TP de programmation n° 7

Julien Reichert

Mardi 10 décembre 2013

Récursivité terminale

. . . et TP final !

1 Définition

Une fonction récursive est dite récursive terminale si, pour chaque appel, elle retourne directement le résultat de l'appel récursif.

Il s'agit souvent de faire des effets de bord ou de travailler directement sur les arguments lors des appels.

Par exemple, la fonction factorielle définie ainsi est récursive terminale (plus précisément, la fonction récursive auxiliaire utilisée pour définir la factorielle est récursive terminale) :

```
let factorielle n =
let rec auxiliaire resultat = function
0 -> resultat
| n -> auxiliaire (n * resultat) (n - 1)
in auxiliaire 1 n;;
```

mais définie ainsi, elle ne l'est pas, car le résultat de l'appel récursif est mutiplié avant d'être retourné :

```
let rec factorielle n = match n with
0 -> 1
|n -> n * factorielle (n-1);;
```

L'impact d'utiliser la récursivité terminale est apparent sur la pile d'appels. Dans le premier cas, quand on demande `factorielle 5`, la pile d'appels contient successivement :

```
auxiliaire 1 5
auxiliaire 5 4
auxiliaire 20 3
auxiliaire 60 2
auxiliaire 120 1
auxiliaire 120 0
120
```

et dans le deuxième cas, sa taille augmente au fur et à mesure des appels :

```
factorielle 5
5 * factorielle 4
5 * (4 * factorielle 3)
5 * (4 * (3 * factorielle 2))
5 * (4 * (3 * (2 * factorielle 1)))
5 * (4 * (3 * (2 * (1 * factorielle 0))))
5 * (4 * (3 * (2 * (1 * 1))))
5 * (4 * (3 * (2 * 1)))
5 * (4 * (3 * 2))
5 * (4 * 6)
5 * 24
120
```

En soi, la complexité en temps n'a pas l'air affectée, mais la récursivité terminale permet d'échapper plus facilement aux dépassements de pile, comme les fonctions itératives. Le compilateur d'OCaml gère d'ailleurs très bien le passage de l'un à l'autre.

2 En parlant de compilateur...

À l'oral, un écueil à éviter est de parler de compilation quand il faudrait parler d'interprétation (et vice-versa).

Ce qui est utilisé, surtout en modélisation, est un interpréteur, c'est-à-dire un programme dans lequel vous entrez vos lignes de code et qui retourne directement le résultat.

Compiler un programme, c'est transformer son code source en du code source

d'un autre langage, souvent du langage machine, lorsque l'on produit un exécutable. Cette opération est effectuée par un compilateur, et le principal compilateur pour OCaml est `ocamlc`.

Concrètement, si jamais vous voulez compiler votre code source en un programme exécutable `sortie`, écrivez dans un terminal ouvert dans le dossier qui contient le code `programme.ml` :

```
ocamlc -o sortie programme.ml
```

Ce programme sera utilisable en faisant `./sortie`, toujours dans le même dossier. Si vous le transférez sur un autre ordinateur, celui-ci aura en théorie besoin d'avoir OCaml installé.

Pour pallier ceci, il est possible de compiler en code natif, par l'instruction `ocamlopt` au lieu de `ocamlc`. Dans ce cas, l'architecture des ordinateurs doit a priori être compatible, mais OCaml ne sera plus nécessaire.

3 Exercices

Exercice 1 : Écrire une fonction récursive terminale qui calcule les termes de la suite de Fibonacci.

Exercice 2 : Écrire une fonction qui prend en entrée deux entiers naturels `n` et `b` et qui retourne la taille de `n` en base `b`, puis une variante qui somme cette valeur de 1 à `n`. Faire le test pour un million et sept.

Exercice 3 : Écrire une fonction qui imprime le parcours postfixe d'un arbre binaire.

Exercice 4 : Soit la relation $\mathcal{R}$ définie sur les mots écrits dans un alphabet quelconque par $x\mathcal{R}y$ si et seulement si on peut écrire x comme uvw et y comme $u\tilde{v}w$, où $\tilde{v}$ est v écrit à l'envers. Écrire une fonction qui décide en temps linéaire si une paire de mots sont en relation par $\mathcal{R}$.