Programmation Fonctionnelle I, 2008-09 Semestre 2 **TP12**

http://deptinfo.unice.fr/~roy

Le but principal du TP est d'obtenir un démonstrateur automatique basé sur l'algorithme de Wang qui fonctionne !

L'ALGORITHME DE WANG

Exercice 9.1 Utilisez sur papier l'algorithme de Wang pour vérifier si la formule ci-dessous est ou non un théorème :

`[(p => ¬q) => q] => [p ou q]`

Exercice 9.2 a) Complétez la définition de la fonction `(reformat fbf)` du cours page 17. Elle retourne une *fbf* dans laquelle toutes les implications ont été éliminées.

b) Idem avec la fonction `(intersection? L1 L2)` qui teste si les listes `L1` et `L2` ont au moins un élément en commun.

c) Programmez une fonction `(first-molecule Lfbf)` prenant une liste de formules `Lfbf` et retournant la première molécule de cette liste, ou bien `#f` si cette liste ne contient que des atomes. Exemple :

`(first-molecule '(p q (non r) s (p => q)))` → `(non r)`

`(first-molecule '(p q r))` → `#f`

c) Programmez une fonction `(remove x L)` retournant une copie de la liste `L` dans laquelle la première occurrence de l'élément `x` aura été supprimée. Exemple :

`(remove 'c '(a b c d c e))` → `(a b d c e)`

d) Programmez une fonction `(add x L)` retournant une copie de la liste `L` dans laquelle l'élément `x` a été rajouté s'il n'y était pas déjà.

Exercice 9.3 Complétez la fonction `(wang LHS RHS)` du cours page 18.

Exercice 9.4 a) Programmez la fonction `(fbf? obj)` prenant un objet Scheme quelconque `obj` et retournant `true` si et seulement si `obj` est une formule bien formée. Suivez bien la grammaire des *fbf*…

b) Complétez la fonction `(théorème? fbf)` du cours page 19. Et voilà !…

Le TP fournit gracieusement une fonction interactive (prover-loop) demandant à l'utilisateur d'entrer une fbf et lui disant s'il s'agit ou non d'un théorème. Regardez au passage comment on gère un dialogue !

EXERCICES COMPLÉMENTAIRES

Exercice 9.5 a) Quelle est la table de vérité de l'opérateur `nand` ? Ecrire la fonction `(nand p q)` prenant deux booléens `p` et `q`. Il est interdit d'utiliser `not`, `and` et `or`.

b) Définissez les fonctions `($not p)`, `($and p q)`, `($or p q)` en utilisant `nand`. Ceci prouvera *l'universalité* du circuit `nand`.

Exercice 9.6 Ecrire la table de vérité de la formule `q ⇒ [p ∨ (¬q ∧ p)]`. Est-ce un théorème ?

Exercice 9.7 Programmez une fonction `(evalb fbf AL)`. Elle prend une formule `fbf` et une A-liste `AL` donnant les valeurs de ses variables, et retournant la valeur de la formule avec ce jeu de variables. Exemple :

`(evalb '(q => (p ou ((non q) et p))) '((p #f) (q #t)))` → `#f`

• *Le problème suivant se propose de programmer la fonction* `(valide? fbf)` *avec les tables de vérité, ce qu'évite l'algorithme de Wang !*

Exercice 9.8 a) Ecrire une fonction `(variables fbf)` retournant l'ensemble [donc sans répétitions] des *variables* présentes dans une `fbf`. Exemple : `(variables '(q => (p ou ((non (q ou r)) et p))))` → `(q r p)`

b) Ecrivez une fonction `(gen-bools n)` prenant un entier $n \geq 0$ et retournant la liste de toutes les listes de longueur `n` ne contenant que `true` et `false`. La liste solution contient 2^n éléments. Exemple :

```
> (gen-bools 3)
((true true true) (true true false) (true false true) ... (false false false))
```

On rappelle que `true` *s'écrit aussi* `#t` *et* `false` *s'écrit aussi* `#f`.

c) Ecrivez une fonction `(un-contre-exemple fbf)` retournant la liste vide `()` si `fbf` est un thèorème, ou bien un contre-exemple si ce n'en est pas un. Exemple [cf exo 6.2] :

```
(un-contre-exemple '(q => (p ou ((non q) et p)))) → ((q #t) (p #f))
```

qui montre que si q est vrai et si p est faux, alors la formule est fausse, et n'est donc pas un théorème.

d) En déduire une fonction `(valide? fbf)` retournant #t si et seulement si `fbf` est un théorème. Exemple :

```
(valide? '(p => (p et q))) → #f
(valide? '(p => (p ou q))) → #t
```

Si vous avez tout fait, chapeau bas !