TP de Logique 1 (FLIN406)

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 types abstraits *connecteur, symbole propositionnels proposition (fbf), ensemble de propositions, interprétation, ensemble d'interprétations, littéral, clause, forme clausale.* D'autre part d'implanter les différentes méthodes de preuve définies en logique des propositions. L'énoncé est écrit pour le langage Maple mais ce TP peut être réalisé en LISP auquel cas certaines représentations devront être adaptées. Un fichier contenant les premières définitions de type peut être récupéré à `http://www.lirmm.fr/~leclere/enseignements/LogProp/tpLogique.mws`.

1 Représentation des propositions

Afin de faciliter la manipulation des propositions en Maple, nous utiliserons une représentation préfixée (fonctionnelle) des propositions : le type maple `function` (correspondant aux expressions fonctionnelles). Ainsi les connecteurs seront assimilés à des symboles de fonctions (unaire pour le $\neg$ et binaires pour les autres). À l'aide du mécanisme de définition de type de Maple, nous proposons trois types :

- le type `connecteur`$= \{non, et, ou, impl, equi\}$;
- le type `symbole` permettant d'utiliser n'importe quel `symbol` (nom) maple différent d'un connecteur (et différent d'une constante ou fonction prédéfinie Maple : Pi, sin...) comme *symbole propositionnel* ;
- le type `fbf` qui reconnaît comme formule bien formée une `expression` maple (bien formée) e telle que `fbf?(`e`)=true` (*i.e.* qui satisfait au prédicat de vérification de typage `fbf?`).

Avec cette représentation, nous noterons les propositions p, $\neg p$ et $p \rightarrow q$ par : `p`, `non(p)`, `impl(p,q)`.

Q 1 Définissez les formules suivantes en maple :

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

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

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

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

Q 2 Ecrivez la fonction récursive `fbf?` en vous servant :

- du prédicat Maple de vérification de typage `type` : $Expression \times Type \rightarrow Booleen$ t.q. $type(e,t) = true$ ssi $e \in Dom(t)$ pour tester si l'expression donnée est un `symbole` ou une `function` ;
- des fonctions Maple d'analyse des expressions `function` :
  - `nops` : $Function \rightarrow Entier$ t.q. $nops(f) = n$ ssi "l'expression fonction" f a n opérandes,
  - `op` : $Entier \times Function \rightarrow Expression$ t.q. $op(n, f) = e$ ssi e est le n^{ieme} opérande de f (cas où $1 \leq n \leq$ nombre opérandes de f), e est le "nom" de la fonction appelée par f (cas où $n = 0$).

Q 3 Testez à l'aide du prédicat Maple type si les 3 formules `impl(a,b), b, ou(et,non(a))` sont des fbf. Vérifiez également vos formules F1, F2, F3 et F4 saisies précédemment.

2 Manipulation des propositions

Q 4 Définissez le prédicat `symb?` : $Fbf \rightarrow Booleen$ t.q. $symb?(f) = true$ ssi f est réduite à un symbole propositionnel.

Q 5 Définissez le prédicat `neg?` : $Fbf \rightarrow Booleen$ t.q. $neg?(f) = true$ ssi f est une formule dont le connecteur racine est $\neg$. De même, écrivez les prédicats `ou?, et?, impl?, equi?`.

Q 6 Définissez les fonctions `ssFbf` : $\{1, 2\} \times Fbf \rightarrow Fbf$ définie pour des fbf non réduites à un symbole propositionnel t.q. $ssFbf(i, f) = f'$ (avec $i \leq$ arité du connecteur racine de f) ssi f' est la sous formule de f correspondant au sous arbre gauche de la racine si $i = 1$, au sous-arbre droit si $i = 2$.

Remarque. On dispose maintenant d'un type fbf et de fonctions de manipulation de ce type (*symb*?, *neg*?, *ssFbf*...). Dans la suite seules ces fonctions sont utiles (i.e. on **n'utilisera plus** les fonctions Maple *nops* et *op*).

3 Analyse syntaxique des propositions

Q 7 Ecrivez la fonction `nbc` : $Fbf \rightarrow Entier$ qui retourne le nombre de connecteurs d'une proposition (nombre d'occurrences).

Q 8 Ecrivez la fonction `prof` : $Fbf \rightarrow Entier$ qui retourne la profondeur de l'arbre associé à la proposition donnée.

Q 9 Ecrivez la fonction `symbProp` : $Fbf \rightarrow 2^{SP}$ qui retourne l'ensemble des symboles propositionnels d'une proposition donnée. Vous utiliserez le type `ensSP` fourni. On rappelle que les ensembles en Maple se manipule simplement en séparant les éléments par des "," et en délimitant l'ensemble par des "{" et "}" ; vous disposez des opérateurs et prédicats ensemblistes `in, union, subset...` pour l'appartenance, l'union, l'inclusion... Consultez l'aide.

4 Affichage (infixé) d'une formule (exercice optionnel)

Q 10 Ecrire une fonction `afficher` qui prend en donnée une fbf, ne retourne rien et a comme effet de bord d'afficher la fbf sous forme infixée (affichage classique). Par exemple si `F :=impl(a,non(et(b,non(c))))` `Afficher(F)` produira `(a impl non (b et non c))`. Pour l'affichage vous utiliserez la fonction `printf` (qui fonctionne comme en C). Exemple : `printf(''%s'',op(0,F))` affiche `'' impl ''`.

5 Définition d'une structure d'interprétation

On représente l'interprétation d'un symbole propositionnel en Maple par une liste à deux éléments $[p, v]$ où p est un symbole propositionnel et $v \in \{0, 1\}$ (cf. le type `coupleInt` fourni). Une **interprétation** est alors un ensemble de `coupleInt` tel que chaque symbole propositionnel n'a qu'une seule valeur associée (i.e. une interprétation est une fonction). Le type `intp`, basé sur le prédicat `intp ?` implémente cette notion.

Q 11 Définissez en Maple 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 12 Ecrivez la fonction `symbInt` : $Interpretation \rightarrow 2^{SP}$ qui retourne l'ensemble des symboles propositionnels d'une interprétation.

6 Sémantique d'une proposition

Q 13 Ecrivez le prédicat `intComp ?` : $Interpretation \times Fbf \rightarrow Booleen$ qui retourne vrai si l'interprétation donnée permet d'associer une valeur à tous les symboles propositionnels de la formule donnée (l'interprétation est alors *complète* pour la formule).

Q 14 Ecrivez la fonction `intSP` : $SP \times Interpretation \rightarrow \{0, 1\}$ 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).

Q 15 Ecrivez les fonctions d'interprétation des connecteurs : `intNon :` $\{0, 1\} \rightarrow \{0, 1\}$, `intEt :` $\{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$, `intOu, intImpl, intEqui`.

Q 16 Finalement, écrivez la fonction `valV` : $Fbf \times Interpretation \rightarrow \{0, 1\}$ qui calcule la valeur de vérité d'une formule f pour une interprétation i complète pour f.

7 Satisfiabilité et validité d'une proposition

Afin d'étudier les propriétés sémantiques des propositions, on se dote d'un type ensemble d'interprétations `ensIntp`. Cela se fait simplement par la commande : `AddType(ensIntp,'set(intp)') ;`

Pour tester la satisfiabilité d'une proposition, il faut calculer l'ensemble de ses interprétations qui ne dépend que de l'ensemble des symboles propositionnels apparaissant dans la proposition.

Remarque. Pour ce qui suit les fonctions Maple `map, ormap, andmap` peuvent être utiles : E étant un ensemble, F une fonction unaire définie sur E et P un prédicat unaire défini sur E :

- `map(F,E)` renvoie $E' = \{f(e)|e \in E\}$ (i.e. construit un ensemble composé des résultats de l'application de F à chaque élément de E). Ex. : $map(sqrt, \{16, 9, 4, 81\}))$ renvoie $\{4, 2, 3, 9\}$;
- `ormap(P,E)` renvoie `true` si l'ensemble E contient un élément vérifiant le prédicat P. Ex. : $ormap(odd?, \{2, 4, 12, 3, 6\}))$ renvoie $true$;
- `andmap(P,E)` renvoie `true` si tous les éléments de l'ensemble E vérifient le prédicat P. Ex. : $andmap(odd?, \{2, 4, 12, 3, 6\}))$ renvoie $false$.

Q 17 Ensemble des interprétations d'un ensemble de SP. Ecrire une fonction `ensInt` qui prend en donnée un ensemble de symboles propositionnels (`ensSP`) et retourne l'ensemble de toutes les interprétations (`ensIntp`) de ces symboles propositionnels. Lorsqu'il n'y a qu'un symbole propositionnel, il n'y a que 2 interprétations possibles. Si il y en a plus d'une, 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).

Q 18 Écrire un prédicat `satisfiable ?` qui retourne true si et seulement si une fbf donnée est satisfiable. Tester 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 19 Écrire un prédicat `valide ?` qui retourne true si et seulement si une fbf donnée est valide. Tester votre prédicat sur les propositions $a, \neg a, (a \vee b), ((a \vee b) \vee \neg a), F_1, F_2, F_3, F_4$.

8 Equivalence et conséquence entre propositions

Remarque. Dans ce qui suit, vous proposerez deux versions : une s'appuyant sur la définition initiale (à partir des interprétations) des notions d'équivalence et conséquence logiques et l'autre s'appuyant sur les propriétés qui lient ces notions à celles de validité et satisfiabilité. Vous en profiterez alors pour vérifier les propriétés démontrées en cours.

Q 20 Ecrire deux versions d'un prédicat `equivalentes ?` qui teste si deux fbf données sont sémantiquement équivalentes (idem elles ont les mêmes valeurs de vérité pour toutes les interprétations). Faites des tests ! En particulier $((a \vee b) \vee \neg a) \equiv \neg((c \wedge d) \wedge \neg c)$.

Q 21 Ecrire deux versions d'un prédicat `consequence2 ?` qui étant donnée 2 propositions F1 et F2, retourne true si F2 est conséquence logique de F1. Vérifiez que $a \models (a \vee b)$, $a \not\models (a \wedge b)$, $((a \vee b) \vee \neg a) \models \neg((c \wedge d) \wedge \neg c)$, $((a \wedge b) \wedge \neg a) \models (c \vee d)$.

Q 22 Étendre la fonction précédente à un prédicat `conséquence ?` 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 $\{f_1, f_2, ...f_n\} \models f$). Testez votre procédure en vérifiant si $\{a \wedge b, \neg a, b \rightarrow d\} \models c \rightarrow d$.

9 Mise sous forme conjonctive

Les 4 premières questions visent à fournir les transformations de fbf permettant un passage à la forme conjonctive. Pour ces 4 fonctions, il faut raisonner sur l'arbre syntaxique associé à la formule. La 5^{e} vise à fournir cette fonction. Attention, la 4^{e} fonction (`distOu`) est particulièrement délicate. Ex. : `distOu(et(et(a,non(b)),ou(c,ou(non(d),et(e,f)))))` doit retourner `et(et(a,non(b)),et(ou(c,ou(non(d),e)),ou(c,ou(non(d),f))))`.

Q 23 Ecrire une fonction récursive `oteEqui` 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 24 Ecrire une fonction récursive `oteImpl` 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 25 Ecrire 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 26 Ecrire 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 : $(A \vee (B \wedge C)) \quad \equiv \quad ((A \vee B) \wedge (A \vee C))$

Q 27 Ecrire alors la fonction `formeConj` qui prend en paramètre une fbf quelconque et retourne une fbf logiquement équivalente sous forme conjonctive.

10 Forme clausale

Q 28 On veut représenter une forme clausale comme un ensemble d'ensemble de littéraux. Ecrire un prédicat `litteral ?` qui quelque soit son (ses) paramètre(s) retourne vrai si et seulement si son unique paramètre est un littéral (positif ou négatif).

Q 29 A l'aide de la fonctionnalité `AddType(nouveauType,saDéfinition) ;` définir à partir du prédicat précédent le type `litteral` (cf. les définitions de types précédentes).

Q 30 Définir alors les types `clause` et `ensClauses`.

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

Q 32 Ecrire une fonction récursive `transfEnsClauses` qui prend en paramètre une fbf sous forme conjonctive et retourne l'ensemble de clauses (`ensClauses`) correspondant à cette fbf.

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

11 Méthodes de preuve

Il s'agit d'implanter l'une des méthodes vues en cours. Il est préférable d'en implanter une correctement et complètement plutôt que d'essayer de toutes mal les faire ! On s'attachera en particulier à se doter d'un jeu d'essais permettant de tester la méthode.

Q 34 Mettre en œuvre la méthode de résolution sur `ensClauses`.

Q 35 Mettre en œuvre Davis et Putnam sur `ensClauses`.

Q 36 Mettre en œuvre la méthode des Tableaux sur `fbf`.

12 Application

Q 37 Modéliser un problème en logique des propositions et résolvez-le à l'aide d'une des 3 méthodes précédentes.