Schriftelijk Examen
Interpretatie van Computerprogramma's I

Coen De Roover

22/06/2026 (reconstructie)

Eerste zittijd academiejaar 2025–2026.

Het examen bestaat uit 4 vragen. Gelieve elke vraag te beantwoorden op een afzonderlijk blad.

Gesloten boek. Bundel met eval-code beschikbaar.

1. Meta-circulaire Evaluator

Beschouw een special form met de syntaxis `(memf <var> <pred-exp> <lst>)`. Deze expressie bindt `<var>` opeenvolgend aan elk element van de lijst waarnaar `<lst>` evalueert. Hierbij is `<pred-exp>` een expressie waarin `<var>` gebruikt kan worden.

Eenmaal `<pred-exp>` slaagt voor zo'n element gebonden aan `<var>` (d.w.z. `<pred-exp>` evalueert naar een waarde verschillend van `#f`), wordt het element en alles wat daarop volgt in de lijst teruggegeven.

Indien `<pred-exp>` voor geen enkel element slaagt, wordt `#f` teruggegeven. Het resultaat van de special form is dus een (sub)lijst van de originele lijst, of `#f`. Merk op dat het bereik van `<var>` beperkt is tot `<pred-exp>`.

Onderstaande interactie licht het gedrag van de special form toe:

```
;;; M-eval input:
(memf arg (> arg 2) '(1 2 3 4 1))

;;; M-eval value:
(3 4 1)

;;; M-eval input:
(memf arg (> arg 4) '(1 2 3 4 1))

;;; M-eval value:
#f
```

1a) Implementeer alle hulpprocedures die de meta-circulaire evaluator nodig heeft om `memf`-expressies te herkennen en er onderdelen uit te selecteren.

1b) Breid de meta-circulaire evaluator uit met ondersteuning voor `memf`-expressies via een dedicated evaluator procedure, bijvoorbeeld *eval-memf*.

1c) Breid de meta-circulaire evaluator uit met ondersteuning voor `memf`-expressies, ditmaal als een afgeleide expressie.

2. Registermachines

Gegeven is de procedure `countDigits`, die telt hoe vaak een bepaald cijfer `d` voorkomt in een getal `n`:

```
(define (countDigits n d)
  (if (= n d)
      1
      (if (> n 9)
          (+ (countDigits (quotient n 10) d)
             (countDigits (modulo n 10) d))
          0)))
```

Toelichting gebruik:

```
> (countDigits 417477 7)
3
```

Zet deze procedure om naar een subroutine in registermachinecode. Het recursieve karakter moet behouden blijven. De registers hebben de volgende functie: `n` en `d` dienen als input, `res` dient voor de tussenresultaten en het uiteindelijke resultaat, en `cont` wordt gebruikt om uit de subroutines te springen. Je mag geen extra registers gebruiken. Vul de subroutine `countDigits` aan in de onderstaande machine-definitie:

```
(define countDigits-machine
  (make-machine
   '(cont n d res)
   `((= ,=) (> ,>) (+ ,+) (quotient ,quotient) (modulo ,modulo))
   '(start
      (assign cont (label stop))

      ;;; <jouw code hier>

      stop)))

(set-register-contents! countDigits-machine 'n 417477)
(set-register-contents! countDigits-machine 'd 7)
(start countDigits-machine)
(display (get-register-contents countDigits-machine 'res))
```

3. Open vragen: inzicht

3a) M-eval: Wat gebeurt er als we de `apply`-procedure van de meta-circulaire evaluator als volgt aanpassen?

```
((compound-procedure? procedure)
 (eval-sequence
   (procedure-body procedure)
   (extend-environment
     (procedure-parameters procedure)
     arguments
     the-global-environment))) ; aangepast
```

Beschouw vervolgens het onderstaande codefragment dat in deze evaluator wordt ingevoerd:

```
(begin
  (define (add x)
    (lambda (y) (+ x y)))

  (define add-5 (add 5))

  (add-5 10))
```

Wat is het verschil in output tussen de aangepaste en de originele versie? Hoe verklaar je dit?

3b) A-eval: Bij `analyze-assignment` doen we de volgende aanpassing. Is deze aanpassing even goed? Motiveer je antwoord.

```
(define (analyze-assignment exp)
  (lambda (env)
    (set-variable-value! (assignment-variable exp)
                         ((analyze (assignment-value exp)) env)
                         env)
    'ok))
```

3c) L-eval: Wat is de uiteindelijke waarde van de variabele `count` bij uitvoering van het onderstaande codefragment in L-eval met `force-it` zonder memoization?

```
(define count 0)

(define (increment)
  (set! count (+ count 1))
  count)

(define result ((lambda (x) (+ x x)) (increment increment)))

count
```

3d) EC-eval: Beschouw de volgende aanpassing. Geef een expressie waarbij dit misloopt en leg uit waarom.

```
ev-if-consequent
  ;; (assign exp (op if-consequent) (reg exp))
  (goto (label eval-dispatch))
```

3e) Compilers: Reconstrueer de argumenten voor de oproep van `compile` voor de volgende compiled code.

```
((assign val (op lookup-variable-value) (const z) (reg env))
 (test (op false?) (reg val))
 (branch (label false-branch2))
 true-branch1
 (assign foo (op lookup-variable-value) (const y) (reg env))
 (goto (label after-if33))
 false-branch2
 (assign foo (op lookup-variable-value) (const x) (reg env))
 (goto (label after-if33))
 after-if3)
```

4. Logisch Programmeren

Gegeven is een feitenbank met WK's, aangemaakt via `assert!`. De feitenbank is zo opgebouwd dat elke gevraagde query of regel minstens één succesvolle match oplevert:

```
(assert! (speler Messi Argentinie aanval))
(assert! (speler Dybala Argentinie aanval))
(assert! (speler Martinez Argentinie doel))
(assert! (speler Yamal Spanje aanval))
(assert! (speler Pedri Spanje mid))
(assert! (speler Llorente Spanje mid))
(assert! (speler Hazard Belgie aanval))
(assert! (speler Courtois Belgie doel))
(assert! (speler Rashin Belgie mid))

(assert! (wereldkampioen 2022 Argentinie))
(assert! (wereldkampioen 2026 Belgie))

(assert! (selectie Argentinie 2022 (Messi Dybala Martinez)))
(assert! (selectie Argentinie 2026 (Messi Martinez)))
(assert! (selectie Belgie 2022 (Courtois Hazard)))
(assert! (selectie Belgie 2026 (Courtois Rashin)))
(assert! (selectie Spanje 2022 (Pedri Llorente)))
(assert! (selectie Spanje 2026 (Yamal Pedri Llorente)))
```

4a) Schrijf een logische query om te bepalen of een speler een middenvelder is en in de ploeg zat van het land dat in een bepaald jaartal wereldkampioen werd.

4b) Gebruik een logische regel om het predicaat `(speler-in-selectie ?speler ?selectie)` te implementeren.

```
(speler-in-selectie ?p (Courtois Hazard))

;;; Query results:
(speler-in-selectie Courtois (Courtois Hazard))
(speler-in-selectie Hazard (Courtois Hazard))
```

4c) Gebruik een logische regel om het predicaat `(wk-veteran ?speler)` te implementeren, dat slaagt wanneer een speler minstens twee keer in een selectie zat.

4d) Schrijf een logische query die de spelers oplevert die wel in een selectie van 2022 zaten, maar niet meer in een selectie van 2026. Duplicaten negeren mag.

```
;;; Q-eval output
(speler Dybala Argentinie aanval)
(speler Hazard Belgie aanval)
```