Le calcul propositionnel

Syntaxe du calcul propositionnel

Soit $\mathcal{R}$ en ensemble dénombrable de lettres dites propositionnelles.

Définition : L'ensemble $\mathcal{F}_{prop}$ de formules du calcul propositionnel, est le plus petit ensemble t.q.
- $\mathcal{R} \subseteq \mathcal{F}_{prop}$
- Si $A \in \mathcal{F}_{prop}$, alors $\neg A \in \mathcal{F}_{prop}$.
- Si $A, B \in \mathcal{F}_{prop}$, alors $\vee(A,B), \wedge(A,B), \rightarrow(A,B) \in \mathcal{F}_{prop}$.

Exemple :	$\neg(p)$	$\vee(p,p)$	$\rightarrow(\wedge(p,q), \neg(r))$
Notation simplifiée :	$\neg p$	$p \vee p$	$(p \wedge q) \rightarrow \neg r$

Remarque :
- Quelques fois on écrira $\#$ pour $\vee$, $\wedge$ ou $\rightarrow$.
- $\mathcal{F}_{prop}$ est un ensemble inductif.

$\mathcal{SF}(A)$: sous-formules d'une formule A

- Si A est une lettre p, $\mathcal{SF}(A) = \{p\}$.
- Si A est $\neg(B)$, $\mathcal{SF}(A) = \{\neg B\} \cup \mathcal{SF}(B)$.
- Si A est $\#(B,C)$, $\mathcal{SF}(A) = \{\#(B,C)\} \cup \mathcal{SF}(B) \cup \mathcal{SF}(C)$.

N.B. : la relation de sous-formule définit naturellement un ordre (partiel) entre les sous-formules de A. Cet ordre peut être completé en un ordre total de telle sorte que les variables propositionnelles apparaissent au tout debut.

Sémantique de la logique propositionnelle

Étant donnée une valeur de l'ensemble $\mathbf{BOOL} = \{\mathbf{V}, \mathbf{F}\}$ pour chaque lettre propositionnelle, on veut établir la valeur d'une formule propositionnelle A.
- Fixer une interprétation qui donne $\mathbf{V}$ ou $\mathbf{F}$ à chaque lettre propositionnelle.
- Définir la fonction booléenne unaire $\mathcal{FB}_\neg : \mathbf{BOOL} \rightarrow \mathbf{BOOL}$ et les fonctions booléennes binaires $\mathcal{FB}_\vee, \mathcal{FB}_\wedge, \mathcal{FB}_\rightarrow : \mathbf{BOOL}^2 \rightarrow \mathbf{BOOL}$.
- Construire la valeur de vérité de la formule A.

La fonction booléenne unaire

$$\begin{array}{lcl} \mathcal{FB}_\neg(\mathbf{V}) & = & \mathbf{F} \\ \mathcal{FB}_\neg(\mathbf{F}) & = & \mathbf{V} \end{array}$$

Les fonctions booléennes binaires

$$\begin{array}{lcl} \mathcal{FB}_\vee(\mathbf{V},\mathbf{V}) & = & \mathbf{V} \\ \mathcal{FB}_\vee(\mathbf{V},\mathbf{F}) & = & \mathbf{V} \\ \mathcal{FB}_\vee(\mathbf{F},\mathbf{V}) & = & \mathbf{V} \\ \mathcal{FB}_\vee(\mathbf{F},\mathbf{F}) & = & \mathbf{F} \end{array} \qquad \begin{array}{lcl} \mathcal{FB}_\wedge(\mathbf{V},\mathbf{V}) & = & \mathbf{V} \\ \mathcal{FB}_\wedge(\mathbf{V},\mathbf{F}) & = & \mathbf{F} \\ \mathcal{FB}_\wedge(\mathbf{F},\mathbf{V}) & = & \mathbf{F} \\ \mathcal{FB}_\wedge(\mathbf{F},\mathbf{F}) & = & \mathbf{F} \end{array}$$

$$\begin{array}{lcl} \mathcal{FB}_\rightarrow(\mathbf{V},\mathbf{V}) & = & \mathbf{V} \\ \mathcal{FB}_\rightarrow(\mathbf{V},\mathbf{F}) & = & \mathbf{F} \\ \mathcal{FB}_\rightarrow(\mathbf{F},\mathbf{V}) & = & \mathbf{V} \\ \mathcal{FB}_\rightarrow(\mathbf{F},\mathbf{F}) & = & \mathbf{V} \end{array}$$

Valeur de vérité d'une formule A par rapport à une interprétation I

- Si A est une lettre p, $[A]_I = I(p)$.
- Si A est $\neg(B)$, $[A]_I = \mathcal{FB}_\neg([B]_I)$.
- Si A est $\#(B,C)$, $[A]_I = \mathcal{FB}_\#([B]_I, [C]_I)$.

Exercice : Soit I l'intéprétation $I(p) = \mathbf{V}$, $I(q) = \mathbf{F}$. Calculer la valeur de vérité de la formule $(p \vee q) \rightarrow \neg(q \wedge q)$ par rapport à I.

Satisfaire et falsifier une formule

Soit I une intéprétation, A une formule et Δ un ensemble de formules.

Définition :
I satisfait une formule A si $[A]_I = \mathbf{V}$
I falsifie une formule A si $[A]_I = \mathbf{F}$.
I satisfait un ensemble de formules Δ si I satisfait toute formule de Δ.
I falsifie un ensemble de formules Δ ssi il existe au moins une formule A dans Δ telle que $[A]_I = \mathbf{F}$.

Formules satisfaisables, contradictoires, valides

Définition : Une formule A est satisfaisable s'il existe au moins une interprétation I qui satisfait A. Un ensemble de formules Δ est satisfaisable s'il existe au moins une interprétation I telle que I satisfait Δ.

Définition : Une formule A est contradictoire si elle n'est pas satisfaisable. Un ensemble de formules Δ est contradictoire si il n'est pas satisfaisable.

Conséquence logique et validité

Définition : Une formule A est valide si toute interprétation satisfait A. Un ensemble de formules Δ est valide si toute formule de Δ est valide.

Définition : Une formule A est conséquence logique d'un ensemble de formules Δ, noté $\Delta \models A$, si toute interprétation qui satisfait Δ satisfait aussi A.

Tables de vérité

À quoi ça sert ? Méthode pour raisonner sur les modèles de formules propositionnelles.
Comment ça marche ? Soit A une formule ayant comme lettres propositionnelles l'ensemble $\{p_1, \ldots, p_n\}$ et dont l'ensemble de sous-formules est $\{A_1, \ldots, A_k\}$.

1. Construire une table où chaque colonne est étiquetée par les sous-formules A_j de A, en ordre de sousformule.
2. Pour chaque ligne m de la table :
   (a) Donner une interprétation I_m aux lettres $p_1, \ldots, p_n$.
   (b) Calculer les valeurs $[A_1]_{I_m}, \ldots, [A_k]_{I_m}$

Comment lire une table de vérité ?

- Si la colonne étiquetée par la formule A (qui est une sous-formule de A) ne contient que de $\mathbf{V}$, alors A est valide.
- Si la colonne de la formule A ne contient que de $\mathbf{F}$, alors A est contradictoire.
- Sinon, l'interprétation qui rends $\mathbf{V}$ la colonne de la formule A satisfait A et l'interprétation qui rends $\mathbf{F}$ la colonne de la formule A falsifie A.

Équivalence logique

Définition : Deux formules A et B sont équivalentes, noté $A \equiv B$, ssi $\{A\} \models B$ et $\{B\} \models A$.

Remarque : $A \equiv B$ ssi $(A \rightarrow B) \wedge (B \rightarrow A)$ est valide.

Encore quelques exemples

(Associativité)	$(A \vee B) \vee C$	$\equiv$	$A \vee (B \vee C)$
	$(A \wedge B) \wedge C$	$\equiv$	$A \wedge (B \wedge C)$
(Commutativité)	$A \vee B$	$\equiv$	$B \vee A$
	$A \wedge B$	$\equiv$	$B \wedge A$
(Idempotence)	$A \vee A$	$\equiv$	A
	$A \wedge A$	$\equiv$	A
(Lois de De Morgan)	$\neg(A \wedge B)$	$\equiv$	$\neg A \vee \neg B$
	$\neg(A \vee B)$	$\equiv$	$\neg A \wedge \neg B$
(Distributivité)	$A \vee (B \wedge C)$	$\equiv$	$(A \vee B) \wedge (A \vee C)$
	$A \wedge (B \vee C)$	$\equiv$	$(A \wedge B) \vee (A \wedge C)$
(Loi de la double négation)	$\neg\neg A$	$\equiv$	A
(Définissabilité de $\rightarrow$)	$A \rightarrow B$	$\equiv$	$\neg A \vee B$

Remarques

1. $\{E_1, \ldots, E_n\} \models A$ ssi la formule $E_1 \wedge \ldots \wedge E_n \rightarrow A$ est valide.
2. Si Δ est satisfaisable et $\Gamma \subseteq \Delta$, alors Γ est satisfaisable.
3. L'ensemble vide est satisfaisable.
4. L'ensemble de toutes les formules est contradictoire.
5. Si Δ est satisfaisable, alors Δ est finiment satisfaisable.
6. Si Γ est contradictoire et $\Gamma \subseteq \Delta$, alors Δ est contradictoire.
7. Toute formule est conséquence logique d'un ensemble insatisfaisable de formules.
8. Toute formule valide est conséquence logique d'un ensemble quelconque de formules, en particulier de l'ensemble vide.
9. A est valide ssi $\neg A$ est insatisfaisable.
10. $\Delta \models A$ ssi $\Delta \cup \{\neg A\}$ est insatisfaisable.

Définissabilité

Définition : Soit A une formule avec n lettres propositionnelles $p_1, \ldots, p_n$ ($n \geq 0$). La fonction booléenne n-aire qui réalise la formule A est une fonction $\mathcal{FB}_A : \mathbf{BOOL}^n \to \mathbf{BOOL}$ telle que

$$\mathcal{FB}_A(v_1, \ldots, v_n) = [A]_I \text{ si } I(p_i) = v_i$$

pour toute interprétation I de $p_1, \ldots, p_n$.

Ensemble de connecteurs complet

Définition : Un ensemble $\mathcal{C}$ de connecteurs est complet ssi pour toute fonction booléenne f il existe une formule A contenant uniquement des connecteurs de $\mathcal{C}$ telle que f réalise A.

Théorème : L'ensemble $\{\neg, \wedge, \vee\}$ est complet.

Intuition de la preuve

v_1	v_2	v_3	$f(v_1, v_2, v_3)$
V	**V**	**V**	**V**
V	**V**	**F**	**F**
V	**F**	**V**	**F**
V	**F**	**F**	**V**
F	**V**	**V**	**V**
F	**V**	**F**	**V**
F	**F**	**V**	**F**
F	**F**	**F**	**V**

Le calcul propositionnel en OCAML

```
# type fprop = L of int | Neg of fprop |
               Ou of fprop * fprop |
               Et of fprop * fprop |
               Impl of fprop * fprop;;

# let rec sf a = match a with
      L(x)     -> [L(x)]
   | Neg(x)    -> Neg(x) :: sf x
   | Ou(x,y)   -> Ou(x,y) :: List.append (sf x) (sf y)
```



```
   | Et(x,y)   -> Et(x,y) :: List.append (sf x) (sf y)
   | Impl(x,y) -> Impl(x,y) :: List.append (sf x) (sf y);;
val sf : fprop -> fprop list = <fun>


# let a1 = Et(Neg(L(3)),Ou(L(2),L(1)));;
val a1 : fprop = Et (Neg (L 3), Ou (L 2, L 1))

# sf a1;;
- : fprop list =
[Et (Neg (L 3), Ou (L 2, L 1));
 Neg (L 3);
 L 3;
 Ou (L 2, L 1);
 L 2;
 L 1]
```

Une interprétation

Une interprétation est une liste de couples de la forme (`lettre, bool`).

```
# let i1 = [(L 1,true); (L 2,false); (L 3,true)];;
val i1 : (fprop * bool) list =
         [(L 1, true); (L 2, false); (L 3, true)]
```

L'interprétation d'une lettre propositionnelle

```
# let rec int_lettre l i = match i with
    []       -> failwith "  erreur"
  | (m,v)::r -> if l=m then v else int_lettre l r;;
val int_lettre : 'a -> ('a * 'b) list -> 'b = <fun>

# int_lettre (L 2) i1;;
- : bool = false
```

Les fonctions booleenes

```
#let fb_neg b = match b with
    true  -> false
  | false -> true;;
val fb_neg : bool -> bool = <fun>
```



```
#let fb_ou(a,b) = match (a,b) with
    (true,_) -> true
  | (_,true) -> true
  | _        -> false;;
val fb_ou : bool * bool -> bool = <fun>


#let fb_et(a,b) = match (a,b) with
    (true,true) -> true
  | _           -> false;;
val fb_et : bool * bool -> bool = <fun>

#let fb_impl(a,b) = match (a,b) with
    (true,false) -> false
  | _            -> true;;
val fb_impl : bool * bool -> bool = <fun>
```

Valeur de verité

```
# let rec valeur a i = match a with
    L(n)      -> int_lettre (L(n)) i
  | Neg(x)    -> fb_neg (valeur x i)
  | Et(x,y)   -> fb_et (valeur x i,valeur y i)
  | Ou(x,y)   -> fb_ou (valeur x i,valeur y i)
  | Impl(x,y) -> fb_impl (valeur x i,valeur y i);;
val valeur : fprop -> (fprop * bool) list -> bool = <fun>

# valeur a1 i1;;
- : bool = false
```

Toutes les interprétations

On utilise une fonction auxiliare `ajouter e l` qui ajoute l'élément e au début de chaque liste de `l`.

```
# let rec genere_int l = match l with
    []   -> [[]]
  | p::r -> let t = genere_int r in
            concat (ajouter (p,true) t) (ajouter (p,false) t);;
```



```
val genere_int : 'a list -> ('a * bool) list list = <fun>


# genere_int [L 1; L 2; L 3];;
- : (fprop * bool) list list =
[[(L 1, true); (L 2, true); (L 3, true)];
 [(L 1, true); (L 2, true); (L 3, false)];
 [(L 1, true); (L 2, false); (L 3, true)];
 [(L 1, true); (L 2, false); (L 3, false)];
 [(L 1, false); (L 2, true); (L 3, true)];
 [(L 1, false); (L 2, true); (L 3, false)];
 [(L 1, false); (L 2, false); (L 3, true)];
 [(L 1, false); (L 2, false); (L 3, false)]]
```

Toutes les lettres d'une formule

On utilise une fonction auxiliare `union l1 l2` qui renvoie l'union de deux listes `l1` et `l2`.

```
# let rec lettres a = match a with
      L(x)     -> [L(x)]
    | Neg(x)    -> lettres x
    | Ou(x,y)   -> union (lettres x) (lettres y)
    | Et(x,y)   -> union (lettres x) (lettres y)
    | Impl(x,y) -> union (lettres x) (lettres y);;
val lettres : fprop -> fprop list = <fun>

#  lettres a1;;
- : fprop list = [L 3; L 2; L 1]
```

Validité d'une formule

```
# let rec valide a =
    let l = lettres a in
    let i = genere_int l in
    let r = map (valeur a) i in
            est_true r;;

# valide a1;;
- : bool = false
# let a2 = Ou(L 1, Neg(L 1)) ;;
```

```
val a2 : fprop = Ou (L 1, Neg (L 1))
# valide a2;;
- : bool = true
```