Université Montpellier II — Master d'Informatique
UEMINM 121 Evaluation des Langages Applicatifs

R. Ducournau – M. Lafourcade

janvier 2005 — 2 heures
Documents autorisés : notes de cours, polys, pas de livre.

Méta-évaluateur paresseux

Les évaluateurs habituels — et le méta-évaluateur vu en cours en fait partie — sont déterministes et « gloutons » (*eager* en anglais) : ils évaluent au fur et à mesure, dans un ordre immuable, toute expression évaluable sans se préoccuper de savoir si le résultat sera utilisé ou pas. L'objectif de cette question est de spécifier un méta-évaluateur « paresseux » (*lazy*) qui n'évalue une expression que lorsqu'il apparait que son résultat est nécessaire pour une autre opération.

Ces deux modes d'évaluation se distinguent par la façon dont les arguments sont évalués lors de l'appel des vraies fonctions :

- un évaluateur glouton évalue tous les arguments d'une fonction avant de l'appliquer ;
- un évaluateur paresseux n'évalue pas les arguments d'une fonction interprétée lors de l'appel : il attend d'avoir besoin de la valeur pour le faire.

Dans les deux cas, les arguments des fonctions prédéfinies sont évalués de façon gloutonne : c'est à cette occasion que l'évaluation des arguments non évalués sera forcée.

Exemple Soit la fonction

```
(defun foo (w x y z)
       (if (< w x) y z))
```

L'évaluation de l'expression `(foo 1 2 (fact n) (fibo n))` ne nécessite pas l'évaluation de `(fibo n)` : la complexité sera donc en $O(n)$ avec un évaluateur paresseux et en $O(2^n)$ avec un évaluateur glouton (en supposant `fibo` écrite de façon naïvement récursive).

Programmation fonctionnelle pure

On suppose d'abord un style de programmation purement fonctionnel, sans aucun effet de bord.

Question 1

Montrer par des arguments informels, que le comportement des deux types d'évaluateurs — c'est-à-dire la valeur retournée par l'évaluation — sera en général le même.

Quelles exceptions peut-il y avoir à cette identité des comportements ? Montrer que si l'évaluateur glouton retourne une valeur, l'évaluateur paresseux retournera la même, et donnez un exemple montrant que la réciproque n'est pas forcément vraie.

Si vous avez des notions de λ-calcul, servez-vous en comme argument.

Environnements paresseux

On reste dans un contexte de programmation fonctionnelle pure, sans effets de bord.

La différence entre les deux modes d'évaluation porte sur l'appel de fonction interprétée (en généralisant aux λ-exressions et au `let`) dont les arguments sont évalués ou pas. Elle se ramène donc d'abord à une différence dans la structure des environnements. Dans un évaluateur glouton, l'environnement lexical est constitué d'une liste d'associations variable-valeur, par exemple `((w . 1)(x . 2)(y . 24)(z . 5))` pour l'exemple précédent, si l'on suppose que l'appel de `foo` s'est fait dans l'environnement `((n . 4))`.

Avec l'évaluateur paresseux, l'environnement lexical doit être une structure plus compliquée où :

– chaque variable est liée, au début, à l'expression argument non évaluée, et à l'environnement dans lequel il faudrait l'évaluer ;
– dès que l'évaluation de la variable a été forcée, la variable doit être liée à la valeur résultant de cette évaluation.

Question 2

Spécification des environnements, de leur création et de leur manipulation.

1. Proposer une structure de données qui permette de représenter cet environnement à géométrie variable ; ne pas oublier qu'il faut être capable de faire la différence entre l'expression évaluée et non évaluée ;
2. Spécifier et définir la fonction LISP `make-lazy-env` qui construit un environnement paresseux à partir des arguments d'un appel et des paramètres de la fonction (ne pas traiter `&optional`, etc.) ;

Question 3

Dans la fonction principale de méta-évaluation paresseuse, appelée `lazy-meval` :

1. Spécifier et définir l'évaluation des variables dans un environnement paresseux tel que défini plus haut ;
2. Traiter le cas des appels de fonction paresseux (fonction méta-définies et formes assimilées telles que λ-expressions et `let`).

Affectation de variable

On abandonne maintenant l'hypothèse d'une style de programmation purement fonctionnel, et on s'intéresse à l'influence des effets de bord, en premier lieu de l'affectation de variable `(setf <var> <val>)`.

Deux séries de questions se posent :

1. l'affectation doit-elle être paresseuse, c'est-à-dire faut-il évaluer la valeur (`<val>`) à affecter ? Quand peut-on s'en passer ?
2. plus généralement, l'affectation a-t-elle un sens dans un évaluateur paresseux ?

Question 4

1. Argumenter un traitement différent suivant que l'on considère l'effet de bord (l'affectation elle-même), ou la valeur à retourner (dans le cas où l'évaluation en cours doit retourner la valeur retournée par le `setf`). Comment le méta-évaluateur peut-il arriver à distinguer ces deux cas ?
2. Considérer les cas suivants :
   – d'une séquence d'expressions (`progn`) incluant l'expression `setf` ;
   – où l'expression `setf` est l'argument d'un appel de fonction paresseux ;
   – où l'expression `<val>` est fonction de `<var>` (par exemple incrémentation de la variable) ;
   – où la variable `x` à affecter figure dans l'expression non encore calculée à laquelle est liée une variable `y` : quel effet aurait l'affectation de `x` ?

   Au vu des ces différents exemples, l'affectation de variable a-t-elle encore un sens dans un évaluateur paresseux ?

Evaluation paresseuse des effets de bord

Question 5

Pour conclure sur les effets de bord et sur l'évaluation paresseuse :

1. montrer que les effets de bord rendraient indéterministe le comportement d'un évaluateur paresseux.
2. On suppose que toute fonction prédéfinie (non connue par le méta-évaluateur) est étiquetée de façon booléenne, comme « fonctionnelle » ou « à effet de bord », suivant que son comportement est purement fonctionnel ou pas.

   Proposer alors un schéma permettant de faire cohabiter évaluateurs glouton et paresseux de telle façon que le comportement soit paresseux tant qu'il n'y a pas d'effet de bord. Au moins deux approches sont possibles, soit par analyse statique des fonctions, soit par changement dynamique du mode d'évaluation : spécifier une de ces solutions mixtes en montrant comment les deux modes interagissent.