Option Informatique

TD $n°1$: prise en main de Caml (sans les mains)

Un peu de pratique et de théorie

Exercice 3. *Mots bien parenthésé*

On considère les deux symboles de parenthèse ouvrante "(" et de parenthèse fermante ")". On appelle **mot parenthésé** une suite finie de parenthèses (ouvrantes ou fermantes). Les mots parenthésés seront codés en Caml par des chaînes de caractères.

On appelle **longueur** d'un mot parenthésé le nombre de parenthèses qui forment le mot. Ainsi :

- Il y a un unique mot parenthésé de longueur 0 : le mot vide ; on le note ε.
- Il y a deux mots parenthésés de longueur 1 : les mots "(" et ")".
- Il y a quatre mots parenthésés de longueur 2 : les mots "((", "()", ")(" et "))".
- Pour tout entier n, il y a 2^n mots parenthésés de longueur n.

Si u et v sont deux mots parenthésés, on appelle **concaténation de** u **et** v et on note uv le mot obtenu en mettant bout à bout u et v. Si par exemple on a $u =$"()" et $v =$")" alors on a $uv =$"())". Lorsqu'on a $w = uv$, on dit que u est un **préfixe** de w.

Un mot parenthésé u est dit **bien parenthésé** lorsqu'il a autant de parenthèses ouvrantes que de parenthèses fermantes et lorsque tout préfixe de de u a au moins autant de parenthèses ouvrantes que de parenthèses fermantes.

1. Combien y a-t-il de mot bien parenthésés de longueur impaire ?

2. Écrire une fonction

   ```
   bien_parenthese : string -> bool
   ```

   qui prend comme argument un mot parenthésé et retourne comme résultat le booléen indiquant si le mot est ou pas bien parenthésé.

3. Montrer qu'un mot prenthésé est bien parenthésé si et seulement si c'est ou bien le mot vide ou bien un mot de la forme $(u)v$ avec u et v des mots bien parenthésés.

4. Écrire une fonction

   ```
   decompose : string -> string*string
   ```

   qui prend comme argument un mot bien parenthésé non vide et retourne comme résultat un couple u, v donné par la question précédente.

5. On note c_n le nombre de mots bien parenthésés de longueur $2n$. Donner une relation de récurrence sur les c_n.

   Comment en calcule-t-on un expression directe du terme général ? Il paraît que cela a été fait en maths ? ?

Un peu de théorie

Exercice 4. *Le problème de l'arrêt*

1. Donner un exemple de fonction Caml `f : int -> int` qui compile mais qui ne termine jamais, c'est-à-dire telle que l'exécution de `f x` ne s'achève jamais quel que soit l'argument `x`.

Dans cet exercice, on montre qu'il n'existe pas de fonction Caml `termine : ('a -> 'b) -> bool` qui prend comme argument une fonction Caml `f` et donne comme résultat le booléen `true` si la fonction `f` termine toujours et le booléen `false` sinon. Dans la suite, on raisonne par l'absurde en supposant qu'il existe une telle fonction.

2. Que fait la fonction suivante ?

```
let rec impossible () =
        if (termine impossible) then impossible ()
        else true;;
```

3. Conclure.