TP de Logique des Propositionnelle (HAI304I)

Licence Informatique - Université de Montpellier

M. Leclère - leclere@lirmm.fr

Résumé

L'objectif de ce TP est de représenter des formules bien formées de la logique des propositions et d'implanter l'ensemble des notions sémantiques vues en cours. Pour réaliser ce TP vous ne disposez que de 3 séances aussi il faut absolument que vous travailliez en dehors des TP pour suivre l'échéancier :

— La séance 1 doit être consacrée aux questions Q1 à Q5 (la question Q6 plus difficile est optionnelle).

— La séance 2 doit être consacrée aux questions Q7 à Q16 .

— La séance 3 doit être consacrée aux questions Q17 à Q24.

Une évaluation de votre travail sera réalisée au retour des vacances et constituera une partie de la note de CC.

1 Syntaxe de la logique propositionnelle

On se dote d'un type `prop` défini par :

```
type prop =
  | Symb of string
  | Top
  | Bot
  | Not of prop
  | And of prop * prop
  | Or of prop * prop
  | Imp of prop * prop
  | Equ of prop * prop;;
```

Ainsi la formule $a \land c \Leftrightarrow \neg b \lor (c \Rightarrow \bot \land \top)$ se représente :

```
let f = Equ(And(Symb "a", Symb "c"),Or(Not(Symb "b"),Imp(Symb "c",And(Bot,Top))));;
```

Q 1 Définir les formules suivantes :

$f_1 = a \land b \Leftrightarrow \neg a \lor b$

$f_2 = \neg(a \land \neg b) \lor \neg(a \Rightarrow b)$

$f_3 = \neg(a \Rightarrow a \lor b) \land \neg\neg(a \land (b \lor \neg c))$

$f_4 = (\neg a \lor b \lor d) \land (\neg d \lor c) \land (c \lor a) \land (\neg c \lor b) \land (\neg c \lor \neg b) \land (\neg b \lor d)$

Q 2 Écrire la fonction `nbc` qui retourne le nombre de connecteurs d'une fbf (nombre d'occurrences).

Q 3 Écrire la fonction `prof` qui retourne la profondeur d'une fbf.

Q 4 Écrire la fonction `sp` qui retourne l'ensemble des symboles propositionnels d'une fbf donnée. On représentera un ensemble de symboles comme une liste de chaines de caractères représentant les symboles. Par exemple `sp f` doit renvoyer `["a";"b";"c"]` (peu importe l'ordre mais il ne doit pas y avoir de doublons). Vous aurez certainement besoin d'écrire des fonctions intermédiaires `union` de deux ensembles et `ajouteSiAbsent` d'un élément à un ensemble.

Q 5 Écrire une fonction `affiche` qui prend en donnée une fbf et affiche cette formule sous forme infixée complètement parenthésée (écriture classique).

Q 6 (**) Écrire une fonction `affichePri` qui prend en donnée une fbf et affiche cette formule sous forme infixée la moins parenthésée en tenant compte des règles d'élimination des parenthèses vues en cours.

2 Sémantique des propositions

On se donne un type énuméré valVerite qui contient les valeurs de vérité Zero et Un. On se propose de représenter une interprétation par une liste de couples (symbole, valVerite).

```
type valVerite = Zero | Un ;;

type interpretation = (string * valVerite) list;;
```

Q 7 Définir les 3 interprétations i_1, i_2, i_3 suivantes : $i_1(a) = i_1(c) = 1$ et $i_1(b) = 0$, $i_2(a) = i_2(b) = i_2(c) = 0$, $i_3(a) = i_3(b) = i_3(c) = 1$.

Q 8 Écrire la fonction `intSymb` qui retourne la valeur d'interprétation 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 `Zero`.

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.

3 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. Nous les représenterons par une liste d'interprétations donc une structure du type `(string * valVerite) list list`.

Q 12 Définir en OCaml 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, l'interpétation vide (donc il faut retourner une liste contenant la liste vide : `[[]]`).

S'il y en a au moins un, il suffit 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).

Pour ça, il peut être judicieux d'écrire préalablement une fonction `consTous : 'a -> 'a list list -> 'a list list` qui étant donnés un élément et une liste l de liste d'éléments de ce type ajoute e en tête de chacune des listes de l. Par exemple : `consTous 2 [[1;3;4];[5;3];[];[2;7]]` doit retourner `[[2;1;3;4];[2;5;3];[2];[2;2;7]]`

Q 14 Écrire une fonction `satisfiable` qui retourne *true* si et seulement si une fbf donnée est satisfiable. Cette fonction pourra s'appuyer sur une fonction `existeModele` qui prend en paramètre une fbf f et un ensemble d'interprétations et retourne vrai si l'une des interprétations est un modèle de f. 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 une fonction `valide` qui retourne *true* si et seulement si une fbf donnée est valide (besoin de `tousModele` ?). 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 une fonction `insatisfiable` qui retourne *true* si et seulement si une fbf donnée est insatisfiable.

4 Equivalence et conséquence logique

Q 17 Écrire **deux** versions d'une fonction `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, besoin de `memesModeles` ?), l'autre `equivalent2` exploitera le lien entre équivalence sémantique et validité. Faites des tests ! En particulier $((a \vee b) \vee \neg a) \equiv \neg((c \wedge d) \wedge \neg c)$.

Q 18 Écrire une fonction `consequence2` qui étant donnée deux fbf h et c retourne *true* si c est conséquence logique de h (i.e. $h \models c$). 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 des liste de fbf donc du type `prop list`.

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

Q 20 Écrire une fonction `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 une fonction `contradictoire` qui retourne *true* si et seulement si un ensemble de propositions est contradictoire. Une fonction intermédiaire sera certainement à écrire.

Q 22 Écrire une fonction `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 $\{f_1, f_2, ...f_n\} \models f$). Ici encore, vous aurez certainement besoin d'une fonction intermédiaire. Vérifier que $\{a \wedge b, \neg a, b \Rightarrow d\} \models c \Rightarrow d$.

Q 23 Proposer une deuxième version de cette fonction`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 cette fonction `consequenceI` exploitant le lien entre conséquence logique et insatisfiabilité.