TP de Logique 1 (HLIN402)

Licence Informatique - Université Montpellier 2
M. Leclère - leclere@lirmm.fr

Résumé

L'objectif de ce TP est d'implanter toutes les notions définies sur les propositions. Il s'agit donc d'une part de définir des représentations pour les *connecteurs, symboles propositionnels, propositions (fbf), ensembles de propositions, interprétations, ensembles d'interprétations, littéraux, clauses, formes clausales* et des fonctions de manipulation de ces représentations. D'autre part d'implanter les différentes méthodes de preuve définies en logique des propositions. Ce TP peut être réalisé dans n'importe quel langage fonctionnel.

Pour réaliser ce TP vous ne disposez que de 8 séances aussi il faut absolument que vous travailliez en dehors des TP pour respecter l'échéancier suivant :

- La séance 1 doit être consacrée aux questions Q1 à Q4.
- La séance 2 doit être consacrée aux questions Q5 à Q11 (la question Q6 est optionnelle).
- La séance 3 doit être consacrée aux questions Q12 à Q18.
- La séance 4 doit être consacrée aux questions Q19 à Q27.
- La séance 5 doit être consacrée aux questions Q28 à Q33.
- Les séances 6 et 7 doivent être consacrées à l'une des questions Q34, Q35 ou Q36.
- La séance 8 doit ête consacrée à la question Q37 .

Une évaluation de votre travail sera réalisée et constituera une partie de la note de CC.

1 Représentation des propositions

La première étape consiste à se doter de fonctions permettant de manipuler un type abstrait **proposition**. Il faut donc définir la représentation des :

- symboles propositionnels
- constantes logiques
- connecteurs
- des formules bien formées

En Scheme, on prendra les listes pour représenter les formules en utilisant une notation préfixée. On pourra par exemple représenter la formule $F = a \wedge c \leftrightarrow \neg b \vee (c \rightarrow \bot \wedge \top)$ par `(define F '(<-> (^ a c) (v (! b) (-> c (^ B T)))))`.

Q 1 Définir les formules suivantes dans la représentation Scheme proposée (on rappelle qu'il faut utiliser un *quote* pour bloquer l'évaluation d'une expression :

$$F1 = a \wedge b \leftrightarrow \neg a \vee b$$

$$F2 = \neg(a \wedge \neg b) \vee \neg(a \rightarrow b)$$

$$F3 = \neg(a \rightarrow a \vee b) \wedge \neg\neg(a \wedge (b \vee \neg c))$$

$$F4 = (\neg a \vee b \vee d) \wedge (\neg d \vee c) \wedge (c \vee a) \wedge (\neg c \vee b) \wedge (\neg c \vee \neg b) \wedge (\neg b \vee d)$$

Q 2 On souhaite définir une fonction `fbf?` qui vérifie qu'une expression quelconque est une formule bien formée de la logique des propositions. On utilisera les fonctions suivantes pour reconnaître les différents éléments de l'alphabet utilisé pour construire des propositions.

```
(define (neg? f) (eq? f '!))
(define (and? f) (eq? f '^))
(define (or? f) (eq? f 'v))
(define (imp? f) (eq? f '->))
(define (equ? f) (eq? f '<->))
(define (top? f) (eq? f 'T))
(define (bot? f) (eq? f 'B))
(define (symbLog? f) (or (top? f) (bottom? f) (and? f) (or? f) (neg? f) (imp? f) (equ? f)))
```

```
(define (symbProp? f) (and (symbol? f) (not (symbLog? f))))
(define (atomicFbf? f) (or (symbProp? f) (top? f) (bottom? f)))
(define (conBin? f) (or (and? f) (or? f) (imp? f) (equ? f)))
```

Vérifier que vos formules précédentes sont bien formées.

2 Syntaxe des propositions

Afin de se promener dans l'arborescence syntaxique d'une formule bien formée on utilisera les fonctions :

```
(define (conRac f) (car f)) ; retourne le connecteur racine d'une fbf non atomique
(define (fils f) (cadr f)) ; retourne le fils d'une fbf dont la racine est un connecteur unaire
(define (filsG f) (cadr f)) ; retourne le fils gauche d'une fbf dont la racine est un connecteur binaire
(define (filsD f) (caddr f)) ; retourne le fils droit d'une fbf dont la racine est un connecteur binaire
(define (negFbf? f)
        (and (not (atomicFbf? f))
             (neg? (conRac f)))) ; teste si une fbf a une négation comme connecteur racine
```

Q 3 Écrire la fonction `nbc` qui retourne le nombre de connecteurs d'une fbf (nombre d'occurrences).

Q 4 Écrire la fonction `prof` qui retourne la profondeur d'une fbf.

Q 5 Écrire la fonction `ensSP` qui retourne l'ensemble des symboles propositionnels d'une proposition donnée. On représentera les ensembles par de simples listes ; vous utiliserez les fonctions `set-union, set-intersect, set-subtract, set-add, set-remove, set=?`.

Q 6 Écrire une fonction `affiche` qui prend en donnée une fbf et affiche cette formule sous forme infixée complètement parenthésée (écriture classique). Par exemple `(affiche F1)` retournera la chaîne $((a \wedge b) \leftrightarrow \neg(a \vee b))$. On utilisera la fonction `display` pour les affichages.

3 Sémantique des propositions

On représentera une interprétation par une liste de paires pointées `(symbole . valeur)`.

Q 7 Définir les 3 interprétations $I1$, $I2$, $I3$ suivantes : $I1(a) = I1(c) = 1$ et $I1(b) = 0$, $I2(a) = I2(b) = I2(c) = 0$, $I3(a) = I3(b) = I3(c) = 1$.

Q 8 Écrire la fonction `intSymb` qui retourne la valeur d'interprétation (0 ou 1) d'un symbole propositionnel donné dans une interprétation donnée (on supposera que ce symbole propositionnel apparaît dans la structure d'interprétation). Exemple : `(intSymb 'b I1)` doit retourner 0.

Q 9 Écrire les fonctions d'interprétation (unaires, binaires ou 0-aire) des connecteurs et constantes logiques : `intNeg, intAnd, intOr, intImp, intEqu, intTop, intBot`.

Q 10 Écrire la fonction `valV` qui calcule la valeur de vérité d'une formule f pour une interprétation i **complète** pour f.

Q 11 Écrire un prédicat `modele?` qui, étant donné une inteprétation et une fbf, retourne *true* si l'interprétation est un modèle de la formule.

4 Satisfiabilité et validité d'une proposition

Afin d'étudier les propriétés sémantiques des propositions, on a besoin de considérer l'ensemble de toutes les interprétations des symboles propositionnels d'une formule donnée.

Q 12 Définir en Scheme l'ensemble de toutes les interprétations des symboles propositionnels p et q.

Q 13 Écrire une fonction `ensInt` qui prend en donnée un ensemble de symboles propositionnels et retourne l'ensemble de toutes les interprétations de ces symboles propositionnels. *Indication : lorsqu'il n'y a pas de symbole propositionnel, il n'y a qu'une seule interprétation possible : (()). S'il y en a au moins un, il est judicieux de calculer récursivement l'ensemble I des interprétations de tous les symboles sauf le premier, puis de prendre en compte le premier symbole en ajoutant à chaque interprétation de I l'interprétation du premier symbole (une fois à* 0 *et une fois à* 1*).* Vous utiliserez les fonctions `let`, `append`, `map` et pourrez exploiter les fonctions anonymes (voir les *lambda expression*).

Q 14 Écrire un prédicat `satisfiable?` qui retourne *true* si et seulement si une fbf donnée est satisfiable. Vous utiliserez la fonction `ormap`. Testez votre prédicat sur les propositions $a, \neg a, (a \wedge b), ((a \wedge b) \wedge \neg a), F_1, F_2, F_3, F_4$.

Q 15 Écrire un prédicat `valide?` qui retourne *true* si et seulement si une fbf donnée est valide. Vous utiliserez la fonction `andmap`. Testez votre prédicat sur les propositions $a, \neg a, (a \vee b), ((a \vee b) \vee \neg a), F_1, F_2, F_3, F_4$.

Q 16 Écrire un prédicat `insatisfiable?` qui retourne *true* si et seulement si une fbf donnée est insatisfiable. Vous utiliserez la fonction `filter`.

5 Equivalence et conséquence logique

Q 17 Écrire **deux** versions d'un prédicat `equivalent?` qui teste si deux fbf données sont sémantiquement équivalentes : l'une `equivalent1?` s'appuiera sur la définition du cours (elles ont les mêmes valeurs de vérité pour toutes les interprétations), l'autre `equivalent2?` exploitera le lien entre équivalence sémantique et validité. Faire des tests ! En particulier $((a \vee b) \vee \neg a) \equiv \neg((c \wedge d) \wedge \neg c)$.

Q 18 Écrire un prédicat `consequence2?` qui étant donnée deux propositions $F1$ et $F2$ retourne *true* si $F2$ est conséquence logique de $F1$. Vérifier que $a \models (a \vee b)$, $a \not\models (a \wedge b)$, $((a \vee b) \vee \neg a) \models \neg((c \wedge d) \wedge \neg c)$, $((a \wedge b) \wedge \neg a) \models (c \vee d)$.

Pour la définition générale de la conséquence logique, on considère des ensembles de fbf naturellement représentés par une liste de fbf.

Q 19 Écrire la fonction `ensSPallFbf` qui retourne l'ensemble des symboles propositionnels d'un ensemble de propositions donné, extension de la fonction `ensSP` à des ensembles de formules.

Q 20 Écrire un prédicat `modeleCommun?` qui retourne *true* si une interprétation donnée est modèle d'un ensemble de propositions donné, extension du prédicat `modele?` à des ensembles de formules.

Q 21 Écrire un prédicat `contradictoire?` qui retourne *true* si et seulement si un ensemble de propositions est contradictoire.

Q 22 Écrire un prédicat `consequence?` prenant en donnée un ensemble de formules $\{f_1, f_2, ...f_n\}$ et une fbf f et retournant true si f est conséquence logique de $f_1...f_n$ (c'est à dire $\{f_1, f_2, ...f_n\} \models f$). Vérifier que $\{a \wedge b, \neg a, b \rightarrow d\} \models c \rightarrow d$.

Q 23 Proposer une deuxième version de ce prédicat `consequenceV?` exploitant le lien entre conséquence logique et validité. Vous aurez certainement besoin d'écrire une fonction intermédiaire `conjonction` qui étant donné un ensemble de formules retourne la formule composée de la conjonction de toutes ces formules.

Q 24 Proposer une troisième version de ce prédicat `consequenceI?` exploitant le lien entre conséquence logique et insatisfiabilité.

6 Mise sous forme conjonctive

Les 5 premières questions visent à fournir les transformations de fbf permettant un passage à la forme conjonctive. Pour ces fonctions, il faut raisonner sur l'arbre syntaxique associé à la formule. La 6^{e} vise à fournir cette fonction. Attention, la 5^{e} fonction (`distOu`) est particulièrement délicate. Ex. : `(distOu '(^ (^ a (! b)) (v c (v (! d) (^ e f)))) )` doit retourner `(^ (^ a (! b)) (^ (v c (v (! d) e)) (v c (v (! d) f))))`.

Q 25 Écrire une fonction récursive `oteEqu` qui prend en paramètre une fbf et retourne une fbf logiquement équivalente qui ne contient pas de connecteur $\leftrightarrow$. Rappel : $(A \leftrightarrow B) \equiv ((A \rightarrow B) \wedge (B \rightarrow A))$

Q 26 Écrire une fonction récursive `oteImp` qui prend en paramètre une fbf et retourne une fbf logiquement équivalente qui ne contient pas de connecteur $\rightarrow$. Rappel : $(A \rightarrow B) \equiv (\neg A \vee B)$

Q 27 Écrire une fonction récursive `oteCste` qui prend en paramètre une fbf et retourne une fbf logiquement équivalente qui ne contient pas de constante logique. Rappel : $\top \equiv (\neg p \vee p)$ et $\bot \equiv (\neg p \wedge p)$

Q 28 Écrire une fonction récursive `redNeg` qui prend en paramètre une fbf ne contenant pas de connecteur $\leftrightarrow$ et $\rightarrow$ et retourne une fbf logiquement équivalente dont la négation ne porte que sur les symboles propositionnels. Rappel : $\neg\neg A \equiv A$, $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$, $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$

Q 29 Écrire une fonction récursive `distOu` qui prend en paramètre une fbf composée de littéraux connectés par des $\wedge$ et $\vee$ et retourne une fbf logiquement équivalente sous forme conjonctive (i.e. conjonction de disjonctions de littéraux). Rappel : $(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$

Q 30 Écrire alors la fonction `formeConj` qui prend en paramètre une fbf quelconque et retourne une fbf logiquement équivalente sous forme conjonctive.

7 Forme clausale

On propose de représenter un littéral par un symbole ou la négation d'un symbole. Une clause sera représentée par la liste (sans doublon) des littéraux qui la composent. Et une forme clausale par la liste de ces clauses.

Q 31 Écrire une fonction récursive `transClause` qui prend en paramètre une fbf disjonction de littéraux et retourne la clause correspondante à cette fbf.

Q 32 Écrire une fonction récursive `transEnsClause` qui prend en paramètre une fbf sous forme conjonctive et retourne l'ensemble de clauses correspondant à cette fbf.

Q 33 Finalement, écrire une fonction `formeClausale` qui prend en paramètre une fbf quelconque et retourne l'ensemble de clauses correspondant à sa forme clausale.

8 Méthodes de preuve

Il s'agit d'implanter les méthodes vues en cours. Il est préférable d'en implanter une correctement et complètement plutôt que d'essayer de toutes les faire ! On s'attachera en particulier à :

- implanter la méthode ;
- proposer des fonctions `satisfiable`, `valide` et `consequence` qui s'appuie sur la méthode développée ;
- se doter d'un jeu d'essais permettant de tester la méthode.

Q 34 Mettre en œuvre la méthode de résolution.

Q 35 Mettre en œuvre Davis et Putnam.

Q 36 Mettre en œuvre la méthode des tableaux.

9 Application

Q 37 Modéliser un problème en logique des propositions et résolvez-le à l'aide d'une des trois méthodes précédentes.