L2/LD — TP 3 — Formules propositionnelles avec variables

1 Introduction

L'objectif de ce TP est d'implanter en OCaml la notion de formule de la logique des propositions.

La fonction `string_of_bool` OCaml convertit un booléen en chaîne de caractères, ce qui permet de l'afficher, avec la fonction `print_string`.

À taper :	`let p = false;;`
Réponse :	*`val p : bool = false`*
À taper :	`string_of_bool p;;`
Réponse :	*`- : string = "false"`*
À taper :	`print_string (string_of_bool p);;`
Réponse :	*`false- : unit = ()`*

2 Représentation de formules propositionnelles avec variables

Il a été vu dans le TP précédent comment définir un type pour représenter des formules propositionnelles sans variables. Nous souhaitons ici élargir ce type pour pouvoir représenter des formules propositionnelles avec variables. Il s'agit de formules construites à partir de symboles de propositions atomiques, appelées aussi *variables propositionnelles*. L'ensemble des formules propositionnelles est alors défini comme le plus petit ensemble tel que :

- Vrai et Faux sont des formules propositionnelles ;
- si P est un symbole de proposition atomique alors P est une formule propositionnelle ;
- si P est une formule propositionnelle alors $\neg P$ est une formule propositionnelle ;
- si P et Q sont des formules propositionnelles alors $P \wedge Q$ et $P \vee Q$ sont des formules propositionnelles.

On admet qu'un symbole de proposition atomique est n'importe quelle chaîne de caractères. On définit donc le type OCaml `ap`[1] suivant :

```
type ap = string;;
```

A partir de ce type `ap`, on définit un deuxième type OCaml, le type `pf` des formules propositionnelles selon la définition précédente, de la manière suivante :

```
type pf = Vrai
        | Faux
        | Atome of ap
        | Neg of pf
```

1. `ap` abrège "Atomic Proposition".

```
          | Et of pf * pf
          | Ou of pf * pf
;;
```

Ensuite, on peut, par exemple, représenter la formule propositionnelle $f = P \wedge (\text{FAUX} \vee (\text{VRAI} \wedge Q))$ par l'expression OCaml :

```
Et(Atome("P"), Ou(Faux, Et(Vrai, Atome("Q"))))
```

1. Dans un fichier `tp3.ml`, déclarer les types `ap` et `pf`.
2. Pour tester le type `pf`, définir la formule propositionnelle $f = P \wedge (\text{FAUX} \vee (\text{VRAI} \wedge Q))$. Si le type a été correctement défini, vous obtenez le résultat suivant :

   ```
   val f : pf = Et(Atome("P"),Ou(Faux,Et(Vrai,Atome("Q"))))
   ```

3. Ecrire une fonction récursive `affichePF : pf -> unit` qui affiche une formule de type `pf` de manière infixée, sans afficher les parenthèses les plus extérieures. Par exemple, avecla formule propositionnelle `f` définie précédemment, on doit avoir :

À taper :	`affichePF f;;`
Réponse :	`P ^ (Faux v (Vrai ^ Q))- : unit = ()`

4. Définir les formules propositionnelles suivantes :
   — $f1 = (\text{VRAI} \wedge \text{FAUX}) \vee (\text{VRAI} \wedge (\text{FAUX} \wedge ((\text{VRAI} \wedge \text{VRAI}) \wedge \text{FAUX})))$
   — $f2 = (((\text{P} \wedge \text{FAUX}) \wedge (\text{FAUX} \vee \text{VRAI})) \wedge ((\text{P} \vee \text{VRAI}) \vee (\text{VRAI} \wedge \text{Q}))) \wedge \text{VRAI}$
   Vérifiez que vous obtenez bien les résultats suivants :

À taper :	`affichePF f1;;`
Réponse :	`(Vrai ^ Faux) v (Vrai ^ (Faux ^ ((Vrai ^ Vrai) ^ Faux)))- : unit = ()`
À taper :	`affichePF f2;;`
Réponse :	`(((P ^ Faux) ^ (Faux v Vrai)) ^ ((P v Vrai) v (Vrai ^ Q))) ^ Vrai- : unit = ()`

3 Valuation

Pour interpréter une formule propositionnelle définie par les types OCaml `ap` et `pf`, il est nécessaire de valuer les propositions atomiques de cette formule.

1. Définir une fonction `valuation1 : ap -> bool` retournant respectivement les valeurs `true` et `false` pour les propositions atomiques `P` et `Q`, en déclenchant une exception lorsque la proposition atomique passée en paramètre n'est pas définie.

   Indication : Le mot-clé `failwith` pour déclencher une exception a été vu dans le TP 1.
2. De même, définir une fonction `valuation2` retournant la valeur `true` pour les propositions atomiques `P` et `Q`, en déclenchant une exception lorsque la proposition atomique passée en paramètre n'est pas définie.

Ces fonctions serviront d'exemples pour tester une fonction dans la partie 5.

4 Fonction ayant en argument une fonction

Une fonction peut être l'argument d'une autre fonction. On peut ainsi définir la fonction `ffois2` qui prend en argument une valeur `x` et une fonction `f` et retourne `2*f(x)` :

```
À taper :   let ffois2 = function
            (x,f) -> f(x)*2
Réponse :   val ffois2 : 'a * ('a -> int) -> int = <fun>
À taper :   ffois2 (4,(function x -> x+3));;
Réponse :   - : int = 14
```

1. De même, définir une fonction `ff` qui prend en argument une variable `x` et une fonction `f` et qui retourne `f(x)`.
2. Donner et observer le résultat des instructions suivantes :

```
À taper :   ff (4,(function x -> 2*x));;
À taper :   ff (true,(function x -> true && false));;
```

5 Interprétation de formules propositionnelles

1. Définissez une fonction `interpretation : pf * (ap -> bool) -> bool` permettant de réaliser l'interprétation booléenne d'une formule propositionnelle à partir d'une valuation donnée en paramètre.
2. Testez la fonction `interpretation` avec
   (a) la formule `f` et la valuation `valuation1`
   (b) la formule `f` et la valuation `valuation2`
   (c) la formule `f1` et la valuation `valuation1`
   (d) la formule `f1` et la valuation `valuation2`
   (e) la formule `f2` et la valuation `valuation1`
   (f) la formule `f2` et la valuation `valuation2`

   Vérifiez que vous obtenez bien les résultats suivants :

```
À taper :   interpretation (f,valuation1);;
Réponse :   - : bool = false
À taper :   # interpretation (f,valuation2);;
Réponse :   - : bool = true
À taper :   # interpretation (f1,valuation1);;
Réponse :   - : bool = false
À taper :   # interpretation (f2,valuation1);;
Réponse :   - : bool = false
À taper :   # interpretation (f1,valuation2);;
Réponse :   - : bool = false
À taper :   # interpretation (f2,valuation2);;
Réponse :   - : bool = false
```

3. Définissez la formule propositionnelle $f3 = \textsc{Faux} \vee ((\mathrm{R} \wedge \textsc{Vrai}) \wedge \mathrm{P})$. Vérifiez que l'application de la fonction `interpretation` à la formule `f3` et à la valuation `valuation1` lève bien une exception.