

Vrije Universiteit Brussel – Faculteit Wetenschappen – Vakgroep
Computerwetenschappen
Academiejaar 2009 – 2010: eerste examenzitting
Interpretatie van Computerprogramma's I
— schriftelijke test —

Voorafgaandelijk: de vragen zijn geformuleerd in functie van van de gepubliceerde nota's; deze mogen, samen met persoonlijke nota's en het referentieboek van Abelson & Sussman, geraadpleegd worden tijdens de test. Het gebruik van (electro-)mechanische hulpmiddelen is niet toegestaan. Gelieve elk antwoord te beantwoorden op een apart blad en te beperken tot maximum één blad. Geef voor alle vragen één blad af en zet bovenaan elk blad je naam en het nummer van de vraag die beantwoord wordt.

*Succes!
Theo D'Hondt
13 juni 2010*

Vraag #1: beschouw deel 1a (De meta-circulaire evaluator).

Bij het maken van ADT's in Scheme komt vaak hetzelfde patroon terug. Traditioneel maak je altijd een constructor, getter, setter, copy, etc. Bijvoorbeeld voor een simpel ADT voor een punt met een x en y coördinaat wordt dit:

```
(define (make-point x y)
  (lambda (msg . args)
    (cond ((eq? msg 'get-x) x)
          ((eq? msg 'get-y) y)
          ((eq? msg 'set-x) (set! x (car args)))
          ((eq? msg 'set-y) (set! y (car args)))
          ((eq? msg 'copy-point) (make-point x y))))
(define (get-point-x point) (point 'get-x))
(define (get-point-y point) (point 'get-y))
(define (set-point-x point value) (point 'set-x value))
(define (set-point-y point value) (point 'set-y value))
(define (copy-point point) (point 'copy-point))
```

Voeg in de meta-circulaire evaluator een nieuwe functionaliteit toe die je toelaat dit allemaal automatisch te laten definiëren met alleen de volgende code:

```
(defstruct point x y)
```

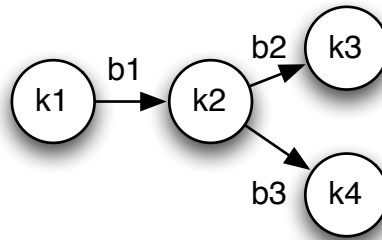
Voeg deze defstruct als een afgeleide expressie aan de meta-circulaire evaluator toe:

```
(defstruct name var1 ... varn)
```

zodanig dat automatisch functies `get-name-vari`, `set-name-vari!`, `copy-name` worden toegevoegd.

Vraag #2: beschouw deel 1d (Logisch programmeren).

Gegeven is de gerichte, acyclische graf g1:



a/ Stel deze graf g1 voor aan de hand van logische feiten van de vorm:

(graf-heeft-knoop ?graf ?knoop)

en:

(graf-heeft-boog ?graf ?boog ?beginKnoop ?eindKnoop).

Stel vervolgens een logische vraagstelling (“query”) op die knopen identificeert waaruit meerdere bogen vertrekken.

b/ Volgend logisch feit kan eveneens gebruikt worden om graf g1 voorstellen:

(graf g1 (a b c d) ((b1 a b) (b2 b c) (b3 b d)))

Dit feit is van de vorm (graf ?graf ?knopen ?bogen) waarbij ?knopen en ?bogen een lijst van respectievelijk alle knopen en alle bogen van de graf zijn. In deze voorstelling wordt elke boog op haar beurt voorgesteld als een lijst van 3 elementen (?boog ?beginKnoop ?eindKnoop).

Herimplementeer, tegenover de nieuwe grafvoorstelling, predicaat (graf-heeft-knoop ?graf ?knoop) en predicaat (graf-heeft-boog ?graf ?beginKnoop ?eindKnoop) aan de hand van logische regels. Je mag hiervoor gebruik maken van het predicaat (element ?el ?lijst) uit de oefeningenlessen dat slaagt wanneer ?el een element is uit de lijst ?lijst.

c/ Het predicaat (pad ?graf ?begin ?einde ?pad) slaagt wanneer er in een **acyclische** ?graf een pad ?pad tussen knopen ?begin en ?einde bestaat. Hierbij is ?pad de lijst van opeenvolgende bogen tussen ?begin en ?einde. Volgende query slaagt bijvoorbeeld:

(pad g1 a d (b1 b3))

Volgens “L. Redeneerder” zouden volgende regels dit predicaat moeten implementeren:

```
(rule (pad ?graf ?knoop ?knoop ())
      (graf-heeft-knoop ?graf ?knoop))
(rule (pad ?graf ?begin ?einde (?boog . ?pad))
      (and (pad ?graf ?knoop ?einde ?pad)
            (graf-heeft-boog ?graf ?boog ?begin ?knoop)))
```

Hierbij stelt de tweede regel dat er een pad van ?begin naar ?einde bestaat als er een pad van ?knoop naar ?einde bestaat en een boog van ?begin naar ?knoop. Alhoewel dit vanuit een logisch perspectief correct is, blijkt dit vanuit het procedurele perspectief van de query evaluator toch tot problemen te leiden. We merken namelijk dat volgende vraagstelling de query evaluator in een oneindige lus brengt (de evaluator "loopt"):

```
;;; Query input:  
(pad g1 ?x ?y ?p)
```

```
;;; Query results:  
<100% CPU gebruik>
```

Nochtans bevat graf g1 geen lussen.

Verklaar en motiveer aan de hand van fragmenten uit de implementatie van de query evaluator. Pas vervolgens de implementatie van de **tweede regel** aan zodat deze niet langer aanleiding geeft tot lussen. Hint: je hoeft slechts een minimale aanpassing door te voeren!

d/ Ervan uitgaande dat je de tweede regel van het predicaat (pad ?graf ?begin ?einde ?pad) correct hebt aangepast, geeft de query evaluator volgende resultaten terug:

```
;;; Query input:  
(pad g1 ?b ?e ?p)
```

```
;;; Query results:  
(pad g1 k2 k3 (b2)) (pad g1 k3 k3 ()) (pad g1 k1 k3 (b1 b2)) (pad g1 k2 k2 ())  
(pad g1 b d (b3)) (pad g1 k1 k1 ()) (pad g1 k1 k2 (b1)) (pad g1 d d ())  
(pad g1 a d (b1 b3)) (pad g1 a a ()) (pad g1 b c (b2)) (pad g1 b b ())  
(pad g1 a b (b1)) (pad g1 c c ()) (pad g1 a c (b1 b2))
```

Tussen deze resultaten staan lege paden van de vorm (pad g1 ?k ?k ()), terwijl men kan argumenten dat geldige paden uit minstens een boog moeten bestaan. Pas de **eerste regel** van het predicaat aan zodat dit niet langer lege paden teruggeeft. Hint: de aanpassing is opnieuw minimaal!

Vraag #3: beschouw deel 2c (Registermachines).

Veronderstel dat je scheme omgeving geen *vermenigvuldig* operatie bevat en enkel beschikt over de primitieve functies -, + en =.

Ontwerp daarom een registermachine subroutine (= stuk registermachinecode dat (1) begint met een label waar naartoe kan gesprongen worden, (2) een resultaat in het val register plaatst en (3) terugspringt naar de inhoud van het continue register) die de vermenigvuldiging uitwerkt als een recursief proces op basis van de operaties -, + en =.

Ontwerp daarvoor eerst een recursieve Scheme functie die twee getallen met elkaar vermenigvuldigt door enkel gebruik te maken van de primitieve functies -, + en =.

Hint: probeer dit laatste op staartrecursieve wijze te doen - dat maakt de registermachine-code ineens veel eenvoudiger.

Vraag #4: beschouw deel 2d (Garbage collection).

In de versie van de garbage collector uit het boek wordt eens zoveel geheugen gebruikt als werkelijk nodig is voor de behoeften van het programma dat de cons-oproepen doet.

Zou het mogelijk zijn om de car- en de cdr-helften van het geheugen los van mekaar te behandelen? Dat wil zeggen eerst alle car-helften af te lopen die bereikbaar zijn vanuit de root, en vervolgens alle cdr-helften. In beide gevallen moet dan wel nog steeds gecopieerd worden naar een nieuw en extra geheugen, maar dat is dan maar half zo groot als in het oorspronkelijke geval.

Geen een beredeneerd antwoord.

Vraag #5: beschouw deel 2e (Vertaling).

Beschouw de volgende registermachinecode die het resultaat is van de vertaling van een Scheme expressie:

```
((env continue)
 (proc argl val)
 ((save continue)
  (save env)
  (assign proc (op lookup-variable-value) (const f) (reg env))
  (assign argl (const ()))
  (test (op primitive-procedure?) (reg proc))
  (branch (label label-1))
label-2
  (assign continue (label label-3))
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
label-1
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
label-3
  (restore env)
  (restore continue)
  (test (op false?) (reg val))
  (branch (label label-4))
label-5
  (save continue)
  (save env)
  (assign proc (op lookup-variable-value) (const g) (reg env))
  (assign argl (const ()))
  (test (op primitive-procedure?) (reg proc))
  (branch (label label-6)))
```

```
label-7
(assign continue (label label-8))
(assign val (op compiled-procedure-entry) (reg proc))
(goto (reg val))
label-6
(assign val (op apply-primitive-procedure) (reg proc) (reg argl))
label-8
(restore env)
(restore continue)
(test (op false?) (reg val))
(branch (label label-9))
label-10
(assign val (op lookup-variable-value) (const p) (reg env))
(goto (reg continue))
label-9
(assign val (op lookup-variable-value) (const q) (reg env))
(goto (reg continue))
label-11
label-4
(assign val (op lookup-variable-value) (const false) (reg env))
(goto (reg continue))
label-12))
```

Merk op dat de labels “anoniem” werden gemaakt. Gevraagd wordt de parameters te reconstrueren van de compile aanroep die deze code heeft gegenereerd.