Circuits

September 13, 2017

1 Table of Contents

2 Texte d'oral de modélisation - Agrégation Option Informatique

2.1 Préparation à l'agrégation - ENS de Rennes, 2016-17

- *Date* : 29 mai 2017
- *Auteur* : Lilian Besson
- *Texte*: Circuits (public2010-D1)

2.2 À propos de ce document

- Ceci est une *proposition* de correction, partielle et probablement non-optimale, pour la partie implémentation d'un texte d'annale de l'agrégation de mathématiques, option informatique.
- Ce document est un notebook Jupyter, et est open-source sous Licence MIT sur GitHub, comme les autres solutions de textes de modélisation que j'ai écrite cette année.
- L'implémentation sera faite en OCaml, version 4+ :

```
In [1]: Sys.command "ocaml -version";;
```

```
The OCaml toplevel, version 4.04.2
```

```
Out[1]: - : int = 0
```

Notez que certaines fonctions des modules usuels `List` et `Array` ne sont pas disponibles en OCaml 3.12. J'essaie autant que possible de ne pas les utiliser, ou alors de les redéfinir si je m'en sers.

2.3 Question de programmation

La question de programmation pour ce texte était donnée en page 6, et était assez courte :

> "Écrire un programme prenant en entrée deux nombres a, b, représentés par des écritures en base B avec des chiffres signés entre $-\beta$ et β, et retournant un résultat ternaire indiquant si $a < b$, si $a = b$ ou si $a > b$. On supposera que $\beta < B < 2\beta$, et que les écritures de a et b ont même longueur."

2.4 Réponse à l'exercice requis

2.4.1 Structure de données

Pour représenter ces entiers, on stocke leur base et le tableau de leur coefficients.

```
In [2]: type entier = {
            base : int;
            t : int array
        }
        ;;

Out[2]: type entier = { base : int; t : int array; }
```

2.4.2 Exemples

```
In [3]: let quatre = {
            base = 10;
            t = [| 4 |]
        }

Out[3]: val quatre : entier = {base = 10; t = [|4|]}
```

Le "bit" de poids faible est, par convention du texte, à gauche au début du tableau :

```
In [4]: let quarantedeux = {
            base = 10;
            t = [| 2; 4 |]
        }

Out[4]: val quarantedeux : entier = {base = 10; t = [|2; 4|]}
```

Avec l'exemple du texte :

```
In [5]: let n2006 = {
            base = 10;
            t = [| 6; 0; 0; 2 |]
        }

Out[5]: val n2006 : entier = {base = 10; t = [|6; 0; 0; 2|]}


In [6]: let n2006 = {
            base = 10;
            t = [| 6; 0; 0; 2 |]
        }

Out[6]: val n2006 : entier = {base = 10; t = [|6; 0; 0; 2|]}


In [7]: let n2006_2 = {
            base = 10;
            t = [| -4; 1; 0; 2 |]
        }

Out[7]: val n2006_2 : entier = {base = 10; t = [|-4; 1; 0; 2|]}


In [8]: let n2006_3 = {
            base = 10;
            t = [| 6; 0; 0; -8; 1 |]
        }

Out[8]: val n2006_3 : entier = {base = 10; t = [|6; 0; 0; -8; 1|]}


In [9]: let n2006_4 = {
            base = 10;
            t = [| -4; 1; 0; -8; 1 |]
        }

Out[9]: val n2006_4 : entier = {base = 10; t = [|-4; 1; 0; -8; 1|]}
```

2.4.3 Fonctions intermédiaires

Caml, dans sa toute beauté, ne permet pas de calculer x^k facilement si x est entier...

```
In [12]: let rec puissance x k =
             match k with
             | 0 -> 1
             | 1 -> x
```

```
            | k when k mod 2 = 0 ->
                (* x^k = x^(k/2 * 2) = (x^2)^(k/2) *)
                puissance (x * x) (k / 2)
            | k ->
                (* x^k = x^((k-1)/2 * 2 + 1) = x * (x^2)^((k-1)/2) *)
                x * (puissance (x * x) ((k - 1) / 2))
        ;;
```

```
Out[12]: val puissance : int -> int -> int = <fun>
```

```
In [13]: puissance 10 0;;
         puissance 10 1;;
         puissance 10 2;;
         puissance 10 3;;
         puissance 10 4;;
```

```
Out[13]: - : int = 1
```

```
Out[13]: - : int = 10
```

```
Out[13]: - : int = 100
```

```
Out[13]: - : int = 1000
```

```
Out[13]: - : int = 10000
```

On va convertir un entier représenté sous cette forme en un entier machine de Caml :

```
In [14]: let valeur {base=b; t=t} =
             let v = ref 0 in
             let n = Array.length t in
             for k = 0 to n - 1 do
                 v := !v + (t.(k) * (puissance b k))
             done;
             !v
         ;;
```

```
Out[14]: val valeur : entier -> int = <fun>
```

```
In [15]: valeur quatre;;
```

```
Out[15]: - : int = 4
```

```
In [16]: valeur quarantedeux;;

Out[16]: - : int = 42


In [17]: valeur n2006;;
         valeur n2006_2;;
         valeur n2006_3;;
         valeur n2006_4;;

Out[17]: - : int = 2006


Out[17]: - : int = 2006


Out[17]: - : int = 2006


Out[17]: - : int = 2006
```

2.4.4 Résolution

Le texte ne demandait *pas* une approche particulièrement maligne, donc on va naïvement répondre à la question : on convertit les deux entiers en leur valeur, on les compare et VOILÀ.

```
In [18]: let compare_entiers (a : entier) (b : entier) : int =
             let va = valeur a in
             let vb = valeur b in
             compare va vb
         ;;

Out[18]: val compare_entiers : entier -> entier -> int = <fun>


In [19]: compare_entiers quatre quarantedeux;;
         compare_entiers quarantedeux quatre;;
         compare_entiers quatre quatre;;
         compare_entiers quarantedeux quarantedeux;;

Out[19]: - : int = -1


Out[19]: - : int = 1


Out[19]: - : int = 0
```

```
Out[19]: - : int = 0
```

```
In [20]: compare_entiers n2006 quarantedeux;;
```

```
Out[20]: - : int = 1
```

2.5 Bonus ?

FLEMME

2.6 Conclusion

Voilà pour la question obligatoire de programmation :

- c'était facile.
- on a pas essayé d'en faire plus.

  Bien-sûr, ce petit notebook ne se prétend pas être une solution optimale, ni exhaustive.