Travaux Pratiques de OCaml - TP 1

Validité d'une formule en logique des propositions

On rappelle qu'une formule est définie inductivement comme suit :

1- tout atome est une formule.

2- si A et B sont des formules alors $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$ sont des formules.

3- toute formule est obtenue par un nombre fini d'applications des règles ci-dessus.

On définit un type Fbf :

```
type Fbf =
      | At of char
      | Non of FbF
      | Et of  FbF * FbF | Ou of  FbF * FbF | Imp of  FbF * FbF
      | Equiv of  FbF * FbF;;
```

La définition du type garantit qu'on ne manipule que des formules syntaxiquement correctes.

Exemple : L'expression

Imp(Ou(Ou(At `a`, Non(Et(At `b`, Non(At `c`)))), At `d`), Ou(Non(Non(At `a`)), At `e`))

représente la formule totalement parenthésée

$(((a \vee (\neg(b \wedge (\neg c)))) \vee d) \rightarrow ((\neg (\neg a)) \vee e))$.

On représente les valeurs de vérité *vrai* et *faux* par true et false. On représente une interprétation d'une formule par une liste de valeurs de vérité dont la longueur est le nombre d'atomes de la formule.

Exemple : Pour la formule

Imp(Ou(Ou(At `a`, Non(Et(At `b`, Non(At `c`)))), At `d`), Ou(Non(Non(At `a`)), At `e`))

- l'ensemble des atomes de la formule est représenté par la liste des caractères : [`a` ; `b` ; `c` ; `d` ; `e`] ,
- l'interprétation [true ; true ; false ; true ; false] correspond aux valuations respectives des atomes a, b, c, d, e à *vrai, vrai, faux, vrai, faux.*

On utilisera les procédures non, imp, et, ou, equiv définies dans le fichier *auxTp1.ml (/users/linfg/bodeveix/TpCaml/TP1/auxTp1.ml)*

1- Valeur d'une formule dans une interprétation.

1.1. Ecrire la fonction **valeur** qui, étant donnés l'ensemble des atomes d'une formule, l'un de ces atomes et une interprétation, retourne la valeur de vérité associée à l'atome dans l'interprétation.

1.2. Ecrire la fonction **applique** qui étant donnés l'ensemble des atomes d'une formule, une interprétation et la formule, retourne la valeur de la formule dans cette interprétation, telle que :

```
# applique [`a`;`b`;`c`] [true;true;false] (Ou(At `a`, Et(At `c`, At `a`))) ;;
- : bool = true
# applique [`a`;`b`;`c`] [false;true;false] (Ou(At `a`, Et(At `b`, At `c`))) ;;
- : bool = false
```

2- Construction de toutes les interprétations d'une formule.

2.1. Ecrire la fonction **ensat** qui construit l'ensemble des atomes qui composent une formule. On utilisera les fonctions sur les ensembles définies dans le fichier *auxTP1.ml*.

2.2. Ecrire la fonction **première_int** qui construit, pour l'ensemble des atomes d'une formule donnée, une première interprétation constituée uniquement de la valeur de vérité *faux*.

2.3. Ecrire la fonction **int_suivante** qui permet de passer d'une interprétation quelconque à une autre interprétation dite "suivante" (dans le sens "la ligne suivante dans la table de vérité"), telle que :

```
# int_suivante [true ; false ; false ; true] ;;
- : bool list = [false ; true ; false ; true]

# int_suivante [true ; true ; true ; true] ;;
- : bool list = [false ; false ; false ; false]
```

Cette opération équivaut à une addition binaire, dans laquelle on placera (pour simplifier) les bits de plus faibles poids à gauche. L'utilisation répétée de cette fonction permettra de construire (si nécessaire) toutes les interprétations d'une formule.

3- Validité.

Ecrire la fonction **valide** qui détermine si son argument est une formule valide ou non.

Remarque : Pour faciliter la lecture des formules construites avec le type `Fbf`, on trouvera dans le fichier `auTp1.ml` une fonction d'affichage et des exemples d'utilisation.