L2/LD – TP 2 – Formules propositionnelles sans variables

1 Introduction

L'objectif de cette série de TP est d'implanter en OCaml la notion de formule de la logique des propositions. Dans ce TP, nous allons nous restreindre aux formules propositionnelles sans variables.

Rappel : OCaml inclut les opérateurs `&&`, `||` et `not` sur le type `bool`.

```
À taper :   true && false;;
Réponse :   - : bool = false
À taper :   true || false;;
Réponse :   - : bool = true
À taper :   not false;;
Réponse :   - : bool = true
```

2 Représentation de formules propositionnelles sans variables

Les formules propositionnelles sans variables sont seulement construites à partir de deux constantes notées Vrai et Faux.

L'ensemble des formules propositionnelles sans variables est le plus petit ensemble tel que :
- Vrai et Faux sont des formules propositionnelles sans variables ;
- si E est une formule propositionnelle sans variables, alors $\neg E$ est une formule propositionnelle sans variables ;
- si E et F sont des formules propositionnelles sans variables, alors $E \wedge F$ et $E \vee F$ sont des formules propositionnelles sans variables.

Pour représenter toutes ces formules propositionnelles, on définit le type 0Caml `cpf` [1] suivant :

```
type cpf =
      Vrai
    | Faux
    | Neg of cpf
    | Et of cpf * cpf
    | Ou of cpf * cpf;;
```

Ensuite, on peut par exemple représenter la formule propositionnelle e = Vrai $\wedge$ (Faux $\vee$ Vrai) par l'expression OCaml

```
Et(Vrai, Ou(Faux, Vrai));;
```

1. Créer un fichier `tp2.ml` et recopier dedans le type OCaml que nous venons de définir pour la représentation des formules propositionnelles sans variables.
2. Dans ce fichier, définir la formule propositionnelle `e` pour tester le type `cpf`. Si le type a été correctement défini, vous devriez obtenir le résultat suivant :

```
e : cpf = Et (Vrai, Ou (Faux,Vrai))
```

1. `cpf` abrège "Constant Propositional Formula".

3 Affichage infixé

Comme vous avez pu le voir en évaluant la formule propositionnelle `e`, l'interpréteur OCaml affiche le résultat à l'aide des constructeurs du type `cpf`. Cette présentation est dite "préfixée".

En général, on lui préfère une présentation "infixée" plus lisible. Nous allons donc définir une fonction d'affichage infixé.

En OCaml, la fonction pour afficher une chaîne est `print_string : string -> unit`.

À taper : `print_string "coucou";;`
Réponse : *`coucou- : unit = ()`*

`print_string x` affiche la valeur de la variable `x`.

À taper : `let ch = "coucou";;`
Réponse : *`val ch : string = "coucou"`*
À taper : `print_string ch;;`
Réponse : *`coucou- : unit = ()`*

Pour enchaîner les affichages, il suffit de séparer les appels à cette fonction par un point-virgule (`;`).

1. Ecrire une fonction récursive `afficheFormuleV1` qui permet d'afficher une formule selon la notation infixée, en plaçant des parenthèses autour des formules composées. Par exemple, la formule propositionnelle `e` définie précédemment sera affichée comme suit :

   `(Vrai` $\wedge$ `(Faux v Vrai))`

2. Cette fonction affichage n'est pas satisfaisante car la formule propositionnelle `e` devrait être affichée sans parenthèses extérieures comme suit :

   `Vrai` $\wedge$ `(Faux v Vrai)`

   Écrire une fonction récursive `afficheFormule` qui permet d'afficher une formule selon la notation infixée, en plaçant des parenthèses autour des formules composées lorsque c'est nécessaire.

   Pour cela, une des nombreuses solutions envisageables consiste à reprendre la fonction d'affichage précédente et à l'augmenter d'un argument booléen signifiant la présence ou non de parenthèses. Les appels récursifs de cette nouvelle fonction devront alors afficher les parenthèses alors que la fonction `afficheFormule` doit y faire appel une première fois sans affichage des parenthèses.

3. Définir les formules propositionnelles suivantes :
   — e1 = (`Vrai` $\wedge$ `Faux`) $\vee$ (`Vrai` $\wedge$ (`Faux` $\wedge$ ((`Vrai` $\wedge$ `Vrai`) $\wedge$ `Faux`)))
   — e2 = ($\neg$ (`Faux` $\vee$ `Vrai`) $\vee$ (`Vrai` $\wedge$ `Vrai`)) $\wedge$ `Vrai`
   — e3 = $\neg$(`Vrai` $\wedge$ `Faux`) $\vee$ (`Vrai` $\wedge$(`Faux` $\wedge$ ((`Vrai` $\wedge$ `Vrai`) $\wedge$ `Faux`)))

4. Vérifier qu'on obtient bien les résultats suivants, lorsqu'on applique la fonction `afficheFormule` à ces formules :

   À taper : `afficheFormule e1;;`
   Réponse : *`(Vrai ^ Faux) v (Vrai ^ (Faux ^ ((Vrai ^ Vrai) ^ Faux)))- : unit = ()`*
   À taper : `afficheFormule e2;;`
   Réponse : *`(Non(Faux v Vrai) v (Vrai ^ Vrai)) ^ Vrai- : unit = ()`*
   À taper : `afficheFormule e3;;`
   Réponse : *`Non(Vrai ^ Faux) v (Vrai ^ (Faux ^ ((Vrai ^ Vrai) ^ Faux)))- : unit = ()`*

4 Evaluation de formules propositionnelles

Les formules propositionnelles que nous considérons dans ce TP sont constituées uniquement des constantes `Vrai`, `Faux` et des connecteurs $\neg$, $\wedge$ et $\vee$. Par conséquent, on peut leur associer la valeur booléenne `true` ou `false` du type `bool` des booléens en OCaml. Dans ce but, il suffit

d'évaluer `Vrai` par `true`, `Faux` par `false` et d'évaluer chaque connecteur selon la table de vérité de l'opérateur booléen correspondant.

Si, par exemple, on évalue la formule propositionnelle `e`, on trouve pour résultat la valeur booléenne `true`.

1. Ecrire une fonction récursive `evalueFormule : cpf -> bool` qui permet d'évaluer une formule propositionnelle de type `cpf`.
2. Tester la fonction `evalueFormule` à l'aide des formules propositionnelles `e1` et `e2` définies précédemment. Le résultat de ces tests doit être le suivant :

À taper :	`evalueFormule e1;;`
Réponse :	`- : bool = false`
À taper :	`evalueFormule e2;;`
Réponse :	`- : bool = true`
À taper :	`evalueFormule e3;;`
Réponse :	`- : bool = true`

5 Extra

Cette dernière partie s'adresse aux étudiants qui ont terminé rapidement les questions précédentes, ainsi qu'à ceux qui souhaitent approfondir le sujet ou réviser ultérieurement.

1. Ajouter au type `cpf` le connecteur $\Rightarrow$.
2. Etendre la fonction `afficheFormule` de manière à traiter le connecteur $\Rightarrow$.
3. Etendre la fonction `evalueFormule` de manière à traiter le connecteur $\Rightarrow$.
4. Définir les formules propositionnelles suivantes afin de pouvoir tester l'ensemble de vos modifications :
   — $((\texttt{Vrai} \Rightarrow \texttt{Faux}) \Rightarrow (\texttt{Vrai} \vee \texttt{Faux}))$
   — $(\texttt{Vrai} \wedge (\texttt{Vrai} \Rightarrow (\texttt{Faux} \vee (\texttt{Faux} \Rightarrow \texttt{Vrai})))) \wedge \texttt{Vrai}$
   — $\texttt{Faux} \vee (\texttt{Vrai} \Rightarrow (\texttt{Vrai} \wedge (\texttt{Vrai} \Rightarrow (\texttt{Vrai} \wedge \texttt{Faux}))))$
   Ces formules propositionnelles sont respectivement logiquement équivalentes à `Vrai`, `Vrai` et `Faux`.

6 Préparation du TP suivant

S'inspirer du type `cpf` pour définir un type `PF` de formules propositionnelles **avec** variables propositionnelles. Compléter ce nouveau type avec une fonction d'affichage infixé.

L'évaluation booléenne de ces formules sera étudiée dans le prochain TP, mais il n'est pas interdit de réfléchir à la manière de procéder.