Programmation de modèles à registres en Sophus

I/Avec des billes dans des boîtes

Les modèles de Post et Wang peuvent être simulés par des boîtes quantiques, où un fermion (obligatoirement unique) peut être présent ou absent, selon la correspondance suivante :

- case cochée : particule présente
- case non cochée : boîte vide

On dit que les machines de Post et de Wang sont ***binaires***, puisque la boîte ne peut exister que dans deux états : vide ou pleine. Pour simplifier l'écriture de programmes pour ces modèles, on peut aller un peu plus loin, en ne mettant aucune limite supérieure au nombre de billes qu'on peut placer dans la boîte (ou registre). L'idée n'a rien de nouveau puisque les babyloniens mémorisaient des nombres entiers en plaçant des petits cailloux (« calculi » en latin) dans des boules creuses d'argile[1]…

Or Sophus permet aisément de simuler des boîtes contenant un nombre entier de billes, en choisissant le nombre entier comme `valeur` d'une `variable` représentant la boîte. On va voir comment, et regarder quelques exemples. Puis on abordera des algorithmes classiques dans l'optique de la construction du nombre (de la représentation des entiers aux addition, soustraction, multiplication et division euclidienne, puis la propriété archimédienne de $\mathbb{R}$ et le calcul de racines carrées, ce qui nous ramènera aux Babyloniens!).

1. Simulation de boîtes

Pour créer une boîte A contenant initialement 3 billes, on peut simplement faire

```
A = nouvelle Variable 3
```

Pour simuler une machine de Post multi-états, on a besoin d'une tableau de variables, ce qu'on peut faire avec quelque chose comme

```
bande = []
pour n dans [0..20]
    bande[n] = nouvelle Variable n
```

Dans l'exemple ci-dessus, la bande contient les entiers successifs. Mais dans les exemples ci-dessous, il y aura toujours suffisamment peu de boîtes pour qu'on n'aie pas besoin de les regrouper dans un tableau de boîtes. C'est d'ailleurs un avantage qu'il y a à ne pas limiter le contenu des registres à zéro ou une bille.

---

1 Ancêtres communes des tablettes d'argile et de la tirelire

2. Addition

Supposons qu'on aie deux boîtes A et B, contenant initialement 5 billes et 3 billes respectivement :

Pour additionner ces deux nombres, on peut utiliser l'algorithme suivant :

- Retirer une bille de la boîte B ;
- La transférer dans la boîte A ;
- Recommencer les deux opérations ci-dessus tant que c'est possible, c'est-à-dire tant que B n'est pas vide.

Voici la manière de programmer cet algorithme en Sophus :

```
A = nouvelle Variable 5
B = nouvelle Variable 3
Tant que B.valeur > 0
    diminuer B de 1
    augmenter A de 1

montrer A
```

Pour soustraire B à A, il suffit de remplacer « augmenter » par « diminuer » dans la boucle. Par ailleurs, « diminuer B de 1 » peut se résumer par « décrémenter B » et l'augmentation de A peut s'écrire « incrémenter A ».

3. Doublement

Pour doubler un nombre, on peut le représenter par des billes placées dans la boîte A dans la configuration ci-dessus ; mais cette fois-ci, la boîte B, destinée à contenir le double de A, est initialement vide. Et surtout, chaque fois qu'on enlève une bille à A, on en dépose ***deux*** dans B.

```
A = nouvelle Variable 5
B = nouvelle Variable 0
Tant que A.valeur > 0
    diminuer A de 1
    augmenter B de 1
    augmenter B de 1

montrer B
```

On verra plus bas, avec la multiplication, qu'il revient au même d'ajouter deux fois de suite une bille dans la boîte B, ou d'en ajouter deux d'un coup. Ce qui donne ce raccourci :

```
A = nouvelle Variable 5
B = nouvelle Variable 0
Tant que A.valeur > 0
    diminuer A de 1
    augmenter B de 2

montrer B
```

II/ Multiplication des entiers

1. Multiplication par 2

Le cas particulier de la multiplication par 2 a été vu ci-dessus et sera généralisé ci-dessous. La multiplication par 2 est assez importante pour qu'une instruction de Sophus y soit consacrée :

```
A = nouvelle Variable 5
doubler A
montrer A
```

2. Multiplication par 3

Une étape importante dans la construction du nombre est la prise de conscience que les deux algorithmes ci-dessous ont le même effet (ajouter 3 billes dans `A`) :

```
A = nouvelle Variable 5
3 fois faire
    incrémenter A

montrer A
```

Et

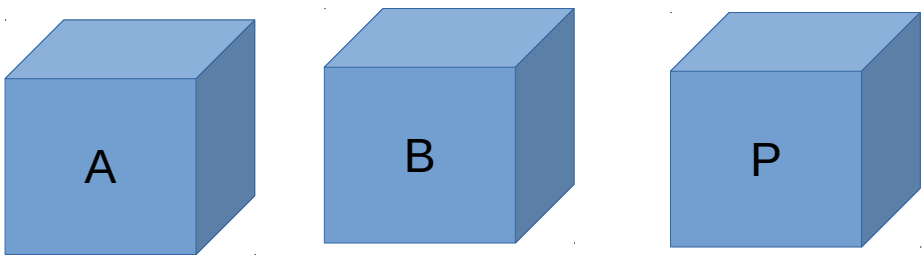
```
A = nouvelle Variable 5
augmenter A de 3

montrer A
```

On peut donc modifier l'algorithme du doublement pour avoir un algorithme de triplement : Il suffit de mettre 3 billes à la fois dans `B`, au lieu de 2...

3. Multiplication des entiers

Pour multiplier deux entiers, on aura besoin d'une troisième boîte `P` destinée à contenir à la fin le produit des deux nombres contenus dans `A` et `B` :



Voici l'algorithme de multiplication en Sophus :

```
A = nouvelle Variable 5
B = nouvelle Variable 3
P = nouvelle Variable 0

Tant que B.valeur > 0
    augmenter P de A
    décrémenter B

montrer P
```

Bien entendu, pour aller plus vite, on peut utiliser l'instruction idoine de Sophus :

```
A = nouvelle Variable 5
multiplier A par 3
montrer A
```

4. Algorithme égyptien

Les égyptiens connaissaient une méthode de multiplication basée sur l'écriture binaire de `B`, et des additions successives (en fait des doublements). `A` contient alors une suite géométrique de raison 2, et la traduction de l'algorithme égyptien en Sophus donne ceci :

```
A = nouvelle Variable 5
B = nouvelle Variable 3
P = nouvelle Variable 0
Tant que B.valeur > 0
    Si B.estImpair()
        augmenter P de A
        décrémenter B
    doubler A
    diviser B par 2

montrer P
```

III/ Division euclidienne

Toujours en Egypte avec le grec Euclide, mais d'abord un petit tour par la Sicile avec une découverte importante mais peu connue d'Archimède :

1. Propriété d'Archimède

La propriété d'Archimède est facile à énoncer, et encore plus facile à concevoir[2] :

> *Pour deux grandeurs inégales, il existe toujours un multiple entier de la plus petite, supérieur à la plus grande.*

Par exemple, bien que 2015 soit un grand nombre, en additionnant 37 à lui-même suffisamment longtemps, on finit par dépasser 2015. Que, pour tout seuil (comme 2015) choisi au départ, les multiples d'un nombre entier non nul (comme 37) finissent par dépasser le seuil, s'exprime en disant que $\mathbb{N}$ est ***archimédien***. Vu comme ça, ça a l'air tout bête, mais

- Archimède semble avoir été le premier à trouver ça ;
- La propriété archimédienne de $\mathbb{N}$ est à la base de l'existence d'une division euclidienne[3] ;
- La propriété archimédienne de $\mathbb{R}$ est à la base de l'existence de la mesure principale d'un angle en radians[4] ;
- C'est en cherchant un corps non archimédien qu'Abraham Robinson a inventé l'analyse non standard…

Pour vérifier la propriété d'Archimède sur les nombres 2015 et 37, on cherche quand les multiples de 37 dépassent 2015, avec l'algorithme suivant :

```
dividende = nouvelle Variable 2015
diviseur = 37
tant que dividende.valeur > diviseur
    diminuer dividende de diviseur
montrer dividende
```

La valeur finale du dividende est ce qu'on appelle le reste euclidien ; assorti au nombre de fois qu'on a parcouru la boucle (le quotient entier[5]), il forme la division euclidienne de 2015 par 37.

---

2 Source : L'article wikipedia

3 Voir plus bas ; c'est d'ailleurs dans les Eléments d'Euclide qu'a été rédigée la propriété énoncée ci-dessus.

4 Le premier multiple de 2π qui dépasse un angle positif, le dépasse d'une quantité qui est justement la valeur principale

5 On peut compter en ajoutant une bille à chaque passage dans la boucle, dans une boîte de comptage initialement vide. Par exemple en ajoutant au début un `compteur = new Variable 0`, et dans la boucle un `incrémenter compteur`.

Comme Sophus est une surcouche de JavaScript, le reste euclidien se note 2015%37 en Sophus comme dans beaucoup d'autres langages de programmation.
On se servira du reste euclidien pour calculer des pgcd ci-après, mais tout d'abord un cas particulier : La division par 2, qui est facile sur des nombres pairs :

2. Suite de Collatz

Le mathématicien hongrois Lothar Collatz a fait deux constatations :

- Si un nombre est pair, sa division par 2 tombe juste : Pas de reste euclidien ;
- Si au contraire un nombre est impair, il en est de même de son triple. Du coup en additionnant 1 au triple, on obtient un nombre pair.

Ces deux remarques l'ont amené à inventer la suite qui porte son nom, et qui est une suite récurrente, entièrement déterminée par son premier terme (un entier naturel non nul) :

- Compter les billes qui sont dans la boîte `u` ;
- Si ce nombre est pair, enlever à `u` la moitié des billes qu'elle contient ;
- Sinon, rajouter le double de son contenu actuel, et encore une bille supplémentaire.
- Recommencer les opérations ci-dessus jusqu'à ce qu'il ne reste plus qu'une bille dans la boîte `u`…

Pour programmer cette suite en Sophus, on rajoute un objet `t` qui est un tableau, initialement vide, et à chaque passage dans la boucle, on empile au bout de `t`, le nombre de billes que contient `u` ; ainsi `t` contient l'historique de l'expérience, et c'est `t` qu'on affiche à la fin :

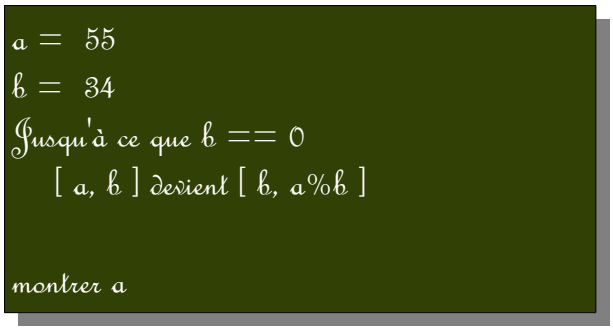
```
u = nouvelle Variable 13
t = [ ]
Tant que u.valeur > 1
    t.empiler u.valeur
    Si u.estImpair()
        tripler u
        incrémenter u
    Sinon
        diviser u par 2
montrer t
```

Aujourd'hui personne ne sait si ce programme de calcul s'arrête un jour…

Mais retour à l'Egypte antique avec Euclide, et l'application de sa division pour calculer un pgcd[6] :

3. Algorithme d'Euclide

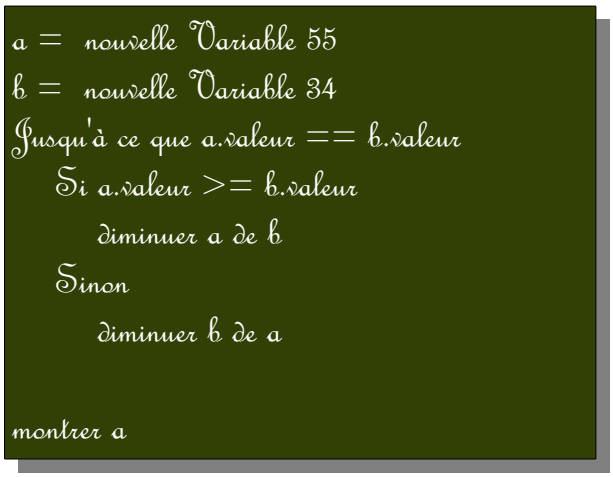
Le cas le plus long avec l'algorithme d'Euclide est celui où on calcule le pgcd de deux nombres de Fibonacci ; ceux-ci sont d'ailleurs premiers entre eux si les deux entiers sont des termes consécutifs de la suite de Fibonacci. On va donc montrer comment Sophus calcule le pgcd de 55 et 34 :



L'algorithme se résume à

- On met b dans a et le reste dans b ;
- On recommence jusqu'à ce que b soit nul ;
- a contient alors le pgcd

En combinant l'idée ci-dessus avec celle d'Archimède, on obtient l'algorithme original d'Euclide :



C'est un des plus vieux algorithmes connus. Un autre vieil algorithme est celui de Heron, que l'on soupçonne d'avoir été utilisé par les Babylobiens, et qui est encore utilisé aujourd'hui pour calculer des racines carrés :

---

6 Le problème vu avec la suite de Collatz (absence de « sinon ») empêche de programmer facilement l'algorithme original d'Euclide qui utilisait des soustractions successives avec tests d'égalité des deux nombres transformés. De toute manière l'algorithme d'Euclide avec divisions euclidiennes est nettement plus rapide en nombre de boucles. C'est donc celui qu'on va programmer ici.

IV/ Racines carrées

Retour à Babylone : Une tablette y a été trouvée, comprenant une valeur approchée de la diagonale d'un carré unité, assez précise pour qu'on aie pu se poser la question de la façon dont cette valeur approchée a été trouvée. On en a déduit que vraisemblablement l'algorithme de Heron d'Alexandrie (encore l'Égypte!) était connu des babyloniens. Dans la mesure où cet article est consacré à Sophus, voici ledit algorithme, utilisé ici pour calculer une valeur approchée de $\sqrt{3}$ [7] :

```
a = nouvelle Variable 1
b = nouvelle Variable 3
4 fois faire
    mettre Dans b, a
    inverser b
    multiplier b par 3
    augmenter a de b
    diviser a par 2
montrer a
```

V/ Sommes de séries

Le modèle de boîtes peut aussi servir à calculer les nombres triangulaires par exemple : On additionne les entiers successifs pour voir combien on obtient. Par exemple pour calculer 1+2+3+4+5+6+7+8+9+10, on prend une boîte initialement vide, puis on met 1 bille, puis 2 etc dans cette boîte. Ou on prend une boîte S (comme « somme ») initialement vide et on y ajoute répétitivement le contenu d'une boîte I (comme « index ») jusqu'à ce que celui-ci soit vide :

```
I = nouvelle Variable 9
S = nouvelle Variable 0
Jusqu'à ce que I.valeur == 0
    augmenter S de I
    décrémenter I
montrer S
```

7 Les nombres traités ici n'étant plus entiers, on peut songer au remplacement des boîtes de billes, par des seaux d'eau.

Comme on voit qu'il y a 9 termes on peut aussi faire plus simple :

```
I = nouvelle Variable 1
S = nouvelle Variable 0
9 fois faire
    augmenter S de I
    incrémenter I
montrer S
```

Cette technique est à la base du calcul d'intégrales par la méthode des rectangles. Par exemple si on veut calculer l'intégrale de la fonction « carré » entre 0 et 1 avec 1000 rectangles :

```
intégrale = nouvelle Variable 0
dx = nouvelle Variable 1 - 0
diviser dx par 1000
x = nouvelle Variable 0
1000 fois faire
    augmenter intégrale de carré x.valeur
    augmenter x de dx.valeur
multiplier intégrale par dx
montrer intégrale
```

La valeur affichée 0,333 est assez proche de l'intégrale elle-même...

Mais là on sort un peu du modèle des boîtes puisqu'on travaille sur des réels et plus sur des entiers.