MOONSCRIPT version 0.6

Au commencement il y a eu la rencontre avec Alain Prouté qui m'a apporté beaucoup d' idées rafraîchissantes :

- être Turing complet n'est pas du tout indispensable si on peut faire les mêmes choses sans l'être
- la théorie des catégories est un cadre à privilégier pour repenser les fondations des assistants de preuves et des langages fonctionnels
- puisque réutiliser est en général une bonne idée, pourquoi ne pas réutiliser la récursion, pourquoi ne pas réutiliser les preuves ?

C'est par ce biais que j'ai ensuite découvert le langage Charity, un langage expérimental qui incarne ces idées tout en y ajoutant la co-récursion. Malheureusement le langage Charity est assez limité, il privilégie la composition sur l'application, il n'est plus maintenu, il ne possède que la récursion et la co-récursion primitives, il n'applique aucune loi de fusion, il ne possède ni les hylomorphismes ni les zygomorphismes.

Qu'est-ce que Moonscript

Moonscript n'est rien de plus qu'une syntaxe jetable pour tester toutes mes idées afin de sonder leurs limites. L'objectif étant de capitaliser un maximum de retour d'expérience sans investir le moindre effort d'implantation.
Le plan initial est simple :

- Commencer par implanter les structures de données les plus communes à l'aide des concepts les plus simples, à savoir le branchement par cas, les foncteurs, les catamorphismes et les paramorphismes.
- Les TADs conteneurs concernés par cette première partie sont:
  1. les résultats / erreurs,
  2. les options,
  3. les listes,
  4. les listes à accès aléatoire rapide,
  5. les double-queues,
  6. les ensembles ordonnées (non équilibrés)
     à partir desquels on pourrait facilement dériver un dictionnaire,
  7. les ensembles d'entiers naturels,
     avec opérations ensemblistes rapides
  8. les tableaux indexés par un entier naturel, avec opérations ensemblistes rapides
  9. les inventaires avec opérations ensemblistes rapides, (4 boulons & 3 vis + 2 boulons & 4 vis = 6 boulons & 7 vis)
  10. les dictionnaires bijectifs (que l'on peut interroger par l'élément ou par la clé)
  11. les files de priorité,
  12. les files de priorité capables de restituer l'ordre d'insertion,
  13. les digraphes inductifs sans étiquettes (ni sur les objets ni sur les flèches)

- En Moonscript la copie partielle et la modification sur place ont la même syntaxe. De cette façon chaque TAD implante à la fois une version mutable et une version immutable du même .
- Un prélude ajoute quelques fonctions usuelles. Le tout est accompagné d' exemples convaincants par leur concision et leur économie de moyens. Un dérivateur formel, une calculatrice ou un interpréteur de λ-calcul deviennent moins impressionnants dès lors qu'on n'a plus à implanter toute la mécanique de récursion.
- Moonscript 0.6 inclut tout ces composants plus quelques sources équivalentes en Objective-Caml 3.11 qui servent principalement à garder les pieds sur terre.

Les sources Moonscript se lisent idéalement avec *Crimson-Editor* et une tabulation réglée à 3 caractères. Une coloration lexicale expérimentale pour *GtkSourceView2* est également fournie mais n'a jamais été testée.

Bilan de la version 0.6

L'objectif depuis la version 0.1 était d'évaluer l'expressivité des récurseurs, au final 4 ont retenu mon attention, plus 3 autres sérieux candidats :

fold est l'itérateur à la façon de Lisp & Caml (catamorphisme, évaluation stricte)
cata est l'itérateur à la façon de Haskell (catamorphisme, évaluation retardée)
recu est le récurseur à la façon de Saunders (paramorphisme, évaluation stricte)
para est le récurseur primitif à évaluation retardée (paramorphisme, façon Charity)
fold2 et **cata2** sont les variantes qui progressent simultanément sur 2 arguments
bicata est une variante qui progresse alternativement sur 2 arguments

Ce choix de 4 récurseurs est un choix pragmatique là où en théorie un seul récurseur pourrait être suffisant. Cependant la diversité des exemples plaide pour un panel plus large qui a le mérite de maximiser la réutilisation.

Et ensuite

D'abord on peut implanter d'autres comme les listes à concaténation rapide, les dictionnaires associatifs et les dictionnaires bijectifs (que l'on peut interroger par l'élément ou par la clé).
Ensuite il faut davantage de moyens pour aborder des exemples plus sophistiqués :
Afin d'atteindre la même efficacité asymptotique que les implantations de TADs classiques il est indispensable de développer de nouveaux récurseurs qui supportent les rotations dans les arbres binaires.
Il y a certainement des généralisations à trouver du côté des bi-récurseurs, notamment pour mieux traiter la fusion, très fréquente dans les files de priorités avancées.
La résolution de jeux (problème du cavalier, Rubik's cube, le compte est bon) montre combien le point de rencontre entre la co-récursion (qui gère la partie génération) et la récursion (qui gère la partie élimination) est au coeur de nombreux problèmes d'exploration.
Grâce aux graphes inductifs de Martin Erwing il devrait être possible d'implanter et de raisonner de façon constructive sur tous les graphes et les multi-graphes.

L'exemple des graphes montre la nécessité de pouvoir dériver un nouveau type de données à partir de pseudo-constructeurs/destructeurs actifs.
Je n'ai pas complètement les moyens d'anticiper quelles initiatives sont fatales au passage en logique constructive.
Par contre je suis prêt à fermer toutes les voies qui pourraient mener à une impasse logique.

Glossaire

- Récurseur
- Type de données conteneur
- Nouveauté
- Idée en gestation
- Idée retenue
- Idée retenue et mise en oeuvre
- Idée abandonnée, obsolète, ou non retenue
- Impossibilité
- Bonne nouvelle
- Mauvaise nouvelle
- Bogue connu
- Nouveau bogue découvert
- Bogue en cours de traitement
- Bogue résolu
- Objective-Caml 3.11
- Moonscript 0.6
- Linux
- Microsoft Windows
- Lien **html**
- Documentation **pdf**
- Personne ayant inspiré Moonscript

Bibliographie

Alain Prouté: *Cours de Logique Catégorique*
(http://www.math.jussieu.fr/~alp/)

Tom Fukushima et **Charles Tuckey:** *Charity User Manual*
(http://pll.cpsc.ucalgary.ca/charity1/www/home.html)

Chris Okasaki: *Purely Functional Data Structures*
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.6229)

Edward Kmett: *Recursion Schemes: A Field Guide*
(http://comonadcom/reader/2009/recursion-schemes/)

Martin Erwing: *Inductive Graphs and Functional Graph Algorithms*
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.9377)

David Turner: *Total Functional Programming*
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.5271)