Devoir de Programmation Fonctionnelle

I) Sujet du devoir

On désire implémenter en `Ocaml` un algorithme permettant de déterminer si une formule propositionnelle donnée est une tautologie (*i. e.* si elle est vraie pour toutes les distributions de valeur de vérité possibles). L'algorithme utilise deux types et procède en trois étapes.

Les deux types

Ils se nomment `fp` (pour « formule propositionnelle ») et `sas` (pour « si alors sinon »). Chacun de ces types contient une infinité de variables, deux constantes « vrai » et « faux », et un certain nombre de connecteurs. Le type `fp` contient les cinq connecteurs classiques (« et », « ou », « implique », « équivalent », « non ») tandis que le type `sas` ne contient qu'un connecteur nommé `Si`. Ce dernier attend 3 arguments et s'interprète ainsi : `Si(a,b,c)` signifie « Si a est vrai Alors b Sinon c ». (Attention : il s'agit d'une nouvelle structure de données et pas de l'habituelle structure de contrôle !) Voici la définition de ces deux types :

```
# type fp = X of int | Vrai1 | Faux1 | Et of fp * fp | Ou of fp * fp | Imp
of fp * fp | Equiv of fp * fp | Non of fp ;;

# type sas = Y of int | Vrai2 | Faux2 | Si of sas * sas * sas ;;
```

Les trois étapes

Les trois étapes de l'algorithme sont les suivantes :
1) Transformation de la formule initiale **F** de type `fp` en une formule équivalente **G** de type `sas`.
2) Mise en forme normale de la formule **G** en une nouvelle formule **H** de type `sas` équivalente.
3) Détermination du fait que la formule **H** est ou non une tautologie.

Détaillons à présent ces trois étapes.
1) On transforme la formule initiale **F** de type `fp` en une formule **G** de type `sas` qui lui est logiquement équivalente. En effet, toute variable `X(i)` se traduit en une variable `Y(i)`, chaque constante se traduit « en elle-même » et chacun des cinq connecteurs classiques se traduit en une formule de type `sas` qui n'utilise que `Si` et les constantes. Ainsi, la formule `Non(X(1))` se traduit en la formule `Si(Y(1),Faux2,Vrai2)`. On trouvera de la même manière les traductions des autres connecteurs classiques en formules qui ne contiennent que `Si` et les constantes. On sera alors en mesure de réaliser une fonction `OCAML trad` qui traduit uniformément toute formule de type `fp` en une formule équivalente de type `sas`.

2) On désire mettre la formule **G** en forme normale, c'est-à-dire transformer la formule **G** en une formule **H** qui lui est logiquement équivalente mais qui possède une forme particulière : le premier argument de tout `Si` est soit une variable soit une constante mais jamais un `Si`. Pour se débarrasser du cas gênant (un `Si` en premier argument d'un `Si`), l'on peut remarquer que :

`Si(Si(a,b,c),d,e)` est équivalent à `Si(a,Si(b,d,e),Si(c,d,e))`

Il suffit de répéter cette transformation sur tous les `Si` pour obtenir la forme normale désirée. On définira donc une fonction `OCAML mefn` (pour « mise en forme normale ») qui traduit uniformément la formule **G** obtenue à la première étape en une nouvelle formule **H** en forme normale.

3) On désire maintenant déterminer si la formule **H** est ou non une tautologie. Il faut pour cela essayer toutes les distributions de valeur de vérité possibles et vérifier que, pour chacune d'entre elles, l'évaluation de la formule **H** donne « vrai » et jamais « faux ». Pratiquement, on ne remplacera pas une variable `Y(i)` par sa valeur de vérité (« vrai » ou « faux ») dans la formule **H** mais on utilisera la notion d'environnement. Un environnement est une liste de couples de type `int * bool`. Initialement, l'environnement est vide. Par la suite, la présence d'un couple `(i,b)` dans cette liste signifie que la variable `Y(i)` s'est vue assignée la valeur de vérité `b` (les valeurs de vérité « vrai » et « faux » sont représentées par les valeurs `OCAML true` et `false`). Pour savoir si **H** est une tautologie, il faut définir une fonction `OCAML eval` qui prend en entrée une formule `f` et un environnement `e`, et dont la valeur `eval f e` est ainsi définie :

Si `f` est la constante `Vrai2`
Alors `true`
Sinon Si `f` est la constante `Faux2`
  Alors `false`
  Sinon Si `f` est une variable `Y(i)`
    Alors Si il existe dans `e` un couple `(i,b)`
      Alors `b`
      Sinon `false`
    Sinon `f` est nécessairement de la forme `Si(g,h,k)`
      Si `g` est la constante `Vrai2`
      Alors `eval h e`
      Sinon Si `g` est la constante `Faux2`
        Alors `eval k e`
        Sinon Si `g` est une variable `Y(i)`
          Alors Si il existe dans `e` un couple `(i,b)`
            Alors Si `b` **Alors** `eval h e` **Sinon** `eval k e`
            Sinon Soit `e1` l'environnement `e` augmenté de `(i,true)`
              et `e2` l'environnement `e` augmenté de `(i,false)`
              dans `eval h e1` **et** `eval k e2`

II) Document attendu

Le devoir sera réalisé en binôme et donnera lieu à un rapport papier à rendre le mardi 22 mai à 14h. Ce document comprendra :

- Une présentation du problème et de sa solution algorithmique en pseudo-code.
- Un listing commenté des types et fonctions `Ocaml` utilisés.
- Des jeux d'essais nombreux et pertinents.

La notation prendra en compte à parts égales le contenu (*i.e.* le code) et la forme (*i.e.* le rapport), ce devoir étant autant l'occasion de rédiger un document que celle de programmer en `Ocaml`.