Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
GLO-66811

Danny Dubé
Hiver 2008

Examen
24 avril 2008

Nom de l'étudiant(e) (facultatif): _____

Matricule: _____

Programme d'études: _____

Directives pédagogiques:

- Documentation papier permise
- Ordinateurs et calculatrices interdits
- Répondre sur le questionnaire
- Le verso des feuilles du questionnaire ne sera pas lu à moins que vous n'y référiez explicitement

1. **λ-calcul**. 15 points. Vous devez étendre la sémantique du λ-calcul paresseux afin d'inclure un opérateur appelé 'strict' au langage. On sait que, dans le λ-calcul paresseux, les arguments ne sont pas évalués avant d'être passés. L'opérateur a l'effet de transformer une fonction ordinaire en une fonction stricte. L'expression (strict e) est d'abord évaluée en évaluant e et, ensuite, en rendant la fonction produite stricte. On veut faire en sorte que, dans l'appel (v e), si v est une fonction ordinaire, alors la β-réduction peut être faite immédiatement mais, si v est une fonction stricte, il faut commencer par évaluer e et ensuite faire la β-réduction. Complétez les définitions suivantes afin d'obtenir ce comportement (on omet les α-réductions).

$$
\begin{array}{rcl}
e & ::= & \mathrm{x} \mid \lambda \mathrm{x}.\ e \mid e\ e \mid \text{strict } e \\
v & ::= & \mathrm{x} \mid \lambda \mathrm{x}.\ e \mid \text{strict } v \\
C'' & ::= & ? \\
 & \overset{\beta}{\mapsto} & \text{est défini comment?}
\end{array}
$$

2. **Implantation des fonctions**. 5 points. Supposons qu'une implantation soit modifiée afin qu'elle devienne *safe for space*. Il se trouve que certains programmes peuvent voir leur complexité en temps s'améliorer (c'est-à-dire, s'améliorer par plus qu'un facteur constant). Par exemple, un programme ayant un temps d'exécution quadratique sur l'implantation originale pourrait avoir un temps d'exécution linéaire sur la nouvelle implantation. Expliquez comment c'est possible.

3. **Optimisations et analyse statique**. 15 points. Ajoutez des *cellules* au langage vu dans la dernière section de matière du cours. Vous devez donner la sémantique étendue en étendant la syntaxe des expressions, celle des valeurs et possiblement celle des contextes en plus de rajouter une ou plusieurs règles de réduction. Les cellules sont pareilles aux paires sauf qu'elles n'ont qu'un seul champ. L'expression 'cell e' crée une nouvelle cellule dont le contenu est fourni par e. L'expression 'cell-ref e' extrait le contenu d'une cellule (en autant que e produise bien une cellule). L'expression 'cell? e' (qui est l'analogue de 'pair? e') teste si e a produit une cellule; dans l'affirmative, elle retourne ladite cellule et, dans la négative, $\#f$. Voici le langage tel que vu en classe:

$$
\begin{array}{rcl}
e & ::= & \#f \\
 & | & \mathrm{x} \\
 & | & (\lambda \mathrm{x}.\ e) \\
 & | & (e\ e) \\
 & | & (\mathrm{if}\ e\ e\ e) \\
 & | & (\mu \mathrm{x}.\ e) \\
 & | & (\mathrm{cons}\ e\ e) \\
 & | & (\mathrm{car}\ e) \\
 & | & (\mathrm{cdr}\ e) \\
 & | & (\mathrm{pair?}\ e)
\end{array}
$$

$$
\begin{array}{rcl}
C^\beta & ::= & (C^\beta\ e) \\
 & | & (v\ C^\beta) \\
 & | & (\mathrm{if}\ C^\beta\ e\ e) \\
 & | & (\mathrm{cons}\ C^\beta\ e) \\
 & | & (\mathrm{cons}\ v\ C^\beta) \\
 & | & (\mathrm{car}\ C^\beta) \\
 & | & (\mathrm{cdr}\ C^\beta) \\
 & | & (\mathrm{pair?}\ C^\beta) \\
 & | & [\![\cdot]\!]
\end{array}
$$

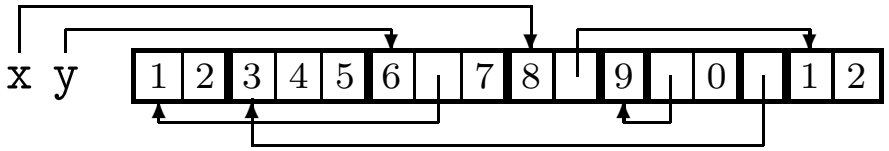
$$
\begin{array}{rcl}
v & ::= & \#f \\
 & | & (\lambda \mathrm{x}.\ e) \\
 & | & (\mathrm{cons}\ v\ v)
\end{array}
$$

$$
\begin{array}{rcl}
C^\alpha & ::= & (\lambda \mathrm{x}.\ C^\alpha) \\
 & | & (C^\alpha\ e) \\
 & | & (e\ C^\alpha) \\
 & | & (\mathrm{if}\ C^\alpha\ e\ e) \\
 & | & (\mathrm{if}\ e\ C^\alpha\ e) \\
 & | & (\mathrm{if}\ e\ e\ C^\alpha) \\
 & | & (\mu \mathrm{x}.\ C^\alpha) \\
 & | & (\mathrm{cons}\ C^\alpha\ e) \\
 & | & (\mathrm{cons}\ e\ C^\alpha) \\
 & | & (\mathrm{car}\ C^\alpha) \\
 & | & (\mathrm{cdr}\ C^\alpha) \\
 & | & (\mathrm{pair?}\ C^\alpha) \\
 & | & [\![\cdot]\!]
\end{array}
$$

$$
\begin{array}{rcl}
C^\alpha[\![\lambda \mathrm{x}.\ e]\!] & \overset{\alpha}{\mapsto} & C^\alpha[\![\lambda \mathrm{y}.\ (e[\mathrm{x} \mapsto \mathrm{y}])]\!] \\
C^\alpha[\![\mu \mathrm{x}.\ e]\!] & \overset{\alpha}{\mapsto} & C^\alpha[\![\mu \mathrm{y}.\ (e[\mathrm{x} \mapsto \mathrm{y}])]\!]
\end{array}
$$

$$
\begin{array}{rcl}
C^\beta[\![(\lambda \mathrm{x}.\ e)\ v]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e[\mathrm{x} \mapsto v]]\!] \\
C^\beta[\![\mathrm{if}\ \#f\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_3]\!] \\
C^\beta[\![\mathrm{if}\ (\lambda \mathrm{x}.\ e_1)\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_2]\!] \\
C^\beta[\![\mathrm{if}\ (\mathrm{cons}\ v_1\ v_2)\ e_2\ e_3]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e_2]\!] \\
C^\beta[\![\mu \mathrm{x}.\ e]\!] & \overset{\beta}{\mapsto} & C^\beta[\![e[\mathrm{x} \mapsto (\mu \mathrm{x}.\ e)]]\!] \\
C^\beta[\![\mathrm{car}\ (\mathrm{cons}\ v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![v_1]\!] \\
C^\beta[\![\mathrm{cdr}\ (\mathrm{cons}\ v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![v_2]\!] \\
C^\beta[\![\mathrm{pair?}\ \#f]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\#f]\!] \\
C^\beta[\![\mathrm{pair?}\ (\lambda \mathrm{x}.\ e)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\#f]\!] \\
C^\beta[\![\mathrm{pair?}\ (\mathrm{cons}\ v_1\ v_2)]\!] & \overset{\beta}{\mapsto} & C^\beta[\![\mathrm{cons}\ v_1\ v_2]\!]
\end{array}
$$

4. **Récupération automatique de la mémoire**. 10 points. Simulez le GC Mark-and-Compact sur le tas suivant. Dessinez le tas après le marquage et à nouveau après le compactage.



5. **Représentation des objets et des types**. 10 points. Un implanteur du langage Bizz a fixé les bits de "taggage" pour les 8 types d'objets du langage. Il l'a fait en pensant qu'il disposerait des **trois** bits les moins significatifs pour le "taggage". La convention qu'il a fixé est présentée plus bas. Or, il a commis une erreur car, en fait, il n'y a que les **deux** bits les moins significatifs à sa disposition. Refaites son travail en justifiant vos choix. (La colonne de droite indique les besoins minimaux en bits pour les objets à représentation directe.)

`xx...xx000`	$\longrightarrow$	*fixnums*	(repr. directe)	(sur 29 bits exact.)
`xx...xx001`	$\longrightarrow$	symboles	(repr. directe)	(sur 20 bits min.)
`xx...xx010`	$\longrightarrow$	liste vide	(repr. directe)	
`xx...xx011`	$\longrightarrow$	booléens	(repr. directe)	(sur 1 bit exact.)
`xx...xx100`	$\longrightarrow$	fermetures	(repr. indirecte)	
`xx...xx101`	$\longrightarrow$	paires	(repr. indirecte)	
`xx...xx110`	$\longrightarrow$	caractères	(repr. directe)	(sur 16 bits exact.)
`xx...xx111`	$\longrightarrow$	vecteurs	(repr. indirecte)	

6. **Implantation de la récursion**. 5 points. Transformez le programme suivant en utilisant la méthode “trampoline”.

```
(define crunch
  (lambda (f n lst)
    (map (lambda (g)
           (f (g n)))
         lst)))
```

7. **Implantation de la récursion**. 10 points. Transformez le programme suivant en forme CPS.

$$(\lambda\ \mathrm{x\ y\ z.\ z} + \#2\ \mathrm{x})\ (1 : 2)\ 3\ 4$$

8. **Implantation des fonctions**. 15 points. Effectuez la défonctionnalisation du programme suivant, une fois afin d'obtenir la représentation chaînée des fermetures et une fois afin d'obtenir la représentation plate.

```
(define S
  (lambda (f)
    (lambda (g)
      (lambda (x)
        ((f x) (g x))))))
```

9. **λ-calcul**. 10 points. En classe, nous avons vu que les définitions suivantes décrivaient la sémantique du λ-calcul avec la stratégie d'évaluation en ordre applicatif. Refaites ces définitions afin d'avoir encore une stratégie d'évaluation en ordre applicatif *mais* avec l'évaluation de droite à gauche (i.e. l'argument est évalué avant l'opérateur). Inscrivez uniquement ce que vous changez.

$$
\begin{array}{rcl}
e & ::= & \mathrm{x} \mid \lambda \mathrm{x}.\ e \mid e\ e \\
v & ::= & \mathrm{x} \mid \lambda \mathrm{x}.\ e \\
C & ::= & [\![\cdot]\!] \mid \lambda \mathrm{x}.\ C \mid C\ e \mid e\ C \\
C'' & ::= & [\![\cdot]\!] \mid C''\ e \mid v\ C'' \\
C[\![\lambda \mathrm{x}.\ e]\!] & \overset{\alpha}{\mapsto} & C[\![\lambda \mathrm{y}.\ (e[\mathrm{x} \mapsto \mathrm{y}])]\!] \\
C''[\![(\lambda \mathrm{x}.\ e)\ v]\!] & \overset{\beta}{\mapsto} & C''[\![e[\mathrm{x} \mapsto v]]\!]
\end{array}
$$

10. **Vrai ou faux**. 5 points.

1. Soit un gestionnaire de mémoire utilisant un GC incrémentiel non-générationnel. À cause de l'exécution en alternance du mutateur et du GC, un objet mort pourrait passer à travers un nombre arbitraire de cycles de GC sans être ramassé.

   _____

2. La représentation par pointeurs taggés ne peut servir que dans l'implantation des langages typés dynamiquement.

   _____

3. La transformation en forme CPS peut ne pas terminer lorsqu'elle s'effectue sur certains termes.

   _____

4. La complexité en temps d'un programme transformé en utilisant la méthode "trampoline" ne peut augmenter comparativement à celle du programme original (c'est-à-dire que toute augmentation ne pourrait être que par un facteur constant).

   _____

5. La complexité en temps d'un programme transformé en forme CPS de la manière vue en classe ne peut augmenter comparativement à celle du programme original (c'est-à-dire que toute augmentation ne pourrait être que par un facteur constant).

   _____