formule logique d'après une valuation donnée. On utilisera le filtrage de motif. Cette fonction s'appuie sur la définition inductive de l'évaluation telle que définie dans le cours.

**Solution :**

```
1 let rec evaluation v f =
2   match f with
3   | T -> true
4   | F -> false
5   | Var x -> v x
6   | Not p -> not (evaluation v p)
7   | And (p, q) -> evaluation v p && evaluation v q
```

---

```

8      | Or (p, q) -> evaluation v p || evaluation v q
9      ;;

```

---

A4. Vérifier que la valuation choisie est bien telle que spécifiée à la question A.2.

**Solution :**

```

1 evaluation valuation f1;;
2 evaluation valuation f2;;
3 evaluation valuation f3;;
4 evaluation valuation f4;;

```

---

## B Taille, hauteur et nombre de termes d'une formule logique

B1. En utilisant la définition de la fonction `taille` d'une formule logique, écrire une fonction de signature `taille : formule -> int` qui renvoie la taille d'une formule.

**Solution :**

```

1 let rec taille f =
2     match f with
3     | T -> 0
4     | F -> 0
5     | Var _ -> 0
6     | Not f -> 1 + (taille f)
7     | And (fa,fb) -> 1 + (taille fa) + (taille fb)
8     | Or (fa,fb) -> 1 + (taille fa) + (taille fb);;

```

---

B2. Vérifier sur les arbres des formules  $f_1$ ,  $f_2$ ,  $f_3$  et  $f_4$  les résultats produits par la fonction `taille`.

**Solution :**

```

1 taille f1;;
2 taille f2;;
3 taille f3;;
4 taille f4;;

```

---

B3. En utilisant la définition de la fonction `hauteur` d'une formule logique, écrire une fonction de signature `hauteur : formule -> int` qui renvoie la hauteur d'une formule.

**Solution :**

```

1 let rec hauteur f =
2     match f with

```

---

```

3      | T -> 0
4      | F -> 0
5      | Var _ -> 0
6      | Not f -> 1 + (hauteur f)
7      | And (fa,fb) -> 1 + (max (hauteur fa) (hauteur fb))
8      | Or (fa,fb) -> 1 + (max (hauteur fa) (hauteur fb));;

```

---

B4. Vérifier sur les arbres des formules  $f_1$ ,  $f_2$ ,  $f_3$  et  $f_4$  les résultats produits par la fonction hauteur.

**Solution :**

```

1 hauteur f1;;
2 hauteur f2;;
3 hauteur f3;;
4 hauteur f4;;

```

---

B5. On code conventionnellement les variables par des entiers en commençant par zéro. Proposer une fonction de signature `max_var : formule -> int -> int` qui renvoie l'entier le plus grand représentant une variable propositionnelle dans une formule. En déduire une fonction `nb_var : formule -> int` qui renvoie le nombre de variables propositionnelles utilisées dans une formule.

**Solution :**

```

1 let rec max_var f k = (* k is current index *)
2   match f with
3   | T | F -> k
4   | Var i -> max i k
5   | Not fa -> max_var fa k
6   | And (fa,fb) -> max_var fb (max_var fa k)
7   | Or (fa,fb) -> max_var fb (max_var fa k)
8
9 let nb_var f = (max_var f 0) + 1

```

---

B6. Vérifier sur  $f_1$ ,  $f_2$ ,  $f_3$  et  $f_4$  les résultats produits par les fonction highest\_var et nb\_var.

**Solution :**

```

1 max_var f1;;
2 max_var f2;;
3 max_var f3;;
4 max_var f4;;
5
6 nb_var f1;;
7 nb_var f2;;
8 nb_var f3;;
9 nb_var f4;;

```

---

## C De la table de vérité à la forme normale (i)

On considère la formule  $\psi = (a \wedge \neg b) \vee (\neg a \wedge b)$ .

C1. Construire l'arbre syntaxique de  $\phi$ .



C2. Établir la table de vérité de  $\phi$ .

**Solution :**

$a$	$b$	$\neg b$	$a \wedge \neg b$	$\neg a$	$\neg a \wedge b$	$\phi$
F	F	T	F	T	F	F
F	T	F	F	T	T	T
T	F	T	T	F	F	T
T	T	F	F	F	F	F

C3. Proposer une forme normale disjonctive de  $\phi$

**Solution :** Il suffit de faire la disjonction des modèles de  $\phi$  :

$$\phi = (\neg a \wedge b) \vee (a \wedge \neg b)$$

On l'avait déjà;-)

C4. Proposer une forme normale conjonctive de  $\phi$

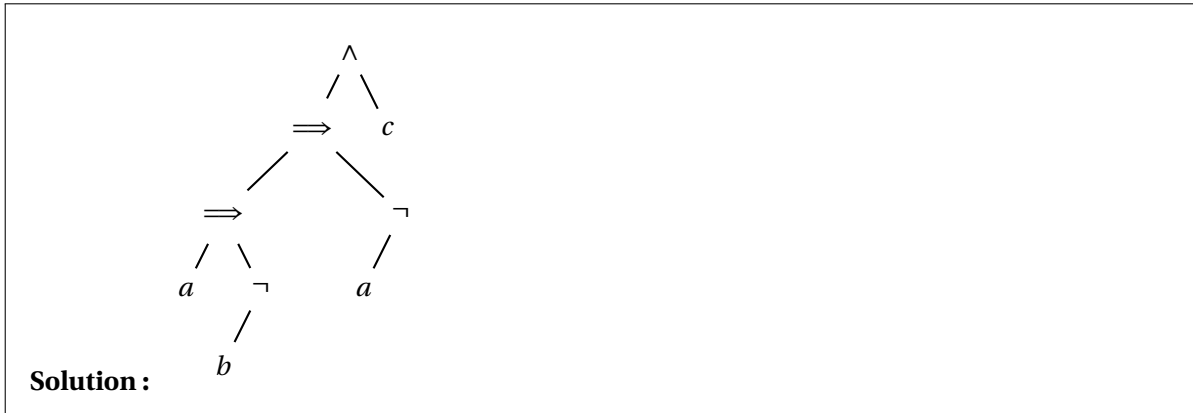
**Solution :** On sélectionne les contre-modèles puis on prend la négation de chaque littéral pour construire une clause disjonctive.

$$\phi = (a \vee b) \wedge (\neg a \vee \neg b)$$

## D De la table de vérité à la forme normale (ii)

On considère la formule  $\phi = ((a \Rightarrow \neg b) \Rightarrow \neg a) \wedge c$ .

D1. Construire l'arbre syntaxique de  $\phi$ .



D2. Établir la table de vérité de  $\phi$ .

**Solution :**

$a$	$b$	$c$	$\neg b$	$a \Rightarrow \neg b$	$\neg a$	$(a \Rightarrow \neg b) \Rightarrow \neg a$	$\phi$
F	F	F	T	T	T	T	F
F	F	T	T	T	T	T	T
F	T	F	F	T	T	T	F
F	T	T	F	T	T	T	T
T	F	F	T	T	F	F	F
T	F	T	T	T	F	F	F
T	T	F	F	F	F	T	F
T	T	T	F	F	F	T	T

D3. Proposer une forme normale disjonctive de  $\phi$

**Solution :** Il suffit de faire la disjonction des modèles de  $\phi$  :

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

D4. Proposer une forme normale conjonctive de  $\phi$

**Solution :** On sélectionne les contre-modèles puis on prend la négation de chaque littéral pour construire une clause disjonctive. Par la suite, quelques observations permettent de simplifier l'expression.

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

On observe que les clauses 1,2,3 et 5 ont en commun  $c$ . Pour que la formule soit vraie, il suffit que  $c$  soit vraie, quelque soit les valeur de  $a$  et de  $b$ . Par le calcul, on met en facteur  $c$  et cela donne :

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

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

Pour que la formule soit vraie, il faut que  $c$  soit vrai. Donc  $\neg c$  est faux. On peut donc simplifier en :

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

## E Du langage naturel à la logique propositionnelle

On considère l'énoncé suivant :

Si le professeur a bien dormi, l'examen est conforme aux exercices de TD. Si j'apprends mon cours, je réussis à trouver la solution des exercices de TD. Si je réussis à trouver la solution des exercices de TD et que l'examen est conforme aux exercices de TD, je réussis l'examen.

E1. Modéliser l'énoncé à l'aide de propositions.

**Solution :** On pose :

- $P$  le professeur a bien dormi.
- $E$  l'examen est conforme aux exercices de TD.
- $A$  j'apprends mon cours.
- $T$  je réussis à trouver la solution des exercices de TD.
- $R$  je réussis l'examen.

La modélisation est donc :

- $P \implies E$
- $A \implies T$
- $(T \wedge E) \implies R$

E2. Peut-on affirmer que, dans tous les cas, si le professeur dort bien, l'élève réussit à l'examen?

**Solution :** Non, naturellement. Comme le professeur dort bien, on a  $P$ . D'après notre première règle, on a donc :

$$(P \wedge (P \implies E)) \implies E$$

(Modus Ponens)

Par contre, on ne peut pas statuer sur  $T$ . Donc,  $R$  peut ne pas être vraie dans tous les cas car  $T$  peut être fausse. .