TÖL304G
Forritunarmál — Programming Languages
Vikublað 4 — Weekly 4

Snorri Agnarsson

7. september 2024

Efnisyfirlit

1 Vakningarfærslur — Activation Records

Vakningarfærslur (*activation records, stack frames*) í Scheme eru geymdar í kösinni (*heap*). Annars hefðum við ekki getað leyst öll verkefnin á verkefnablaði 3. Við íhugum hvernig vakningarfærslur og lokanir (*closures*) eru meðhöndlaðar í Scheme og öðrum forritunarmálum.

Activation records in Scheme are stored on the heap. Otherwise we would not have been able to solve all the problems in assignment sheet 3. We will consider how activation records and closures are handled in Scheme and other programming languages.

Vakningarfærsla er það minnissvæði sem einstök vakning af falli eða stefi notar meðan það keyrir. Öll forritunarmál sem hafa föll eða stef hafa vakningarfærslur á einn eða annan hátt. Í flestum tilfellum eru vakningarfærslur geymdar á hlaða (stack),

en það er ekki algilt. Í öðrum tilfellum eru vakningarfærslur í kös (heap), t.d. í Scheme, Haskell, Morpho og ML, eða geymdar í sama minnissvæði og víðværar (global) breytur, t.d. í sumum afbrigðum af FORTRAN og COBOL. Þessi staðsetning vakningarfærslanna hefur afgerandi áhrif á notkunarmöguleika stefja og falla í viðkomandi forritunarmáli.

An activation record is the memory area that a single activation of a function or procedure uses while it is running. All programming languages that have functions or procedures have activation records in some fashion. In most cases are stored on a stack, but this is not universal. In other cases activation records are on the heap, for example in Scheme, Haskell, Morpho and ML, or are stored in the same area as global variables, for example in some versions of FORTRAN and COBOL. Where the activation records are located has a crucial effect on the way functions and procedures can be used.

Í bálkmótuðum forritunarmálum (block-structured programming languages) innihalda vakningarfærslur (activation records) eftirfarandi upplýsingar:

In block-structured programming languages activation records contain the following information:

- Viðföng — Arguments.
- Staðværar breytur — Local variables.
- Vendivistfang — Return address.
- Stýrihlekk – Dynamic link, control link.
- Tengihlekk (aðgangshlekkur) — Access link (static link).

Ef vakningarfærslur eru geymdar á hlaða, sem algengast er þegar þetta er ritað (t.d. Í C, C++, Java, C#), bendir stýrihlekkurinn ávallt á næstu vakningarfærslu í hlaða, þ.e. vakningarfærslu þess falls sem kallaði á núverandi fall. Sama gildir í þeim fornu forritunarmálum sem einungis leyfðu mest eina samtímis vakningu á hverju falli.

If activation records are stored on a stack, which is most commonly the case when this is written (e.g. in C, C++, Java, C#), the control link always points to the next activation record on the stack, i.e. the activation record of the function that called the current function. The same holds in those ancient programming languages that only allowed one concurrent activation for each function.

Ef vakningarfærslur eru í kös má nota sömu aðferð með stýrihlekkinn, þ.e. láta hann ávallt benda á vakningarfærslu þess sem kallaði. En einnig kemur til greina að nota almennari aðferð sem gefur kost á almennri halaendurkvæmni, sem er sú aðferð að láta stýrihlekkinn benda á vakningarfærslu þess stefs sem á að fá niðurstöðuna úr núverandi kalli. Það stef þarf þá ekki endilega að vera stefið sem kallaði, þegar um halaendurkvæmni er að ræða. Vendivistfangið þarf þá að sjálfsögðu, til samræmis, að

benda á viðeigandi stað í þulu (code) þess falls sem snúið er til baka í, þ.e. þess falls sem fá skal niðurstöðuna úr núverandi kalli.

If activation records are on the heap the same method can used for the control link, i.e. have it always point to the activation record of the calling function. But it is also possible to use a more general method that supports general tail recursion, which is to let the control link point to the activation record of the function that should receive the result from the current call. That function does not necessarily have to be the function that called the current function, in the case of tail recursion. The return address must then, accordingly, point to the appropriate place in the machine code for the function being returned to, i.e. the function that should receive the result from the current call.

Tengihlekkurinn bendir ávallt á viðeigandi vakningarfærslu þess stefs sem inniheldur viðkomandi stef, textalega séð. Viðeigandi vakningarfærsla er ávallt sú vakningarfærsla, sem inniheldur breyturnar í næstu földunarhæð, og er næstum alltaf nýjasta vakningarfærsla ytra stefsins. Undantekningar frá þeirri reglu geta orðið til við flókna notkun á *lokunum* (*closures*).

The access link always points to an appropriate activation record of the function that encloses the current function, textually. The appropriate activation record contains variables in the enclosing scope and is almost always the newest activation record of that function. Exceptions from this rule happen when closures are used in complicated ways.

Vakningarfærslur í forritunarmálum, sem ekki eru bálkmótuð, s.s. C, C++, C# og Java, innihalda sömu upplýsingar og talið er upp að ofan, nema hvað tengihlekkur er ekki til staðar. Enda er tengihlekkur aðeins notaður til aðgangs að breytum í efri földunarhæðum, og er því merkingarlaus ef ekki er um bálkmótun að ræða.

Activation records in programming languages that are not block structured, such as C, C++, C# and Java, contain the same information as listed above, except that there is no access link. The access link is only used to access variables in enclosing scopes, and is therefore meaningless if there is no block structure.

2 Lokanir — Closures

Lokun (*closure*) er gildi, sem í bálkmótuðum málum er notað á sama hátt og fallsbendar eru notaðir í C og C++.

A closure is a value that, in block structured programming languages, is used in the same way as function pointers are used in C and C++.

Í Standard Pascal eru lokanir til staðar, en ekki í Object Pascal (Delphi) eða Turbo Pascal. Öll þrjú þessara forritunarmál hafa vakningarfærslur á hlaða. Dæmi um notkun og tilurð lokunar í Standard Pascal er eftirfarandi:

Standard Pascal has closures, but not Object Pascal (Delphi) or Turbo Pascal. All three of these programming languages have activation records on the stack. An example of the use and creation of a closure in Standard Pascal is the following.

```
type func = function( x: Real ): Real;
function root(f: func; a,b,eps: real): real;
begin
  if (b-a) < eps then
    root := a
  else if f(a)*f((a+b)/2.0) <= 0 then
    root := root(f,a,(a+b)/2.0,eps)
  else
    root := root(f,(a+b)/2.0,b,eps)
end;

function h(a,b: real): real;
  var x,y,eps: real;
  function g(x: real): real;
  begin
    g:=a*x+b;
  end;
begin
  ...  gefum x, y og eps viðeigandi gildi ...
  ...  give x, y and eps appropriate values ...
  h:=root(g,x,y,eps);
end;
```

Fallið sem er fyrsta viðfang í kallinu á `root` er lokun. Takið eftir að fallið `g` sem notað er sem viðfang í `root` er faldað inn í fallið `h` og notar staðværar breytur í efri földunarhæð. Þær breytur (`a` og `b`) eru til staðar uns kallinu á `h` lýkur. Eftir að kallinu á `h` lýkur eru þær ekki lengur til.

The function that is the first argument in the call to `root` is a closure. Notice that the function `g` which is used as an argument for `root` is nested inside the function `h` and uses local variables in the enclosing scope. Those variables (`a` and `b`) exist until the call to `h` finishes. After the call to `h` finishes they no longer exist.

Lokun inniheldur — A closure contains:

- Fallsbendi á vélarmálsþulu viðkomandi falls — A function pointer to the machine code for the function in question
- Aðgangshlekk, sem bendir á viðeigandi vakningarfærslu þess stefs, sem inniheldur viðkomandi fall, textalega séð — An access link that points to the appropriate activation record of the function that encloses that function, textually

Í eldri gerðum af bálkmótuðum forritunarmálum, s.s. Standard Pascal, er einungis hægt að nota lokanir sem viðföng í köll. Ekki er hægt að skila lokun sem niðurstöðu

úr kalli eða vista lokun í breytu. Ástæða þessarar takmörkunar er sú að aðgangshlekkurinn í lokuninni inniheldur tilvísun á vakningarfærslu. Sú vakningarfærsla er áreiðanlega til staðar þegar lokunin verður til og allt þar til bálkur sá sem lokunin verður til í lýkur keyrslu, en eftir það er mögulegt að vakningarfærslunni sé eytt. Lokunin er aðeins í nothæfu ástandi ef aðgangshlekkurinn vísar á vakningarfærslu sem til er.

In older sorts of block structured programming languages, such as Standard Pascal, it is only possible to use closures as arguments to function calls. It is not possible to return closures as a function result or to store a closure in a variable. The reason for this restriction is that the access link in the closure contains a pointer to an activation record. That activation record is certain to exist when the closure is created and until the block that contains the definition of the closure ceases running, but after that it is possible that the activation record is destroyed. The closure is only in a usable state if the access link points to an existing activation record.

Í Standard Pascal er t.d. *ekki* löglegt að skrifa:

In Standard Pascal it is, for example, *not* legal to write:

```
type adder = function( i: Integer ): Integer;
function newadder( k: Integer ): adder;
  function theadder( i: Integer ): Integer;
  begin
    theadder := k+i;
  end;
begin
  newadder := theadder; {þessi skipun gengur ekki}
end;
```

Nauðsynlegt er að nemendur skilji vel hvers vegna svona forrit eru ekki leyfð í Standard Pascal.

It is necessary for students to understand why such programs are not allowed in Standard Pascal.

Í λ-reikningi er ekkert vandamál að skilgreina svona fall:

In λ-calculus it is no problem to define such a function:

$$\text{newadder} = \lambda k.(\lambda i.i + k)$$

Í Object Pascal (Delphi) og gamla Turbo Pascal eru ekki lokanir. Í þeim forritunarmálum er ekki leyft að senda staðvær (*local*) föll sem viðföng, aðeins víðvær (*global*). Í Object Pascal má skrifa:

In Object Pascal (Delphi) and old Turbo Pascal there are no closures. In these programming languages it is not allowed to send local functions as arguments to functions, only global functions can be sent. In Object Pascal we can write:

```
type func = function( x: Real ): Real;
```

```
function root(f: func; a,b,eps: Real): Real;
begin
  if (b-a) < eps then
    root := a
  else if f(a)*f((a+b)/2.0) <= 0 then
    root := root(f,a,(a+b)/2.0,eps)
  else
    root := root(f,(a+b)/2.0,b,eps)
end;

var globala, globalb: Real;

function g(x: Real): Real;
begin
  g:=globala*x+globalb;
end;

function h(a,b: real): real;
  var x,y,eps: real;
begin
  ...  gefum x, y og eps viðeigandi gildi ...
  ...  give x, y and eps appropriate values ...
  globala := a;
  globalb := b;
  h:=root(g,x,y,eps);
end;
```

Þar má einnig skrifa:
There we can also write:

```
type adder = function( i: Integer ): Integer;
var globalk: Integer;
function theadder( i: Integer ): Integer;
begin
  theadder := globalk+i;
end;
function newadder( k: Integer ): adder;
begin
  globalk := k;
  newadder := theadder; {þessi skipun gengur hér}
end;
```

Eins og sjá má leyfir Object Pascal að föll séu vistuð í breytum og að skilað sé föllum.

En öll slík notkun á föllum takmarkast við víðvær föll. Slík fallsgildi eru *ekki* lokanir. Þar eð um víðvær föll er að ræða er engin þörf á aðgangshlekki.

As we can see Object Pascal allows functions to be stored in variables and allows functions to be returned as values from functions. But all such usage is restricted to global functions. Such function values are *not* closures. Sice the functions are global there is no need for access links.

Sum önnur bálkmótuð forritunarmál s.s. Scheme, ML, Morpho og Haskell hafa ekki þessa takmörkun á notkun lokana. Þau forritunarmál hafa ruslasöfnun (eins og Java, sem er ekki bálkmótað). Ruslasöfnun er nauðsynleg (en ekki nægjanleg) forsenda þess að unnt sé að nota lokanir á sveigjanlegan hátt. Og reyndar er það einnig nauðsynleg forsenda að vakningarfærslur taki þátt í ruslasöfnun. Í stað þess að vakningarfærslu sé skilað um leið og bálkur vakningarfærslunnar lýkur keyrslu lifir vakningarfærslan meðan til er einhver tilvísun á hana úr einhverjum lifandi lokunum.

Some other block structured programming languages such as Scheme, ML, Morpho and Haskell do not have these restrictions on the use of closures. These programming languages have garbage collection (like Java, which is not blokc structured). Garbage collection is a necessary (but not sufficient) premise for being able to use closures in a flexible way. And actually it is also a necessary premise that activation records take part in garbage collection. Instead of activation records being destroyed as soon as the block of the activation record terminates, the activation record survives while there is some reference to it from any existing closure.

3 Framhöld — Continuations

Við höfum séð að lokun inniheldur tengihlekk og fallsbendi. Til er annað skylt fyrirbæri sem kallast framhald (*continuation*). Framhald inniheldur stýrihlekk og vendivistfang. Í Scheme má vinna með framhöld og í öðrum forritunarmálum má oft líta svo á að framhöld séu notuð í innviðum á útfærslum á afbrigðameðhöndlun, s.s. try-catch í Java og C++.

We have seen that a closure contains an access link and a function pointer. There exists another related phenomenon called a continuation. A continuation is a value that contains a control link and a return pointer. In Scheme we can work with continuations and in other programming languages we can often take the view that continuations are used internally in implementing exception handling, such as try-catch in Java and C++.

4 Straumar í Scheme — Streams in Scheme

Á vefnum[1] má finna skjal um „óendanlega“ strauma í Scheme, ásamt Scheme forrits-

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

texta[2] fyrir föllin þar.

On the web we can find a document about "infinite"streams in Scheme, along with Scheme code[3] fyrir föllin þar.

[2]https://cs.hi.is/snorri/downloads/straumar.s eða https://cs.hi.is/snorri/downloads/straumar-utf8.s

[3]https://cs.hi.is/snorri/downloads/straumar.s eða https://cs.hi.is/snorri/downloads/straumar-utf8.s