Rapport du projet orienté objet de programmation fonctionnelle et symbolique

Yasin SENEL & Jérémy TURON & Salomé COAVOUX

14 décembre 2012

Table des matières

1 Les énumérateurs à liste **3**

2 Les énumérateurs inductifs **3**

3 Les énumérateurs parallèles **3**

4 Les énumérateurs à filtre **3**

5 Les énumérateurs par concaténation **4**

6 Les énumérateurs à mémoire **4**

7 Les énumérateurs à produit **4**

8 Conclusion **5**

1 Les énumérateurs à liste

Le code des énumérateurs à liste est très simple. La structure comporte trois champs. Le champ *list-elem* contient la liste donnée à la construction de l'énumérateur. Le champ *next-elem* contient la liste des éléments non parcourus, à laquelle on enlève le *car* à chaque appel de *call-enum*. A la création, ou à l'initialisation, ces deux champs font référence à la même liste. Ainsi, *call-enum* est destructif sur *next-elem*, mais pas sur *list-elem*, ce qui permet, de réinitialiser l'énumérateur en utilisant ce dernier. Le dernier champ de la structure est un booléen *is-circular* pour savoir si la liste est circulaire ou pas. Ainsi, en appelant *call-enum*, on avance dans la liste et à la fin, on retourne au début si l'énumérateur est circulaire, sinon la liste est finie et tous les prochains appels renverront *(nil nil)*.

2 Les énumérateurs inductifs

Les énumérateurs inductifs ont, eux, besoin d'un *first-element* qui constituera le premier élément, sur lequel se basera le reste de l'énumérateur. De plus, ils utilisent *function-enum*, une fonction permettant de construire l'élément suivant à partir de l'élément courant, et qui sera donc appliquée à *next-elem* à chaque appel de *call-enum*. Le champ *next-elem*, quant à lui, contient uniquement le champ suivant la position courante, contrairement aux énumérateurs par liste. Il sera donc initialisé au deuxième élément de l'énumérateur, c'est-à-dire à la valeur de *first-elem* à laquelle on aura appliqué *function-enum*. Nous avons décidé de ne pas considérer les espaces de définition des fonctions, rendant ainsi les énumérateurs inductifs infinis.

3 Les énumérateurs parallèles

Les énumérateurs parallèles utilisent une liste d'énumérateurs *list-element* pour fonctionner. Elle se contente, dans chaque méthode, d'appeler la méthode de la classe propre à chacun des énumérateurs de sa liste, grâce, notamment, à la fonction *mapcar*.

4 Les énumérateurs à filtre

Les énumérateurs à filtre ont pour but d'appliquer à un énumérateur *f-enum* le prédicat *pred*, ce qui aura pour effet de renvoyer à chaque appel de *call-enum* le prochain élément de *f-enum* pour lequel le prédicat est correct. Ainsi, le champ *next-elem* contient le prochain élément de l'énumérateur passé en paramètre. C'est *call-enum* qui s'occupera de filtrer ce dernier, c'est-à-dire d'afficher *next-elem* si le prédicat lui est applicable, ou de chercher dans les éléments postérieurs le prochain qui sera valable pour le filtre en question. Les problèmes majeurs de cette implémentation sont :

– La méthode *next-element* renvoie le prochain élément à tester, et pas le prochain élément du filtre.
– La méthode *next-element-p* indique s'il y a d'autres éléments à tester.
– Dans le cas d'un énumérateur infini, il est possible que le prédicat ne soit plus jamais vrai à partir d'une certaine valeur. Dans ce cas-là, un appel à *call-enum* entraînera une boucle infinie.

5 Les énumérateurs par concaténation

Ces énumérateurs fonctionnent sensiblement comme les énumérateurs à liste, à la seule différence qu'au lieu d'appeler le prochain élément de sa liste, il renvoie le prochain élément de l'énumérateur qu'il est en train de traiter, ou le premier élément de l'énumérateur suivant dans le cas où on est à la fin de l'énumérateur courant. La structure a été reprise des énumérateurs à liste, à laquelle on a supprimé le champ *is-ciruclar*. Ici, *list-element* stocke la liste des énumérateurs à concaténer, et *next-element* celle des énumérateurs qu'il reste à traiter. De plus, il s'agit d'une copie par référence, et pas par valeur.

6 Les énumérateurs à mémoire

Ces énumérateurs contiennent :
– *m-enum*, l'énumérateur sur lequel va être appliquée la mémorisation.
– *next-elem*, qui va stocker le prochain élément juste avant qu'il soit renvoyé, de manière à pouvoir le renvoyer en cas de mémorisation.
– *rep*, booléen indiquant si la mémorisation est activée ou non.
– *end*, qui indique s'il reste un prochain élément à l'énumérateur sans prendre en compte la répétition.
– *out*, un troisième booléen qui sert à indiquer si l'on est encore dans l'énumérateur et que l'on répète l'élément, ou si l'on est sorti de celui-ci et que l'on répète *NIL NIL*.

7 Les énumérateurs à produit

Nous avons décidé d'implémenter les énumérateurs à produit généraux plutôt que les énumérateurs à produit cartésien, car ils sont fortement identiques ; il suffirait de supprimer le champ *fun* et de remplacer *(fun e)* dans la méthode *next-element* par un simple #'*. Les champs des énumérateurs à produit sont :
– *list-element*, la liste des énumérateurs à traiter, sous forme d'énumérateur à mémoire.
– *fun*, la fonction appliquée à ces énumérateurs.
– *has-next*, un booléen qui permet de savoir s'il existe un prochain élément. Ce champ a été rajouté car nous n'arrivions pas à déterminer si nous étions à la fin de l'énumérateur à produit en utilisant simplement *next-element-p* sur l'ensemble des énumérateurs contenus dans *list-element*, ceci à cause

de la mémorisation. En effet, les énumérateurs à traiter ont toujours la mémorisation activée sauf au moment où on passe à leur élément suivant. Ainsi, quand tous les énumérateurs étaient à leur dernier élément, *next-element-p* renvoyait vrai à cause de la mémorisation.

Dans un énumérateur à produit, le n-ième énumérateur de *list-element* ne passe à son élément suivant que si le n+1-ième énumérateur vient de finir de parcourir tous ses éléments. Ainsi, notre implémentation de *next-element* évalue la fonction sur l'ensemble des éléments mémorisés sans modifier les énumérateurs, puis fait appel à la fonction *suivant* qui va prendre en paramètre la liste renversée des énumérateurs, car il faut d'abord traiter le n+1-ième énumérateur pour savoir quoi faire du n-ième. Cette fonction va tout d'abord faire passer à l'élément suivant le premier énumérateur de cette liste, et, si celui-ci est terminé, le réinitialise et traite récursivement le *cdr* de la liste. S'il s'agit du dernier énumérateur, et qu'il est terminé, cela signifie que tous les autres énumérateurs étaient également terminés, on a donc fini de parcourir tous les éléments de tous les énumérateurs ; on signale cette fin en faisant passer *has-next* à *NIL*.

8 Résultats

Dans l'ensemble, toutes nos classes fonctionnent quelle que soit l'imbrication entre les énumérateurs. Le seul problème a été, comme dit précédemment, pour les énumérateurs à filtre. Ce problème nous a obligé à utiliser *call-enum* au lieu de *next-element* dans toutes les autres classes dont l'un des champs était un énumérateur dont nous voulions obtenir l'élément suivant.