C. Charignon

Table des matières

Première partie
Cours

On étudie la logique dans un cadre très strict : on n'étudie pas les quantificateurs.

1 Calcul booléen

1.1 Notations

Les booléen « Vrai » et « Faux » sont notés « $\top$ » et « $\bot$ », respectivement. Pour ce chapitre, on notera $\mathcal{B} = \{\top, \bot\}$ l'ensemble des booléens.

Cet ensemble est muni de deux lois de composition internes : « et », qu'on notera $\wedge$ et « ou » qu'on note $\vee$. En outre, nous connaissons une fonction de $\mathcal{B}$ dans $\mathcal{B}$: la fonction « non » qu'on notera $\neg$.

L'ordre de priorité entre ces opérations est :

1. la négation,
2. la conjonction,
3. la disjonction.

Ainsi $\neg a \wedge b \vee c$ signifie $\big((\neg a) \wedge b\big) \vee c$.

Remarque : Comme $\mathcal{B}$ est de cardinal 2, il existe 4 fonctions de $\mathcal{B}$ dans $\mathcal{B}$. Outre $\neg$ il y a identité (parfois appelée la fonction « oui ») et les deux constantes.

1.2 Règles de calcul

Proposition 1.1.
- *La lci $\wedge$ est associative, commutative et admet $\bot$ comme élément neutre.*
- *La lci $\vee$ est associative, commutative et admet $\top$ comme élément neutre.*
- *La lci $\vee$ est distributive sur $\wedge$.*
- *La lci $\wedge$ est distributive sur $\vee$.*
- *(loi de De Morgan) Pour tout $(a, b) \in \mathcal{B}^2$, on a $\neg(a \wedge b) = \neg a \vee \neg b$ et $\neg(a \vee v) = \neg a \wedge \neg b$.*
- *(tiers exclus) $\forall a \in \mathcal{B}$, $a \vee \neg a$.*

Le plus simple pour démontrer ces relations est de dresser les tables de vérités. Par exemple pour $\neg(a \wedge b) = \neg a \vee \neg b$: Voici la table de vérité de $\neg(a \wedge b)$:

a	b	$a \wedge b$	$\neg(a \wedge b)$
V	V	V	F
V	F	F	V
F	V	F	V
F	F	F	V

Et par ailleurs celle de $\neg a \vee \neg b$:

a	b	$\neg a$	$\neg b$	$\neg a \vee \neg b$
V	V	F	F	F
V	F	F	V	V
F	V	V	F	V
F	F	V	V	V

Nous vérifions bien que quelle que soit la valeur de a et de b, $(a \wedge b) = \neg a \vee \neg b$.

La méthode de la table de vérité utilisée ci-dessus est pratique pour montrer l'égalité entre deux expressions booléenne, à condition que le nombre de variables employé soit petit (trois, ce qui fait huit lignes dans la table, au maximum quatre ce qui ferait seize lignes).

1.3 Algèbre de Boole

Nous avons vu que les opérateurs $\wedge$ et $\vee$ sont commutatifs et associatifs, et admettent un neutre. En outre, $\wedge$ est distributif par rapport à $\vee$. Ces ressemblances avec les opérations usuelles $+$ et $\times$ justifient l'emploi des notations plus simples suivantes :

- $\top$ pourra aussi être noté 1, et $\bot$ noté 0 ;
- $\wedge$ pourra aussi être noté noté $\cdot$, et $\vee$ noté $+$;
- $\neg a$ pourra aussi être noté $\overline{a}$.

Ces notations seront plus pratiques dans les calculs.

Remarque : En mémoire, le booléen $\top$ est souvent enregistré de la même manière que l'entier 1, et $\bot$ comme l'entier 0. Et dans de nombreux langage (notamment SQL), on peut indifféremment taper « 1 » ou « true ».

N.B. À première vue, on obtient une algèbre, semblable à $\mathbb{Z}/2\mathbb{Z}$. Il y a cependant des différences :

1. $1+1=1$ alors que dans $\mathbb{Z}/2\mathbb{Z}$, $1+1=0$.
2. $(\mathcal{B},+,\cdot)$ n'est *pas* un anneau. En effet 1 n'a pas d'opposé (puisque $1+0=1$ et $1+1=1$, il n'existe pas de $x\in\mathcal{B}$ tel que $1+x=0$).

En revanche, en notant $\oplus$ le ou exclusif (XOR en anglais), $(\mathcal{B},\oplus,\cdot)$ est un corps isomorphe à $\big(\mathbb{Z}/2\mathbb{Z},+,\cdot\big)$.
Au passage, on peut remarquer que $\oplus$ est la même chose que $\neq$ sur l'ensemble $\mathcal{B}$.

1.4 Autres connecteurs logiques

Remarque : Il y a 2^4 tables de vérité de format 2×2, autrement dit 16 opérateurs logiques binaires.

Parmi eux, 8 sont commutatifs. Parmi eux, 2 sont constants. On connaît les opérateurs $\wedge$ (et) et $\vee$ (ou), et on a mentionné « ou exclusif » $\oplus$.

1.4.1 Implication et équivalence

Définition 1.2.

- *Le connecteur logique $\Rightarrow$ est la lci qui a la table d'opération suivante :*

$\Rightarrow$	V	F
V	V	F
F	V	V

.

- *Le connecteur logique $\Leftrightarrow$ est la lci qui a la table d'opération suivante :*

$\Leftrightarrow$	V	F
V	V	F
F	F	V

.

Remarque : En fait, $\Leftrightarrow$ n'est autre que $=$ lorsqu'on l'utilise dans $\mathcal{B}$.

En mathématique, on utilise les implications uniquement pour des prédicats et précédées d'un quantificateur universel. Ainsi l'énoncé d'un théorème de mathématiques est généralement de la forme suivante : on dispose d'un ensemble E et de deux prédicats P et Q sur E, et le théorème affirme que :

$$\forall x\in E,\quad P(x)\Rightarrow Q(x).$$

Cela signifie que pour tout $x\in E$, le booléen $\big(P(x)\Rightarrow Q(x)\big)$ doit être vrai, et au vu de la table d'opération de $\Rightarrow$, cela signifie qu'à chaque fois que $P(x)$ est vrai, $Q(x)$ doit l'être aussi. Par contre lorsque $P(x)$ est faux, $Q(x)$ peut être faux ou vrai indifféremment.

Proposition 1.3. *Pour tout $(a,b)\in\mathcal{B}^2$,*

- $(a\Leftrightarrow b)=(a\Rightarrow b)\wedge(b\Rightarrow a)$.
- $(a\Rightarrow b)=(\neg a\vee b)$.

Remarques :

- Avec le formalisme qui sera développé dans la suite du cours, on notera $\equiv$ plutôt que $=$ entre deux formules logiques.

- En conséquence, on obtient $\neg(a \Rightarrow b) = a \wedge \neg b$, formule vue en math en première année. Ainsi, pour montrer qu'un théorème de type « $\forall x \in E,\quad P(x) \Rightarrow Q(x)$ », on doit prouver « $\exists x \in E$ tq $\quad P(x) \wedge \neg Q(x)$ », ce qui s'appelle trouver un « contre-exemple ».
- Toutes les formules logiques peuvent être écrites uniquement à l'aide de $\neg$, $\wedge$ et $\vee$, ce qui explique qu'on n'a pas parlé de $\Rightarrow$ dans un premier temps, et qu'il n'est pas implanté dans Caml.

Exercice : Vérifier que $\forall (a,b) \in \mathcal{B}$, on a $(a \Rightarrow b) \vee (b \Rightarrow a)$.
Remarque : En reprenant E, P, Q comme avant, on peut donc affirmer :

$$\forall x \in E,\ \big(P(x) \Rightarrow Q(x)\big) \vee \big(Q(x) \Rightarrow P(x)\big)$$

formule qui n'a strictement aucun intérêt en mathématiques.

Par contre, la formule suivante est *fausse* :

$$\big(\forall x \in E,\ P(x) \Rightarrow Q(x)\big) \vee \big(\forall x \in E, Q(x) \Rightarrow P(x)\big)\,.$$

Cette formule signifierait que pour tous prédicats P et Q sur E, l'implication $\forall x \in E,\ P(x) \Rightarrow Q(x)$ ou sa réciproque est toujours vraie. En d'autre termes que si une implication est fausse alors sa réciproque est vraie !

En gros, la quantificateur $\forall$ n'est pas distributif sur le connecteur logique $\vee$.

1.4.2 Non-et et Non-ou

Deux autres opérateurs sont fréquemment utilisés en informatique : non-et (NAND, parfois noté $|$ ou $\overline{\wedge}$) et non-ou (NOR).

Deux remarques expliquent l'intérêt de ces opérateurs :

- tous les autres opérateurs peuvent être décrit à l'aide de NAND (**cf exercice :** 9) ;
- ces deux opérateurs nécessitent 4 transistors pour être implantés dans un circuit électronique alors que $\wedge$ et $\vee$ en nécessitent 5.

2 Formules propositionnelle

Dans ce chapitre, nous définissons un nouvel objet, appelé « formule logique ».

2.1 Définition du type Caml

Nous définissons un type Caml pour représenter les formules logiques :

```
type formule =
 Vrai
| Faux
| Var of string
| Non of formule
| Et of formule*formule
| Ou of formule*formule
;;
```

Cependant, pour pour se laisser la possibilité ultérieure de rajouter de nouveaux opérateurs, on peut préférer :

```
type formule =
   Cte of bool
 | Var of string
 | Non of formule
 | Op of (bool->bool->bool)*formule*formule
;;
```

On définit alors les quatre opérateurs ainsi :

```
let et x y = x && y;;
let ou x y  = x || y;;
let implique x y = y || not x;;
let equi x y = (x&& y) || ( not x && not y);;
```

Par exemple la formule $\neg a \vee b \wedge c$ sera représentée par :

```
let formuleExemple =
 Op(et,
      Non(
          Op( ou,
                Var "a",
                Var "b")
         ),
    Var "c"
);;
```

Remarques :

- Même si nous n'avons pas utilisé un constructeur appelé `Noeud`, il s'agit bien d'un arbre, et nous pourrons représenter comme telles les formules à l'écrit.
- Une méthode pratique pour taper une formule logique serait de la taper en notation polonaise inversée et d'écrire un petit programme pour transformer une formule écrite en notation polonaise inversée en une formule donnée par un arbre comme ci-dessus. Il suffit d'employer la méthode vue dans le chapitre sur les piles en MPSI.

  On peut aussi écrire un programme permettant de lire une formule écrite en notation traditionnelles (avec des parenthèses), mais ce sera plus compliqué.

2.2 Définition mathématique

Maintenant, voici une définition formelle d'une formule logique, correspondant au type Caml écrit ci-dessus.
On fixe un ensemble $\mathcal{A}$ (pour « alphabet ») dont les éléments seront appelés des « variables propositionnelles ».

Définition 2.1. *Nous définissons l'ensemble des formules propositionnelles récursivement ainsi :*

- *(cas de base : variables propositionnelles) Pour tout $x \in \mathcal{A}$, x est une formule propositionnelle ;*
- *(deuxième cas de base : les constantes) Pour tout $b \in \mathcal{B}$, b est une formule propositionnelle ;*
- *Pour toute formule f, $(\neg f)$ est une formule propositionnelle ;*
- *Pour toutes formules f et g, $(f \wedge g)$ et $(f \vee g)$ sont des formules propositionnelles.*

On notera $\mathcal{F}(\mathcal{A})$ l'ensemble des formules sur $\mathcal{A}$.

La formule $\neg x$ se lit « non x », $(x \wedge y)$ se lit « x et y » , et $(x \vee y)$ se lit « x ou y ».

On notera que la définition mathématique suit très précisément la définition du type Caml. De plus pour l'instant, une formule n'est qu'une suite de symbole : aucun sens ne lui est attaché. De même que le type Caml est parfaitement inutile tant que nous n'avont créé aucune fonction l'utilisant.

2.3 Sémantique

Définition Larousse du mot « sémantique » :

- Étude du sens des unités linguistiques et de leurs combinaisons.
- Étude des propositions d'une théorie déductive du point de vue de leur vérité ou de leur fausseté.
- Aspect de la logique qui traite de l'interprétation et de la signification des systèmes formels, par opposition à la syntaxe, entendue comme l'étude des relations formelles entre formules de tels systèmes.

Bref, maintenant que nous avons défini formellement les formules logiques, il s'agit de leur donner du sens.

Définition 2.2. *Un « contexte », ou « distribution de vérité », ou « valuation », sur $\mathcal{A}$ est une fonction de $\mathcal{A}$ dans* $\{\top, \bot\}$

En d'autre termes, définir un contexte revient à choisir pour chaque variable une valeur dans $\mathcal{B}$.
Une fois fixé un contexte, on peut évaluer si une formule est vraie ou fausse.
On commence par le point de vue « Caml ». Nous donnons ci-après une définition formelle de l'évaluation d'une formule, mais en premier, nous allons nous baser sur l'idée intuitive pour écrire une fonction Caml qui prend en entrée une formule et un contexte et qui indique si la formule est vérifiée pour ce contexte.
En Caml, on pourra représenter un contexte à l'aide d'un dictionnaire. Or, nous connaissons trois manières d'implanter un dictionnaire :

- liste d'adjacence (simple mais peu efficace) ;
- table de hachage (pour un dictionnaire mutable) ;
- ABR (pour un dictionnaire persistant).

Pour taper de petits exemples à la main, le plus simple est la liste d'association.

```
(* Ci-dessous on a pris la manière basique : listes d'association.*)
type contexte = (string*bool) list;;
(* Pour utiliser une autre implémentation d'un contexte, changer simplement la fonction
    ↪ valeur_de ci-dessous. *)
let valeur_de x= List.assoc x;;

let contexteExemple=[ ("a",true); ("b", true); ("c", false)];;
valeur_de "a" contexteExemple;;


let rec evaluation formule (contexte:contexte) =
  (* contexte est un dictionnaire associant
  à chaque variable utilisée dans formule
  une valeur de vérité. *)
  match formule with
    | Vrai   -> true
    | Faux  -> false
    | Variable x  -> valeur_de x contexte
    | Non(f)  -> not (evaluation f  contexte)
    | OpBinaire(f1, op, f2) ->
                    op (evaluation f1  contexte)
                       (evaluation f2  contexte)
;;

evaluation fExemple contexteExemple;;
```

Passons maintenant à une définition formelle de l'évaluation. On notera $\mathcal{E}_c(f)$ le résultat de l'évaluation de la formule f dans le contexte c.

Définition 2.3. *Soit c un contexte sur $\mathcal{A}$. On définit l'évaluation selon c, qu'on note $\mathcal{E}_c$ ainsi :*

- $\mathcal{E}_c(Vrai) = \top$
- $\mathcal{E} - c(Faux) = \bot$
- $\forall x \in \mathcal{A}$, $\mathcal{E}_c(x) = c(x)$
- $\forall f \in \mathcal{F}$, $\mathcal{E}_c(\neg f) = \neg\mathcal{E}_c(f)$, *c'est-à-dire* $\mathcal{E}_c(\neg f)$ *est vraie ssi* $\mathcal{E}_c(f)$ *est fausse.*
- $\forall (f,g) \in \mathcal{F}^2$, *et pour tout opérateur logique* `op`, $\mathcal{E}_c(f\texttt{op}g) = \mathcal{E}_c(f)\texttt{op}\mathcal{E}_c(g)$.

On remarque que la fonction Caml est une traduction directe de la définition formelle.

Remarque : Ainsi, donner une formule de n variables propositionnelles fournit une fonction de $\mathcal{B}^n$ dans $\mathcal{B}$.

2.4 Formule satisfiable

Définition 2.4. *Soit f une formule.*

- *f est satisfiable s'il existe un contexte c tel que $\mathcal{E}_c(f)$ est vrai.*
- *f est une tautologie si pour tout contexte c, $\mathcal{E}_c(f)$ est vrai.*
- *f est une antilogie si pour tout contexte c, $\mathcal{E}_c(f)$ est faux.*

Définition 2.5. *Soient f et g deux formules. On dit qu'elles sont équivalentes, et on note $f \equiv g$ lorsque pour tout contexte c, $\mathcal{E}_c(f) = \mathcal{E}_c(g)$.*

La question de savoir si une formule est satisfiable ou pas est un sujet central en informatique théorique. En effet, de nombreux problèmes peuvent être traduit par une formule logique et il s'agit de trouver les contextes la vérifiant.

Une stratégie naïve pour déterminer si une formule est satisfiable, ou si c'est une tautologie est de dresser la liste de tous les contextes, et d'évaluer la formule en chacun d'entre eux.

N.B. Cela revient à construire la table de vérité de la formule.

On va créer les fonctions intermédiaires suivantes :

- `variables : formule -> string list` qui renvoie la liste des variables utilisées
- `tousLesContextes : string list -> (string*bool) list list` qui prend une liste de variables et renvoie la liste de tous les contextes possibles avec ces variables.

Si n est le nombre de variables employées dans la formule, alors le nombre de contextes est 2^n. Ainsi la complexité de notre fonction sera au moins en 2^n, ce qui fait que cet algorithme n'est pas praticable au dela de la dizaine de variables.

```
let rec concatSansDoublon l1 l2=
  (* l1 et l2 sont deux listes sans doublon
  Ceci renvoie l1@l2 privée des doublons   *)
  match l1 with
  |[] -> l2
  |t::q when List.mem t l2  -> concatSansDoublon q l2
  |t::q   ->t::concatSansDoublon q l2
;;

let rec variables = function
  | Vrai  -> []
  | Faux  -> []
  | Variable x  -> [x]
  | Non f     -> variables f
  | OpBinaire (f1, op, f2)
    -> concatSansDoublon (variables f1)
   (variables f2)
;;
```

Remarque : Pour plus d'efficacité, on pourrait faire en sorte que les listes soient toujours triées. Alors pour `concatSansDoublon`, on utilise une légère variante de la fusion utilisée dans le tri fusion.

Passons au calcul de tous les contextes :

```
let rec ajouteDevant x l=
  (* l est une liste de listes
  Ceci renvoie la liste obtenue en rajoutant x
  devant chaque élément de l   *)
  match l with
    |[] -> []
    |t::q -> (x::t)::(ajouteDevant x q)
;;

(* En une ligne : *)
let ajouteDevant x = map (fun l -> x::l);;


let rec tousLesContextes listeVariables=
  match listeVariables with
  |[] -> [[]]
  |t::q -> let suite= tousLesContextes q in
     (* les contextes où t est vrai*)
     ajouteDevant (t,true) suite
     @
       (* les contextes où t est faux*)
       ajouteDevant (t,false) suite
;;
```

On déduit alors facilement une fonction qui renvoie la liste des contextes vérifiant une formule :

```
let contextesVérifiant f=
    let listeVariables = variables f in
    let listeContextes = tousLesContextes listeVariables in
```

```
    let rec parcourtListe = function
        (* fonction auxiliaire pour parcourir la liste des contextes *)
        (* Renvoie la liste des contextes pour lesquels f est vraie *)
        |[] -> []
        |c::q when evaluation f c -> c ::parcourtListe q
        |c::q     -> parcourtListe q

  in parcourtListe listeContextes;;
```

On écrit facilement quelques variantes : tester si une formule est satisfiable, est une tautologie, ou si deux formules sont équivalentes.

2.5 Interlude : induction structurelle

2.5.1 Exemple : les entiers naturels

Les entiers naturels sont définis ainsi :

```
type entier_nat = Zero | Succ of entier_nat;;
```

Autrement dit, un entier naturel peut être :

- l'entier nul ;
- ou le successeur d'un autre entier.

Le premier constructeur est constant, le second prend un argument, qui est un entier naturel.

Maintenant, le théorème de la récurrence est le suivant :

Théorème 2.6. *Soit P un prédicat sur $\mathbb{N}$. On suppose que :*

- $P(0)$
- *Pour tout $n \in \mathbb{N}$ tel que $P(n)$, on a $P(n+1)$. (Si P est vrai sur un entier, alors il est vrai sur son successeur.)*

Alors pour tout $n \in \mathbb{N}$, $P(n)$.

Autrement dit, pour démontrer que P est toujours vrai, il suffit de vérifier :

- Qu'il est vrai sur le constructeur constant ;
- qu'il est héréditaire sur le constructeur récursif.

2.5.2 Pour les formules

Le théorème analogue à celui de la récurrence sur $\mathbb{N}$ est vrai pour les formules :

Théorème 2.7. *Soit P un prédicat sur $\mathcal{F}$. On suppose :*

- *Pour tout $b \in \mathcal{B}$, $P(b)$;*
- *Pour tout $x \in \Sigma$, $P(x)$;*
- *Pour tout $f \in \mathcal{F}$, $P(f) \Rightarrow P(Nonf)$:*
- *Pour tout $(f,g) \in (\mathcal{F})^2$, $\big(P(f) \wedge P(g)\big) \Rightarrow P(f \wedge g)$;*
- *Pour tout $(f,g) \in (\mathcal{F})^2$, $\big(P(f) \wedge P(g)\big) \Rightarrow P(f \vee g)$.*

Alors, $\forall f \in \mathcal{F}$, $P(f)$.

Ainsi, pour prouver que P est toujours vrai, il suffit de prouver qu'il est vrai pour tous les constructeurs non récursifs, et qu'il est héréditaires pour tous les constructeurs récursifs.

Cette généralisation de la récurrence classique est appelée « induction structurelle ».

On la démontre facilement en utilisant le théorème de récurrence classique sur la hauteur de l'arbre qui représente la formule.

En devoir, vous pourrez toujours utiliser une récurrence classique sur la hauteur de l'arbre, mais ce théorème permet de gagner un peu de temps sur la rédaction.

Enfin, le principe est le même pour tous les type construits, notamment pous les arbres.

cf exercice : 9

2.6 Forme normale

On commence par un peu de vocabulaire :

Définition 2.8.

- *Une formule f est appelée un littéral lorsqu'il existe v une variable telle que $f = v$ ou $f = \overline{v}$.*
- *Un monôme est une conjonction de littéraux.*
- *Un minterme pour $\mathcal{A}$ est une conjonction de littéraux (donc un monôme) où chaque variable de $\mathcal{A}$ apparaît une et une seule fois.*
- *De même, un maxterme pour $\mathcal{A}$ est une disjonction de littéraux où chaque variable de $\mathcal{A}$ apparaît une et une seule fois.*

Remarque : Il y a dont un et un seul contexte qui satisfasse un minterme, et un et un seul contexte que ne satisfasse pas un maxterme. D'où les noms.

Soit f une formule quelconque. La distributivité de $\cdot$ sur $+$ permet de développer f comme une somme de monômes. Ensuite, si certains monômes ne sont pas des mintermes, il suffit d'utiliser $1 = x + \overline{x}$, et de redévelopper, pour n'obtenir plus que des mintermes.
Remarque : On remarque que deux mintermes différents sont toujours incompatibles, de sorte que le $+$ pourrait être remplacé par $\oplus$ dans la formule obtenue.

La formule obtenue s'appelle la forme normale disjonctive de f. Remarquez que les mintermes obtenus correspondent aux ligne de la table de vérité de f où le résultat est 1. En particulier, ceci prouve *l'unicité* de la forme normale disjonctive.

Ce qui est moins habituel c'est que $+$ aussi est distributif sur $\cdot$. De sorte qu'on peut refaire le même procédé en échangeant $+$ et $\cdot$. On obtient la « forme normale conjonctive de f ».
Remarque : La forme normale conjonctive de f peut aussi être obtenue à partir de la forme normale disjonctive de $\neg f$ par les lois de De Morgan sur la négation. En particulier, on peut en déduire l'unicité de cette forme normale.

Proposition 2.9.

- *Toute formule est équivalente à une unique (à l'ordre des termes près) disjonction de minterme.*
- *Toute formule est équivalente à une unique (à l'ordre des termes près) conjonction de maxterme.*

Définition 2.10. *Soit f une formule.*

- *L'unique disjonction de mintermes équivalente à f s'appelle la forme normale disjonctive de f.*
- *L'unique conjonction de maxxtermes équivalente à f s'appelle la forme normale conjonctive de f.*

Corollaire 2.11. *Deux formules sont équivalentes ssi elles ont la même formu normale disjonctive, ssi elle ont la même forme normale conjonctive.*

Proposition 2.12.

- *Le nombre de termes dans la forme normale disjonctive est le nombre de contextes sur $\mathcal{A}$ pour lesquelles f est vraie.*
- *Le nombre de termes dans la forme normale conjonctive est le nombre de contextes sur $\mathcal{A}$ pour lesquelles f est fausse.*

Ainsi, mettre deux formules sous forme normale est une manière automatique de vérifier si elles sont équivalentes. De même, mettre une formule sous forme normale permet de prouver qu'il s'agit d'une tautologie.

cf exercice : 11

3 Commentaires

3.1 Problèmes NP-complets

On a déjà dit que notre programme pour tester si une formule est satisfiable a une complexité exponentielle. Une conjecture classique affirme qu'il n'existe pas d'algorithme de complexité polynomiale permettant de tester la satisfiabilité d'une formule logique quelconque. Malgré de nombreuses recherches, cette conjecture n'a jamais pu être ni démontrée ni infirmée.

En outre il a été démontré que de nombreux problèmes sont équivalents à celui de la satisfiabilité : une solution de l'un permettrait d'obtenir une solution de l'autre.

Ainsi, le problème de la satisfiabilité d'une formule de logique sert de référence pour tout une classe de problèmes dont on soupçonne qu'il n'existe pas de solution en temps polynomiale. Ces problèmes sont appelés les problèmes NP-complets. Citons par exemple le problème du voyageur de commerce, la création d'emploi du temps...

Deuxième partie

Exercices

Exercices : Logique

1 Connecteurs logiques

Exercice 1. *! Ou exclusif

On note $\oplus$ la disjonction exclusive, définie par $\forall (x,y) \in \mathcal{B}^2$, $x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)$.

1. Démontrer que $\oplus$ est associative.
2. Démontrer que $(\mathcal{B}, \oplus, \wedge)$ est isomorphe à $\big(\mathbb{Z}/2\mathbb{Z}, +, \cdot\big)$ (isomorphisme de corps).
3. Soient x et y deux variables booléennes. Simplifier les formules $\big(x \oplus (x \oplus y)\big)$ et $x \oplus (x \oplus y) \oplus (x \oplus y)$.
4. La fonction Caml `lxor` [1] effectue une disjonction exclusive sur chaque bit des deux entiers qui lui sont fournis. Écrire en se basant sur cette fonction une fonction `échange` qui échange de contenu de deux références à des entiers.

Exercice 2. *! Fonctions Caml à rédiger avec des connecteurs logique

1. Écrire les fonctions `for_all`, `exists`, et `exists_unique` en n'utilisant aucun `when` ni aucun `if`.
2. Même question avec la fonction `mem` sur les listes.
3. Même question avec la fonction `memTab` qui cherche un élément dans un tableau.
4. Même question avec la fonction `memABR` sur les arbres binaires de recherche.
5. Même question pour une fonction indiquant si une année est bissextile.

Exercice 3. *! Identités remarquables

Dans cet exercice, on adopte les notations $\cdot, +, 0, 1$ pour $\wedge, \vee, \bot, \top$.

1. (idempotence) Soit $x \in \mathcal{B}$, simplifier x^2 et $x + x$.
2. (éléments absorbants) Soit $a \in \mathcal{B}$. Que valent $0 \cdot a$ et $1 + a$?
3. Soit $a \in \mathcal{B}$. Que valent $a \cdot \overline{a}$ et $a + \overline{a}$?
4. Soit $(a,b) \in \mathcal{B}^2$. Simplifier $a + ab$.
5. Montrer que $\forall (a,b,c) \in \mathcal{B}^3$, $(a+b)\cdot(a+c) = a + bc$

Exercice 4. ** Tous les connecteurs logiques

Donner toutes les tables d'opérations de lci sur $\mathcal{B}$ qui soient commutatives. Pour chacune, reconnaître le connecteur logique associé, ou l'exprimer de manière simple à partir des connecteurs logiques connus.

Exercice 5. * Implication

Quatre cartes vous sont présentées. Elles contiennent toutes une lettre de l'alphabet sur chacune de leur faces. Sur les faces visibles, on lit : D G P L

Combien faut-il retourner de carte(s) pour vérifier la proposition : « Derrière tout G se trouve L » ?

2 Formules logiques

Exercice 6. *! Exemples de tautologies

Soient p, q et r des variables propositionnelles. Vérifier que les formules suivantes sont des tautologies :

1. $\big(p \wedge (p \Rightarrow q)\big) \Rightarrow q$ (modus ponens)
2. $(p \Rightarrow q \wedge \neg q) \Rightarrow \neg p$ (modus tollens)
3. $\big((p \Rightarrow q) \wedge (q \Rightarrow r)\big) \Rightarrow (p \Rightarrow r)$
4. $(p \Rightarrow q) \vee (q \Rightarrow p)$
5. $\big((p \Rightarrow q) \Rightarrow p\big) \Rightarrow p$
6. $\overline{pq} + \overline{p}q\overline{r} + p\overline{r} + p\overline{q}r + qr$

Exercice 7. * Contraposée

Soit p et q des variables propositionnelles. Montrer que $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$.

Exercice 8. ** Logique et chimie

Un raisonnement fréquent en chimie est celui-ci : « Supposons la concentration de l'espèce 1 négligeable devant la concentration de l'espèce 2. Alors... (faire des calculs)... On voit alors que la concentration de l'espèce 1 était bien négligeable. J'en déduis que la concentration de l'espèce 1 est négligeable.»

1. En Python c'est $\wedge$.

1. Écrire la formule logique que sous-tend ce raisonnement.
2. Cette formule est-elle une tautologie ? La simplifier.

Exercice 9. **! Non-et

L'opérateur « incompatibilité », ou encore NON-ET, ou NAND en anglais, que nous noterons $\overline{\wedge}$ dans cet exercice, est définie ainsi : $\forall (x,y) \in \mathcal{B}^2$, $x \overline{\wedge} y = \neg (x \wedge y)$.

Nous le noterons ci-dessous $\overline{\wedge}$

1. Écrire la table d'opération de $\overline{\wedge}$.
2. NON-ET est-il associatif ?
3. Démontrer que toute formule logique peut être écrite uniquement à l'aide de NON-ET.

   Remarque : Voici typiquement le genre de démonstration à savoir faire en vue des concours.

   En outre, réaliser une porte NAND ne nécessite que 4 transistors, contre 5 pour un ET. En fait, pour câbler un ET, on commence par un NON-ET qu'on fait suivre d'un NON (un transistor de plus).
   Ainsi, on trouvera de nombreuses portes NAND dans les circuits électroniques.
   De même on peut câbler un NON-OU (NOR en anglais) en 4 transistors, et une porte OU s'obtient en y rajoutant un NON.
4. Écrire une fonction Caml qui convertit une formule logique en une formule utilisant uniquement NON-ET.

Exercice 10. ** Visualisation d'une formule sur trois variables

On considère la formule f sur les trois variables P, Q, R suivante : $f = \overline{PQR + \overline{P}\,\overline{Q}\,\overline{R}}$.

1. Donner la forme normale conjonctive, puis la forme normale disjonctive de f.
2. Écrire f comme somme de trois monômes.

3 Formules logiques : implémentation Caml

Exercice 11. * Mise sous forme normale**

1. Écrire une fonction `negVersLeBas` qui descend toutes les négations juste au dessus des feuilles en utilisant les formules de De Morgan.
2. Écrire une fonction prenant en entrée une forme logique et renvoyant sa forme normale disjonctive.

Exercice 12. ** Triléens, application à la satisfiabilité

Les triléens sont au nombre de trois : Vrai, Faux, et Indéterminé.

```
type trileen = Vrai | Faux | Indetermine;;
```

1. Définir les opérations $\wedge$, $\vee$ et $\neg$ qui vous paraissent cohérentes. Implémenter les fonctions correspondantes dans Caml.
2. Les variables sont numérotées et donc représentées par des entiers.

   Nous considérerons des valuations « partielles » : il s'agira d'un tableau de triléens. Pour une telle valuation `v`, pour tout i `v.(i)` indique si la variable i est Vrai, Fausse, ou Indéterminée.

   On définira alors les formules par le type :

   ```
   type formule = Constante of bool | Variable of int | Non of formule | Et of formule *
       ↪ formule | Ou of formule * formule;;
   ```

   (a) Écrire une fonction `eval_paresseuse` prenant en entrée une formule et une valuation partielle et évaluant cette formule pour cette valuation. On prendra soin de ne pas faire de calcul inutile : si l'évaluation d'un des deux arguments d'un $\wedge$ ou d'un $\vee$ suffit à conclure, ne pas calculer l'autre argument.

   (b) Un littéral sera représenté par le type `type lit = V of int  F of int | (` Si les variables sont $v_0, ..., v_n$, `V i` représente v_i, et `NV i` représente $\neg v_i$). Écrire une fonction prenant une liste de littéraux, et indiquant si la formule formée de la conjonction de ces littéraux est satisfiable. Cette fonction devra parcourrir une seule fois la liste.

   (c) (***) Maintenant on veut écrire une fonction pour tester si une formule est satisfiable qui soit plus efficace que celle du cours.

i. Écrire une première fonction pour tester si une formule `f` est satisfiable. On créera une valuation partielle `val`, initialement entièrement indéterminée, que l'on remplira peu à peu. On créera une fonction auxiliaire prenant en entrée i indice de la prochaine case de `val` à remplir.

Lorsque `val` est entièrement remplie (cas d'arrêt) la fonction `eval_paresseuse` permet de conclure.

Dans le cas général, on fait un essai en mettant `val.(i)` à `Vrai`, puis un essai en mettant `val.(i)` à `Faux`, et on renvoie `true` si un des deux essais au moins à renvoyé `true`, `false` sinon.

ii. À présent, modifier la fonction précédente pour effectuer une évaluation partielle en entrée de chaque appel à la fonction auxiliaire. Si cet appel renvoie autre chose que `Indetermine`, on peut conclure immédiatement.

Exercice 13. ** Écriture polonaise inversée

Écrire une fonction prenant en entrée une formule logique écrite en notation polonaise inversée et renvoyant l'arbre représentant cette formule. On donne ou rappelle la fonction Caml `string.split_on_char` qui découpe une chaîne de caractères selon un caractère passé en argument.

Résoudre l'exercice 4.1 au moyen de la fonction précédente, et de la fonction `contextesVerifiant` écrite en cours.

Exercice 14. ** Formes normales

On définit le type suivant pour représenter un littéral :

```
type litteral = V of char | NV of char;;
```

Une forme normale peut être implémentée par une simple liste de listes de littéraux.

1. Écrire une fonction pour évaluer une formule sous forme normale conjonctive donnée par une liste de listes de littéraux.
2. Même question pour une forme normale disjonctive.

4 Exercices à la CCP

4.1 Ethnologie (CCP 2017)

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau. Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons X, Y et Z. Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner.

Le premier sujet abordé est la région dans laquelle se trouve le village :

X indique : « Le village se trouve dans la vallée » ;

Z réplique : « Non, il ne s'y trouve pas » ;

X reprend : « Ou alors dans les collines ».

Nous noterons V et C les variables propositionnelles associées à la région dans laquelle se trouve le village.

Nous noterons X_1 et Z_1 les formules propositionnelles correspondant aux affirmations de X et de Z sur le premier sujet.

Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

X dit : « Le chemin de gauche conduit au village » ;

Z répond : « Tu as raison » ;

X complète : « Le chemin de droite y conduit aussi » ;

Y affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;

Z indique : « Celui du milieu n'y conduit pas ».

Nous noterons G, M, D les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village.

Nous noterons X_2 , Y_2 et Z_2 les formules propositionnelles correspondant aux affirmations de X, de Y et de Z sur le second sujet.

1. Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendants des formules propositionnelles X_1 et Z_1.

2. Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions X_1 et Z_1 dépendant des variables V et C.
3. En utilisant la résolution avec les propriétés des opérateurs booléens et les formules de De Morgan en calcul des propositions, déterminer dans quelle région vous devez vous rendre pour rejoindre le village.
4. Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_1, Y_2 et Z_2.
5. Représenter les informations données par les participants sous la forme de trois formules du calcul des proposition X_2, Y_2 et Z_2 dépendant des variables G, M, et D.
6. En utilisant la résolution avec les tables de vérité en calcul des propositions, déterminer quel chemin vous devez suivre pour rejoindre le village.
7. En admettant que les trois participants aient menti, pouviez-vous prendre d'autres chemins ? Si oui le ou lesquels ?

4.2 Test de Turing (CCp 2015)

De nombreux travaux sont réalisés en Intelligence Artificielle pour construire un programme qui imite le raisonnement humain et soit capable de réussir le test de Turing, c'est-à-dire qu'il ne puisse pas être distingué d'un être humain dans une conversation en aveugle. Vous êtes chargé(e)s de vérifier la correction des réponses données par un tel programme lors des tests de bon fonctionnement. Dans le scénario de test considéré, le comportement attendu est le respect de la règle suivante : pour chaque question, le programme répondra par trois affirmations dont une seule sera correcte. Nous noterons A_1 , A_2 et A_3 les propositions associées aux affirmations effectuées par le programme.

1. Représenter le comportement attendu sous la forme d'une formule du calcul des propositions qui dépend de A_1, A_2, et A_3.

   Premier cas : que manger ?

   Vous demandez au programme : Quels éléments doivent contenir les aliments que je dois consommer pour préserver ma santé ?

   Il répond les affirmations suivantes :

   A_1 Consommez au moins des aliments qui contiennent des glucides, mais pas des lipides !

   A_2 Si vous consommez des aliments qui contiennent des glucides alors ne consommez pas d'aliments qui contiennent des lipides !

   A_3 Ne consommez aucun aliment qui contient des lipides !

   Nous noterons G, respectivement L, les variables propositionnelles qui correspondent au fait de consommer des aliments qui contiennent des glucides, respectivement des lipides.
2. Exprimer A_1, A_2, et A_3 sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables G et L.
3. En utilisant le calcul des propositions (formules de De Morgan), déterminer ce que doivent contenir les aliments que vous devez consommer pour préserver votre santé.

   Second cas : quelles activités pratiquer ?

   Vous demandez au programme : Quelles activités dois-je pratiquer si je veux préserver ma santé ?

   Suite à une coupure de courant, la dernière affirmation est interrompue.

   A_1 Ne faites des activités sportives que si vous prenez également du repos !

   A_2 Si vous ne faites pas d'activité intellectuelle alors ne prenez pas de repos !

   A_3 Prenez du repos ou faites des activités . . . !

   Nous noterons S, I et R les variables propositionnelles qui correspondent au fait de faire des activités sportives, des activités intellectuelles et de prendre du repos.
4. Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre de S, I, et R.
5. En utilisant une table de vérité, déterminer quelle(s) activité(s) vous devez pratiquer pour préserver votre santé.

4.3 Autres

Exercice 15. ** Trois processeurs

Un ordinateur est composé de trois processeurs d'une nouvelle génération capables de détecter les pannes des autres composants. Ils sont branchés à la même mémoire.

Un processeur en état de fonctionnement normal n'affirme que des assertions vraies, un processeur en état de panne n'affirme que des assertions fausse.

On pose les deux question suivantes au processeur 1 :

- Les processeurs 2 et 3 sont-ils en état de fonctionnement normal ?
- Le processeur 2 est-il en état de fonctionnement normal ?

Le processeur 1 répond « oui » à la première question et « non » à la seconde.

On suppose que l'état des processeurs ne change pas entre deux questions. On note P_1, resp. P_2 et P_3 la proposition « Le processeur 1 (resp. 2 et 3) est en panne ».

1. Exprimer les réponses aux deux questions sous forme d'une formule logique dépendant de P_1, P_2 et P_3.
2. Déterminer l'état de chaque processeur.

Exercice 16. ** Le menteur, le juste, et les deux portes

Vous êtes face à 2 portes, l'une donne sur l'enfer et l'autre sur le paradis. Vous ne savez pas laquelle mène au paradis, et laquelle à l'enfer.

Juste à côté de ces portes, sont présentes deux personnes. L'une est un menteur, l'autre dit toujours la vérité.

Vous ne pouvez poser qu'une seule et même question aux deux personnes pour savoir quelle porte prendre.

Quelle est cette question ?

Exercice 17. ** Logique cognitive

Trois logiciens vont prendre un verre au café. Le serveur demande : « Un demi pour tout le monde ? »

Le premier logicien répond « Je ne sais pas ». Le second répond « Moi non plus. » Le troisième enfin : « Alors je sais. »

Que vont prendre les trois logiciens ?

Quelques indications

1 1. Utiliser une table de vérité.

9 1.

2.

3. Après avoir vérifier que les opérateurs $\neg$ $\wedge$ et $\vee$ peuvent être écrit à l'aide de NON-ET, utiliser une récurrence sur la hauteur de la formule.
4. La récurrence de la question précédente se traduit immédiatement en une fonction récursive.

10 Remarquer que f est vrai ssi au moins une des variables est vraie et ue autre fausse.

On peut aussi considérer f comme un opérateur trinaire et dessiner l'analogue de sa table d'opération. Ici ce sera un tableau à trois dimensions.

11

16 Que répondrait ton collègue ?

17 Ils vont prendre un demi.