TÖL304G
Forritunarmál — Programming Languages
Vikublað 5 — Weekly 5

Snorri Agnarsson

13. september 2024

Efnisyfirlit

1 Efni vikunnar — The Weeks Material

Við höldum áfram með Scheme, lokanir, bálkmótun og strauma. Vakningarfærslur (*activation records*) eru lykillinn að skilningi á þessum fyrirbærum. Nauðsynlegt er að skilja hvernig vakningarfærslur eru notaðar og hvernig þær tengjast saman í keðjur gegnum stýrihlekki (*control link, dynamic link*) annars vegar, og gegnum tengihlekki (aðgangshlekki, *access link, static link*) hins vegar.

Einnig þarf að skilja hverjar afleiðingarnar eru af því að geyma vakningarfærslur á hlaða (*stack*), annars vegar, og í kös (*heap*), hins vegar. Ruslasöfnun minnis kemur einnig inn í dæmið, eins og minnst hefur verið á.

We continue with Scheme, closures, block structure and streams. Activation records are the key to understanding these phenomena. It is necessary to understand how

activation records are used and how they are chained together through control links (dynamic links) on one hand, and through access links (static links) on the other hand.

It is also necessary to understand what the consequences are from storing the activation records on the stack, on one hand, and in the heap, on the other hand. Garbage collection of memory also plays a role as has been mentioned.

2 Fikt með lokanir í Scheme — Tricks With Closures in Scheme

2.1 xcons, xcar og xcdr

Íhugum eftirfarandi í Scheme — Consider the following in Scheme.

```
;; Notkun: (xcons x y)
;; Fyrir:  x og y eru hvaða gildi sem er.
;; Gildi:  Fall (lokun) c með eftirfarandi lýsingu:
;;         Notkun: (c z)
;;         Fyrir:  z er satt eða ósatt (#t eða #f).
;;         Gildi:  x ef z er satt, y annars.
;; Usage:  (xcons x y)
;; Pre:    x and y are any values.
;; Value:  A function (closure) c with the following
;;         description:
;;         Usage:  (c z)
;;         Pre:    z is true or false (#t or #f).
;;         Value:  x if z is true, y otherwise.
(define (xcons x y) (lambda (z) (if z x y)))

;; Notkun (xcar (xcons x y))
;; Fyrir: x og y eru hvaða gildi sem er.
;; Gildi: x.
;; Usage  (xcar (xcons x y))
;; Pre:   x and y are any values.
;; Value: x.
(define (xcar c) (c #t))

;; Notkun (xcdr (xcons x y))
;; Fyrir: x og y eru hvaða gildi sem er.
;; Gildi: y.
;; Usage  (xcdr (xcons x y))
```

```
;; Pre:   x and y are any values.
;; Value: y.
(define (xcdr c) (c #f))
```

Við sjáum að virkni þessa falla er svipuð virkni innbyggðu fallanna `cons`, `car` og `cdr`. Lykilatriði hér er að ljóst er að fallið `xcons` pakkar greinilega saman tveimur gildum í eitt, og föllin `xcar` og `xcdr` sækja þau gildi. Það eru sömu vensl milli fallanna `xcons`, `xcar` og `xcdr` og eru milli fallanna `cons`, `car` og `cdr`.

We see that these function work similarly to the built-in functions `cons`, `car` and `cdr`. A key fact is that the function `xcons` packages two values into one value and the functions `xcar` and `xcdr` fetch those values. We have the same relationships between the functions `xcons`, `xcar` and `xcdr` as between the functions `cons`, `car` and `cdr`.

```
(xcar (xcons x y)) = x
(xcdr (xcons x y)) = y

(car (cons x y)) = x
(cdr (cons x y)) = y
```

Jafnaðarmerkið þýðir að segðin vinstra megin skilar sama gildi og segðin hægra megin. `x` og `y` geta verið hvaða gildi sem er. Ef hliðarverkanir eru ekki til staðar (eins og við forðumst í Scheme) þá mega `x` og `y` vera hvaða segðir sem er.

Við sjáum því að föllin `cons`, `car` og `cdr` eru í raun óþarfi sem grunnföll því við gætum búið til þeirra virkni út frá annarri grunnvirkni í Scheme. Hins vegar er hraðvirkara að nota þau og auk þess fáum við að nota fallið `pair?` sem segir til um hvort eitthvert tiltekið gildi er par.

Athugið að ef við hefðum ekki þann möguleika í Scheme að skila lokun sem gildi falls þá væri þetta ekki hægt.

The equality sign means that the expression on the left returns the same value as the expression on the right. `x` and `y` can be any values. If there are no side effects (which we will avoid in Scheme) then `x` and `y` can be any expressions.

Notice that if we did not have the possibility in Scheme to return a closure as a function value then this would not be possible.

2.2 Straumar — Streams

Straumar í Scheme eru skemmtileg fyrirbæri. Sumar útgáfur Scheme hafa innbyggða virkni fyrir strauma en í öðrum útgáfum er auðvelt að útfæra þá. Straumar, svipað og listar, byggja á samspili þriggja aðgerða sem hafa eftirfarandi eiginleika:

Streams, similarly to lists, are based on the interaction of three operations that have the following properties:

```
(stream-car (cons-stream x y)) = x
(stream-cdr (cons-stream x y)) = y
```

Jafnaðarmerkið þýðir, eins og lengra að ofan, að segðin vinstra megin skilar sama gildi og segðin hægra megin. `x` og `y` geta verið hvaða segðir sem er. Munurinn á virkninni á `cons`, `car` og `cdr` annars vegar, og `cons-stream`, `stream-car` og `stream-cdr` hins vegar, er að `cons-stream` er ekki fall, heldur lykilorð, og seinna viðfangið í `cons-stream` er ekki reiknað fyrr en fallinu `stream-cdr` er beitt á útkomuna úr segðinni `(cons-stream x y)`. Þetta veldur því að hægt er að reikna skilgreiningar eins og þessa:

```
(define e (cons-stream 1 e))
```

Þetta býr til óendanlegan straum af `1` því segðin `(stream-car e)` skilar `1`, og segðin `(stream-car (stream-cdr e))` skilar einnig `1` vegna þess að segðin `(stream-cdr e)` skilar `e`, og svo koll af kolli.

Á vefnum[1] má finna skjal um „óendanlega“ strauma í Scheme, ásamt Scheme forritstexta[2] fyrir föllin þar.

Útfæra má strauma á ýmsan hátt, en þægilegt er að gera það þannig að eftirfarandi jafngildi séu notuð:

Streams can be implemented in various ways, but it is convenient to base the implementation on the following equivalences:

```
(cons-stream x y) = (cons x (delay y))
(stream-car z) = (car z)
(stream-cdr z) = (force (cdr z))
```

Lykilorðið `delay` virkar þannig að segðin `(delay e)`, þar sem `e` er hvaða segð sem er, skilar svokölluðu *loforði* (promise), sem inniheldur segðina `e`, án þess að segðin hafi enn verið gilduð (evaluated). Sé fallinu `force` seinna beitt á þetta loforð þá verður segðin `e` gilduð og gildinu skilað og lagt á minnið. Við höfum því þetta jafngildi:

The keyword `delay` has the effect that the expression `(delay e)`, where `e` is any expression, returns a so-called *promise*, which contains the expression `e`, without immediately evaluating the expression. If the function `force` is later applied to the promise then the expression `e` is evaluated and its value returned and memorized. We have the following equivalence:

```
(force (delay e)) = e
```

[1] http://cs.hi.is/snorri/downloads/straumar.pdf

[2] http://cs.hi.is/snorri/downloads/straumar.s

Sé fallinu `force` beitt aftur er skilað sama gildi án þess að það þurfi að endurreikna það. Við munum seinna sjá svipaða virkni í forritunarmálinu Haskell, en í því tilviki er það hluti af grundvallarvirkni forritunarmálsins og krefst ekki beitingu neinna sérstakra aðgerða.

Í framhaldinu, í Scheme, má auðveldlega útfæra föllin `stream-car` og `stream-cdr` svona:

If the function `force` is applied again to the same promise, the same memorized value is returned without computing it again. We will lated see similar effects in the programming language Haskell, but in that case this lazy evaluation is part of the basic structure of the programming language and does not need any special syntax or special operations.

Subsequently, in Scheme, we may easily implement the functions `stream-car` and `stream-cdr` like this:

```
(define stream-car car)
(define (stream-cdr z) (force (cdr z)))
```

Hins vegar þarf aðrar aðferðir til að útfæra `cons-stream` því það er ekki fall heldur lykilorð, eins og `delay`. Allar útfærslur Scheme hafa einhverjar aðferðir til að útfæra ný lykilorð, en við munum ekki fjalla um slíkt, eða a.m.k. er ekkert slíkt til prófs.

On the other hand we need other methods to implement `cons-stream` because it is not a function but rather a keyword, like `delay`. All versions of Scheme have some functionality to implement new keywords, but students in this course are not required to delve into that.

3 Rökstudd forritun

Finna má á vefnum[3] skjal um röksemdafærsluaðferðir í forritun. Þar eru meðal annars dæmi um notkun á ýmsum gerðum stöðulýsinga sem mikilvægar eru í rökstuddri forritun. Í þessu námskeiði verður gerð sú krafa að í verkefnum og á prófi séu úrlausnir skjalaðar með slíkum stöðulýsingum.

Krafan er sú að öll föll hafi lýsingu þar sem fram komi forskilyrði, eftirskilyrði og hvernig kalla skuli á fallið. Síðar, þegar við hefjumst handa við hlutbundna forritun og einingaforritun, þá verður krafist fastayrðingar gagna (data invariant) fyrir sérhvert nýtt gagnamót (data structure) sem skilgreint er.

[3] http://cs.hi.is/snorri/downloads/rokjava.pdf