

# Colle 15 : Logique propositionnelle

25 mars 2022

## 1 EXPRESSIONS BOOLÉENNES ET ARBRE

---

On peut représenter une expression booléenne de la logique propositionnelle par un arbre où les noeuds internes sont étiquetés par "et", "ou" et "non" et les feuilles sont étiquetées par "0" et "1". Dans la suite, on pourra utiliser le type arbre suivant : `type 'a arbre = Noeud of 'a * 'a arbre list` et le type `string arbre` pour les expressions booléennes.

▷ **Question 1.** Ecrire une fonction `est_correct : string arbre -> bool` qui dit si l'arbre passé en argument a la syntaxe correcte d'une expression booléenne. ◀

▷ **Question 2.** Ecrire une fonction `evalue : string arbre -> bool` qui donne la valeur booléenne de l'expression représentée par l'arbre. ◀

▷ **Question 3.** Ecrire une fonction `negation : string arbre -> string arbre` qui transforme l'expression en sa négation. ◀

On appelle maintenant formule booléenne une expression booléenne où les feuilles sont étiquetées par des variables "Xi" et non plus par des valeurs booléennes. On utilisera alors le type suivant :

```
type expr = Var of int | Noeud of string* expr list;;
```

▷ **Question 4.** Ecrire une fonction `maxVar` qui prend en entrée une expression et qui renvoie le plus grand  $n$  tel que  $x_n$  apparait dans `expr`. ◀

▷ **Question 5.** Ecrire une fonction qui prend en entrée une expression booléenne et une interprétation des variables la composant sous forme d'un tableau et qui évalue l'expression sur cette interprétation. ◀

▷ **Question 6.** Ecrire une fonction qui prend en entrée une expression et dit si elle est satisfiable. ◀

## 2 RÉSOLUTION PROPOSITIONNELLE

---

Cette partie est issue d'un TP écrit par Jean Christophe Filiâtre

On se donne un ensemble de symboles de propositions atomiques. On définit alors l'ensemble des propositions de la manière suivante:

- VRAI et FAUX sont des propositions;
- si  $P$  est un symbole de proposition atomique alors  $P$  est une proposition;
- si  $P$  est une proposition alors  $\neg P$  est une proposition;
- si  $P$  et  $Q$  sont des propositions, alors  $P \rightarrow Q$ ,  $P \wedge Q$  et  $P \vee Q$  sont des propositions.

On représentera les propositions par le type Caml suivant:

```

type proposition_atomique = string

type proposition =
|Atome of proposition_atomique
|Vrai
|Faux
|Et of proposition*proposition
|Ou of proposition*proposition
|Imp of proposition*proposition
|Neg of proposition

```

**Exemple.** La proposition  $A \rightarrow (A \rightarrow B) \rightarrow B$  est représentée par le terme suivant:

```

let A= Atome "A" and B=Atome "B" in Imp(A, Imp(Imp(A, B), B)) ;;

```

On sait que la logique propositionnelle est décidable; on peut par exemple examiner les tables de vérité des propositions (Post, 1921). On se propose ici d'implanter un autre algorithme de décision: la résolution propositionnelle.

## 2.1 FORME NORMALE CONJONCTIVE

**Définition 1 (Clauses et formes clauseuses)** Une clause est une proposition de la forme  $A_1 \vee A_2 \vee \dots \vee A_n$  où  $A_i$  est une proposition atomique, la négation d'une proposition atomique, FAUX ou  $\neg$  FAUX. Une forme clauseuse est une proposition de la forme  $C_1 \wedge C_2 \wedge \dots \wedge C_n$  où  $C_i$  est une clause.

Pour des raisons pratiques, on représentera les clauses et les formes clauseuses par des listes :

```

type clause = proposition list
type forme_clauseuse = clause list

```

▷ **Question 7.** Ecrire une fonction `union : 'a list -> 'a list -> 'a list` effectuant la concaténation de deux listes en supprimant les répétitions. ◁

Soient  $C^1 = C_1^1 \vee \dots \vee C_1^m$  et  $C^2 = C_2^1 \vee \dots \vee C_2^m$  deux clauses. On note  $C_1 \bar{\vee} C_2$  l' "union" de ces deux clauses, c'est-à-dire la disjonction  $C_1^1 \vee \dots \vee C_1^m \vee C_2^1 \vee \dots \vee C_2^m$  dans laquelle on a supprimé toute répétition. De même, si  $F_1$  et  $F_2$  sont deux formes clauseuses, on note  $F_1 \bar{\wedge} F_2$  leur "union", c'est-à-dire la conjonction de leurs clauses parmi lesquelles on a supprimé toute répétition.

Les deux opérations  $\bar{\vee}$  et  $\bar{\wedge}$  sont donc tout simplement réalisées par la fonction `union`.

Soit  $A = A_1 \wedge \dots \wedge A_n$  et  $B = B_1 \wedge \dots \wedge B_m$  deux formes clauseuses (les  $A_i$  et  $B_i$  étant des clauses). On note  $A \otimes B$  la forme clauseuse définie par

$$A \otimes B = \bigwedge_{i,j} \bar{\wedge}_i \bar{\vee}_j B_j$$

▷ **Question 8.** Ecrire une fonction `prod: forme_clauseuse -> forme_clauseuse -> forme_clauseuse` implantant l'opération  $\otimes$ . ◁

**Définition 2 (Mise en forme clauseuse)** Soit  $P$  une proposition. On définit sa forme clauseuse  $\mathcal{C}(P)$ , ou forme normale conjonctive, de la manière suivante:

- Si  $P$  est une proposition atomique, la négation d'une proposition atomique, FAUX ou  $\neg$ FAUX, alors  $\mathcal{C}(P) = P$ ;
- $\mathcal{C}(A \wedge B) = \mathcal{C}(A) \bar{\wedge} \mathcal{C}(B)$ ;
- $\mathcal{C}(A \vee B) = \mathcal{C}(A) \otimes \mathcal{C}(B)$ ;
- $\mathcal{C}(A \rightarrow B) = \mathcal{C}(\neg A) \otimes \mathcal{C}(B)$ ;
- $\mathcal{C}(\text{VRAI}) = \neg \text{FAUX}$  et  $\mathcal{C}(\neg \text{VRAI}) = \text{FAUX}$
- $\mathcal{C}(\neg(A \wedge B)) = \mathcal{C}(\neg A) \otimes \mathcal{C}(\neg B)$ ;
- $\mathcal{C}(\neg(A \vee B)) = \mathcal{C}(\neg A) \bar{\wedge} \mathcal{C}(\neg B)$ ;
- $\mathcal{C}(\neg(A \rightarrow B)) = \mathcal{C}(A) \bar{\wedge} \mathcal{C}(\neg B)$ ;
- $\mathcal{C}(\neg \neg A) = \mathcal{C}(A)$

▷ **Question 9.** Ecrire une fonction `fnc: proposition -> forme_clauseuse` qui calcule la forme clauseuse  $\mathcal{C}(P)$  d'une proposition  $P$  passée en argument.

Applications: Calculer la forme clausale des propositions suivantes:  $(A \wedge B) \rightarrow A$ ,  $A \rightarrow (A \wedge B)$  et  $A \rightarrow (A \rightarrow B) \rightarrow B$ . ◀

Proposition. Soit  $P$  une proposition.  $P$  est vraie si et seulement si  $(P)$  est vraie.

## 2.2 MÉTHODE DE RÉOLUTION

Définition 3 (Règle de résolution) De deux clauses

$$P = B_1 \vee \dots \vee B_k \vee A \vee B_{k+1} \vee \dots \vee B_n$$
$$Q = C_1 \vee \dots \vee C_l \vee \neg A \vee C_{l+1} \vee \dots \vee C_p$$

on peut déduire la clause

$$R = (B_1 \vee \dots \vee B_k \vee B_{k+1} \vee \dots \vee B_n) \vee (C_1 \vee \dots \vee C_l \vee C_{l+1} \vee \dots \vee C_p)$$

Lorsque cette clause est vide on pose  $R = \text{FAUX}$ .

Soient deux clauses  $A$  et  $B$ . La règle de résolution peut s'appliquer, ou ne pas s'appliquer, aux clauses  $A$  et  $B$ . Le cas échéant, elle peut parfois s'appliquer de plusieurs façons différentes. On représentera donc le résultat d'une résolution de  $A$  et  $B$  par la liste des clauses que l'on peut obtenir à partir de  $A$  et  $B$  par résolution.

Exemple: Si  $A = P \vee \neg Q$  et  $B = Q \vee R \vee \neg P$  alors le résultat est la liste de deux clauses:

$$[P \vee R \vee \neg P; \neg Q \vee Q \vee R]$$

▷ **Question 10.** Ecrire une fonction `enleve`: `'a list -> 'a -> 'a list` qui enlève un élément d'une liste. ◀

▷ **Question 11.** Ecrire une fonction `resolutions`: `clause -> clause -> clause list` qui calcule toutes les résolutions possibles de deux clauses. ◀

▷ **Question 12.** Ecrire une fonction `succes` : `clause list -> bool` qui teste si la clause FAUX apparaît dans une liste de clauses (test de succès). ◀

Proposition (Robinson, 1965). Soit  $P$  une proposition et  $C_1 \wedge \dots \wedge C_n$  la forme clausale de  $\neg P$ . Alors  $P$  est vraie si et seulement si FAUX est dérivable par résolution propositionnelle à partir des clauses  $C_1, \dots, C_n$ .

▷ **Question 13.** En déduire une fonction `decision`: `proposition -> bool` qui, appliquée à  $P$ , renvoie true si  $P$  est vraie et faux sinon.

Indication: Etant donnée une liste de clauses  $l = c_1, \dots, c_n$ , on pourra dans un premier temps calculer toutes les paires  $\{x, y\}$  d'éléments distincts de  $l$ . Ensuite, pour chaque paire  $\{x, y\}$  on pourra calculer avec `resolutions` la liste de toutes les résolutions de  $x$  et  $y$ . Si cette liste contient la clause réduite à FAUX, alors on retourne true. Sinon, pour chaque résolution  $r$ , on rappelle récursivement la procédure de décisions sur  $(l \setminus \{x, y\}) \cup \{r\}$ .

◀

▷ **Question 14.** Application: le Club Ecosais. Il existe en Ecosse un club très fermé qui obéit aux règles suivantes :

- Tout membre non-écossais porte des chaussettes rouges;
- Tout membre porte un kilt ou ne porte pas de chaussettes rouges;
- Les membres mariés ne sortent pas le dimanche;
- Un membre sort le dimanche si et seulement s'il est écossais;
- Tout membre qui porte un kilt est écossais et marié;
- Tout membre écossais porte un kilt.

On souhaite prouver que ce club est si fermé qu'il ne peut accepter personne!

Donner une proposition  $P$  telle que  $P$  est vraie si et seulement si aucun membre ne peut être accepté dans ce club.

Appliquer la fonction `decision` ci-dessus à cette proposition pour montrer qu'aucun membre ne peut être accepté dans ce club. ◀