

Vrije Universiteit Brussel – Faculteit Wetenschappen – Vakgroep
Computerwetenschappen
Academiejaar 2009 – 2010: tweede examenzittijd
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
28 augustus 2010*

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

Voeg aan de meta-circulaire evaluator van Scheme de *special form* `freeze` toe. Als je deze operatie toepast op een variabele, wordt het onmogelijk om die variabele destructief aan te passen. Bijvoorbeeld, na een `(freeze a)` op een bestaande variabele `a` krijg je een foutmelding bij `(set! a 42)`.

Let op dat de scoping regels gerespecteerd blijven; het volgende voorbeeld mag geen fout melden.

```
;;; M-Eval input:
(define a 2)
(define b 3)
(define c 4)
(freeze a)
(freeze c)
(define (set-fn! x y)
  (define a 0)
  (set! a x)
  (set! b y)
  (cons a b c))
(set-fn! 0 1)
```

```
;;; M-Eval output:
(0 1 4)
```

Maar als je hier nog het volgende aan toevoegt, moet je een foutmelding krijgen op `b`:

```
;;; M-Eval input:
(begin
  (freeze b)
  (set-fn! 5 6))
ERROR -- Trying to assign a frozen variable b
```

Geef je antwoord in de vorm van broncode. Je mag niet aangepaste broncode van de originele metacirculaire evaluator voorstellen met '...' wanneer de context duidelijk is.

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

Beschouw volgende beweringen over directe vliegtuigverbindingen tussen enkele steden:

(verbinding Brussel Praag)
(verbinding Amsterdam Madrid)
(verbinding Praag Madrid)
(verbinding Madrid Lausanne)
(verbinding Brussel Lausanne)
(verbinding Amsterdam Glasgow)

- a) Schrijf een predicaat (rechtstreeks ?x ?y) dat alle mogelijke rechtstreekse verbindingen (in beide richtingen) geeft tussen ?x en ?y.
- b) Schrijf een predicaat (onrechtstreeks ?x ?y) dat alle mogelijke onrechtstreekse verbindingen met één overstap geeft tussen ?x en ?y.
- c) Voorspel de resultaten van de volgende queries:

(onrechtstreeks Brussel ?z)

(onrechtstreeks ?x Amsterdam)

- d) Optimaliseer je oplossing zodanig dat resultaten waarbij vertrek en bestemming identiek zijn zoals (onrechtstreeks Praag Praag) uit het antwoord gefilterd worden.

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

Brainfuck is een programmeertaal die rond 1993 door Urban Müller is gemaakt. De taal is gebaseerd op een zeer eenvoudige machine die buiten het programma bestaat uit een rij van getallen die geïnitieerd worden op nul, dit is het geheugen (memory). Verder is er een index naar het geheugen (memory-index) (de startwaarde van deze index is ook nul). Het programma zelf bestaat uit een sequentie van instructies die je kan beschouwen als een vector van karakters (program). Zeven instructies van brainfuck, die elk bestaan uit een enkel karakter, zijn weergegeven in de volgende tabel.

Karakter	Betekenis	Scheme pseudo code
>	verhoog de memory index.	(set! memory-index (+ memory-index 1))
<	verlaag de memory index.	(set! memory-index (- memory-index 1))
+	verhoog de waarde waar de index naar wijst met 1.	(vector-set! memory memory-index (+ (vector-ref memory memory-index) 1))
-	verlaag de waarde waar de index naar wijst met 1.	(vector-set! memory memory-index (+ (vector-ref memory memory-index) 1))
.	print de waarde waar de index naar wijst als ASCII-output.	(display-char (vector-ref memory memory-index))

[	spring voorwaarts naar het statement na de corresponderende] indien de waarde waar de index naar wijst een nul is.	(if (= (vector-ref memory memory-index) 0) (ga-naar-het-corresponderende-sluit-haakje) (next-instruction))
]	spring terug naar het statement achter de corresponderende [indien de waarde waar de index naar wijst geen nul is.	(if (not (= (vector-ref memory memory-index) 0)) (spring-terug) (next-instruction))

Dit is een voorbeeld van een programma dat twee getallen optelt:

```

; neem aan dat memory-index 0 is
[
; loop
-
; Trek van memory[0] één af (omdat memory-index = 0)
>
; zet de memory-index op 1
+
; Tel bij memory[1] één op
<
; zet de memory-index terug op 0
]
; als memory[0] gelijk is aan nul stop anders spring naar loop

```

Vervolledig de registermachine hieronder zodat zij de zeven instructies van de tabel implementeert.

```

(define brainfuck-machine
  (make-machine
    '(program program-counter memory memory-index instruction tmp)

    (list (list '+ +) (list '- -)(list '= eq?) (list '> >)
      (list '!= (lambda (x y) (not (eq? x y))))
      (list 'make-vector make-vector)
      (list 'vector-length vector-length)
      (list 'vector-ref vector-ref)
      (list 'display display)
      (list 'vector-set! vector-set!)
      (list 'display-char (lambda (x) (display (integer->char x)))))

    '(initialise
      (assign program-counter (const 0))
      (assign memory-index (const 0))
      (assign memory (op make-vector) (const 100))
      (goto (label dispatch-instruction))
      next-instruction
      (assign program-counter (op +) (reg program-counter) (const 1))
      dispatch-instruction
      (assign tmp (op vector-length) (reg program))
      (test (op =) (reg program-counter) (reg tmp))
      (branch (label bf-done))
      (assign instruction (op vector-ref) (reg program) (reg program-counter))
      (test (op =) (reg instruction) (const #\>))
      (branch (label increase-memory-index))
      (test (op =) (reg instruction) (const #\<))
      (branch (label decrease-memory-index))
      (test (op =) (reg instruction) (const #\+))
      (branch (label increase-memory-location))
      (test (op =) (reg instruction) (const #\ -))
      (branch (label decrease-memory-location))

```

```
(test (op =) (reg instruction) (const #\.)  
(branch (label print-memory))  
(test (op =) (reg instruction) (const #\[))  
(branch (label start-loop))  
(test (op =) (reg instruction) (const #\]))  
(branch (label end-loop))  
(goto (label next-instruction))  
  
increase-memory-index  
(assign memory-index (op +) (reg memory-index) (const 1))  
(goto (label next-instruction))  
  
bf-done)))
```

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

Het Scheme geheugen dat in dit deel besproken wordt dient niet enkel voor het opslaan van cons-cellen die worden aangemaakt tijdens de evaluatie van een programma geschreven door de Scheme programmeur. Het wordt ook gebruikt om geheugen nodig voor de evaluator zelf op te slaan.

- a) geef aan waar ergens in de expliciete controle evaluator geheugen wordt aangemaakt (voor intern gebruik van de evaluator) dat uiteindelijk zal moeten beheerd worden door het geheugenbeheerssysteem;
 - b) geef aan onder welke omstandigheden dergelijk geheugen zal vrijgemaakt worden door de garbage collector;
 - c) heeft het zin om deze twee vormen van geheugen (één voor de evaluator, één voor het te evalueren programma) op te splitsen in twee aparte systemen?
-

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

Het volgende registermachine codefragment werd gegenereerd door aanroep van (compile exp 'val 'next):

```
((assign val (op make-compiled-procedure) (label Label1) (reg env))
(goto (label Done))
Label1
(assign env (op compiled-procedure-env) (reg proc))
(assign env (op extend-environment) (const (x y)) (reg argl) (reg env))
(save continue)
(assign proc (op lookup-variable-value) (const >) (reg env))
(assign val (op lookup-variable-value) (const y) (reg env))
(assign argl (op list) (reg val))
(assign val (op lookup-variable-value) (const x) (reg env))
(assign argl (op cons) (reg val) (reg argl))
(test (op primitive-procedure?) (reg proc))
(branch (label Label3))
Label2
(assign continue (label Label4))
(assign val (op compiled-procedure-entry) (reg proc))
(goto (reg val))
Label3
(assign val (op apply-primitive-procedure) (reg proc) (reg argl))
Label4
(restore continue)
(test (op false?) (reg val))
(branch (label Label6))
Label5
(assign val (const 1))
(goto (reg continue))
Label6
(assign val (const 2))
(goto (reg continue))
Label7
Done))
```

We hebben daarbij wel alle labels “anoniem” gemaakt. Gevraagd wordt om (1) de oorspronkelijke uitdrukking `exp` te reconstrueren (dit heel *decompileren*) en (2) de belangrijkste functies op te sommen die door de vertaler opgeroepen werden om dit resultaat te produceren.