

Schriftelijk Examen:

“Interpretatie van Computerprogramma’s I”

Eerste zittijd, academiejaar 2016-2017

Coen De Roover

Vrije Universiteit Brussel

26 juni 2017

Het examen bestaat uit 5 vragen, verdeeld over 6 bladzijden.

Gelieve elke vraag te beantwoorden op een afzonderlijk blad.

Zet je naam, studierichting en rolnummer op *elk* blad dat je afgeeft.

De vragen zijn geformuleerd ten opzichte van de gepubliceerde slides en het referentieboek. Deze mogen, samen met *persoonlijke* nota's, geraadpleegd worden tijdens deze test.

Het gebruik van elektronische media is echter *niet* toegestaan.

Veel succes!

1 Meta-circulaire evaluator

Breid de meta-circulaire evaluator uit met ondersteuning voor expressies van de vorm (`cas <VAR> <EXPECTED-EXP> <NEW-EXP>`). De semantiek van zulke `cas`-expressies is als volgt: indien de waarde van variabele `<VAR>` gelijk is aan de waarde van expressie `<EXPECTED-EXP>`, wordt de waarde van expressie `<NEW-EXP>` toegekend aan variabele `<VAR>`, en evalueert de volledige expressie naar de oude waarde van de `<VAR>`. Anders wordt er *geen* nieuwe waarde aan `<VAR>` toegekend, wordt `<NEW-EXP>` *niet* geëvalueerd, en evalueert de volledige `cas`-expressie naar de ongewijzigde waarde van `<VAR>`.

```
1  ;;; M-Eval input:
2  (define x 1)
3  ;;; M-Eval value:
4  'ok
5  ;;; M-Eval input:
6  x
7  ;;; M-Eval value:
8  1
9  ;;; M-Eval input:
10 (cas x 1 (+ 2 20))
11 ;;; M-Eval value:
12 1
13 ;;; M-Eval input:
14 x
15 ;;; M-Eval value:
16 22
17 ;;; M-Eval input:
18 (cas x 1 (+ 3 30))
19 ;;; M-Eval value:
20 22
21 ;;; M-Eval input:
22 x
23 ;;; M-Eval value:
24 22
25 ;;; M-Eval input:
26 (cas x 22 (+ 4 40))
27 ;;; M-Eval value:
28 22
29 ;;; M-Eval input:
30 x
31 ;;; M-Eval value:
32 44
```

- 1.a) Implementeer alle hulpfuncties die je nodig hebt om de meta-circulaire evaluator uit te breiden met ondersteuning voor `cas`-expressies.
- 1.b) Breid de meta-circulaire evaluator uit met ondersteuning voor `cas`-expressies. Implementeer `cas` als afgeleide expressie.
- 1.c) Breid de meta-circulaire evaluator opnieuw uit met ondersteuning voor `cas`-expressies. Implementeer `cas` als een “special form” met een “dedicated evaluator” procedure.

2 Logisch Programmeren

Beschouw volgende feitenbank over voedingswaren en allergenen:

```
1  ;;; (product naam bestanddelenlijst kostprijs-in-euro)
2  (product koek (graan suiker plantaardige-olie zout melkpoeder) 2.9)
3  (product cola (sprankelend-water suiker citroensap E150d E338 E300) 4.2)
4  (product keukenzout (zout) 0.5)
5
6  ;;; (allergeen bestanddeel allergie-of-intolerantie)
7  (bevat-allergeen graan gluten-allergie)
8  (bevat-allergeen melk lactose-intolerantie)
9  (bevat-allergeen melkpoeder lactose-intolerantie)
10 (bevat-allergeen kaas lactose-intolerantie)
11 (bevat-allergeen pinda pinda-allergie)
```

2.a) Geef een *logische query* die alle producten vindt die meer dan €2,5 kosten.

2.b) Implementeer het predicaat `product-met-allergenen` aan de hand van een *logische regel*, zodat `(product-met-allergenen ?prod)` slaagt voor elk product waarvan minstens één bestanddeel een gekend allergeen is. Met bovenstaande feitenbank is dan volgende interactie mogelijk:

```
1  ;;; Query input:
2  (product-met-allergenen ?p)
3  ;;; Query results:
4  (product-met-allergenen koek)
5  (product-met-allergenen koek)
```

2.c) Implementeer het predicaat `product-met-meeste-bestanddelen` aan de hand van een *logische regel*, zodat `(product-met-meeste-bestanddelen ?prod)` enkel slaagt voor het product met de meeste bestanddelen.

Je kan hiervoor een hulpregel (`lijst-langer ?a ?b`) definiëren die slaagt wanneer `?a` en `?b` lijsten zijn, en `?a` meer cons-cellen bevat dan `?b`.

Met bovenstaande feitenbank is dan volgende interactie mogelijk:

```
1  ;;; Query input:
2  (product-met-meeste-bestanddelen ?p)
3  ;;; Query results:
4  (product-met-meeste-bestanddelen cola)
```

3 Automatisch geheugenbeheer

Beschouw de Scheme-versie van de stop-and-copy garbage collector uit de slides. Veronderstel dat voor een specifieke toepassing het geheugensysteem met een bepaald adres <BARRIER> geconfigureerd kan worden zodat:

- alle cons-cellen met een adres kleiner dan <BARRIER> hun adres bewaren tijdens een garbage collection,
- indien de car en cdr van een dergelijke cons-cel een adres bevat, dan is dit adres steeds kleiner dan <BARRIER>.

Pas de code van de stop-and-copy garbage collector aan zodat die, gebruikmakende van bovenstaande informatie, geen overbodige operaties uitvoert voor zulk een geheugensysteem.

4 Programmeren van registermachines

Beschouw volgende definitie van een iteratieve procedure `collatz` die het aantal zogenaamde Collatz-stappen berekent voor een gegeven getal `n`, en daarbij beroep doet op een hulpprocedure `even`:

```
1 (define (even n)
2   (if (= n 0)
3       1
4       (if (= n 1)
5           0
6           (even (- n 2)))))
7
8 (define (collatz n)
9   (define (collatz-iter n ctr)
10    (if (= 1 n)
11        ctr
12        (collatz-iter (if (= (even n) 1)
13                          (/ n 2)
14                          (+ (* n 3) 1))
15                      (+ ctr 1))))
16   (collatz-iter n 0))
```

(vraag wordt vervolgd op volgende bladzijde)

Vul onderstaande registermachinecode aan vanaf lijn 24 met een vertaling voor de procedure collatz. Procedure even hebben we al voor jou vertaald. Je mag geen extra primitieve operaties of registers aan de registermachine toevoegen. Probeer spaarzaam te zijn met instructies en met het gebruik van de stapel.

```
1 (make-machine
2   '(cont res v1 v2)
3   `((= ,=) (+ ,+) (* ,*) (- ,-) (/ ,/))
4   '(start
5     (assign v1 (const 10))
6     (assign cont (label stop))
7     (goto (label collatz-start))
8
9     even-start
10    (test (op =) (reg v1) (const 0))
11    (branch (label even-true))
12    (test (op =) (reg v1) (const 1))
13    (branch (label even-false))
14    (assign v1 (op -) (reg v1) (const 2))
15    (goto (label even-start))
16    even-true
17    (assign res (const 1))
18    (goto (reg cont))
19    even-false
20    (assign res (const 0))
21    (goto (reg cont))
22
23    collatz-start
24    ...
25
26    stop
```

5 Compilatie

Onderstaand codefragment werd gegenereerd door een oproep van de procedure `compile`. Reconstrueer de argumenten van de oproep. Merk op dat we de labels geanonimiseerd hebben.

```
1  (save env)
2  (assign proc (op lookup-variable-value) (const p) (reg env))
3  (assign val (const 1))
4  (assign arg1 (op list) (reg val))
5  (test (op primitive-procedure?) (reg proc))
6  (branch (label label-7))
7  label-8
8  (assign continue (label label-9))
9  (assign val (op compiled-procedure-entry) (reg proc))
10 (goto (reg val))
11 label-7
12 (assign val (op apply-primitive-procedure) (reg proc) (reg arg1))
13 label-9
14 (restore env)
15 (test (op false?) (reg val))
16 (branch (label label-5))
17 label-4
18 (assign val (op lookup-variable-value) (const false) (reg env))
19 (goto (label label-6))
20 label-5
21 (assign val (const 4))
22 label-6
23 (test (op false?) (reg val))
24 (branch (label label-2))
25 label-1
26 (assign proc (op lookup-variable-value) (const q) (reg env))
27 (assign arg1 (const ()))
28 (test (op primitive-procedure?) (reg proc))
29 (branch (label label-10))
30 label-11
31 (assign continue (label label-3))
32 (assign val (op compiled-procedure-entry) (reg proc))
33 (goto (reg val))
34 label-10
35 (assign val (op apply-primitive-procedure) (reg proc) (reg arg1))
36 (goto (label label-3))
37 label-12
38 label-2
39 (assign val (op lookup-variable-value) (const false) (reg env))
40 label-3
```
