TÖL304G
Forritunarmál — Programming Languages
Vikublað 7 — Weekly 7

Snorri Agnarsson

30. september 2024

Efnisyfirlit

1 Miðmisserispróf — Midterm Exam

Miðmisserisprófið verður haldið fimmtudaginn 10. október. Prófið verður í fyrirlestrartímanum í Öskju frá 10:00 til 11:30.

The midterm exam will be held on Thursday the 10^{th} of October. The exam will be during the lecture hours in Askja from 10:00 to 11:30.

2 Morpho

Við byrjum nú að leggja stund á Morpho. Aðalatriðin í notkun Morpho eru listavinnsla og einingaforritun. Listavinnslu höfum við kynnst áður í Scheme og CAML. Listavinnslan í Morpho er eins, en þó er sá munur að í Morpho er venjan frekar sú að nota

hliðarverkanir þegar henta þykir. Til dæmis notum við lykkjur óspart, og snúningur lista í Morpho gæti verið forritaður á eftirfarandi hátt:

We now start working with Morpho. The main issues in using Morpho are list processing and modular programming. We have already been acquainted with list processing in Scheme and CAML. The list processing in Morpho is similar, but one difference is that the convention is to use side effects when convenient. For example we use loops a lot and reversing a list in Morpho can be programmed as follows:

```
z=x; y=[];
;;; x=z=[x1,...,xN], y=[]
while( z!=[] )
{
  ;;; Fastayrðing lykkju / Loop invariant:
  ;;; z=[xI,...,xN], y=[xI-1,...,x1]
  y = head(z) : y;
  z = tail(z);
};
;;; y=[xN,...,x1]
```

Þetta má forrita í einingu á eftirfarandi hátt — This can be programmed in a module as follows:

```
"reversion.mmod" =
{{
;;; Notkun: z = reverse(x)
;;; Fyrir:  x er listi [x1,...,xN]
;;; Eftir:  z er nýr listi [xN,...,x1]
;;; Usage:  z = reverse(x)
;;; Pre:    x is a list [x1,...,xN]
;;; Post:   z is a new list [xN,...,x1]
reverse =
  fun(x)
  {
    var z=x, y=[];
    while( z!=[] )
    {
      ;;; z=[xI,...,xN]
      ;;; y=[xI-1,...,x1]
      y = head(z) : y;
      z = tail(z);
    };
    return y;
  };
}};
```

Einnig má forrita þetta sem staðvært fall — We can also program this as a local function:

```
;;; Notkun: z = reverse(x)
;;; Fyrir:  x er listi [x1,...,xN]
;;; Eftir:  z er nýr listi [Xn,...,x1]
;;; Usage:  z = reverse(x)
;;; Pre:    x is a list [x1,...,xN]
;;; Eftir:  z is a new list [Xn,...,x1]
rec fun reverse(x)
{
  var z=x, y=[];
  while( z!=[] )
  {
    ;;; z=[xI,...,xN]
    ;;; y=[xI-1,...,x1]
    y = head(z) : y;
    z = tail(z);
  };
  return y;
};
```

Morphoþýðandann (`morpho.jar`) og handbókina (`Morpho.pdf`) má finna í Canvas.

The Morpho compiler (`morpho.jar`) and the manual (`Morpho.pdf`) can be found in Canvas.

2.1 Morpho keyrsluumhverfið — The Morpho Runtime Environment

Morpho er bálkmótað forritunarmál með lokunum og samhliða vinnslu. Þetta flækir dálítið málið þegar kemur að hönnun keyrsluumhverfisins. Mynd 1 sýnir almennt ástand í keyrslu Morpho sýndarvélarinnar.

Morpho is a block-structured programming language with closures and parallel programming. This complicates the design of the runtie environment a little. Figure 1 shows a general state while running the Morpho virtual machine.

Við ræðum um hana í fyrirlestri til að styrkja skilninginn á því hvernig innviðir forritunarmála virka.

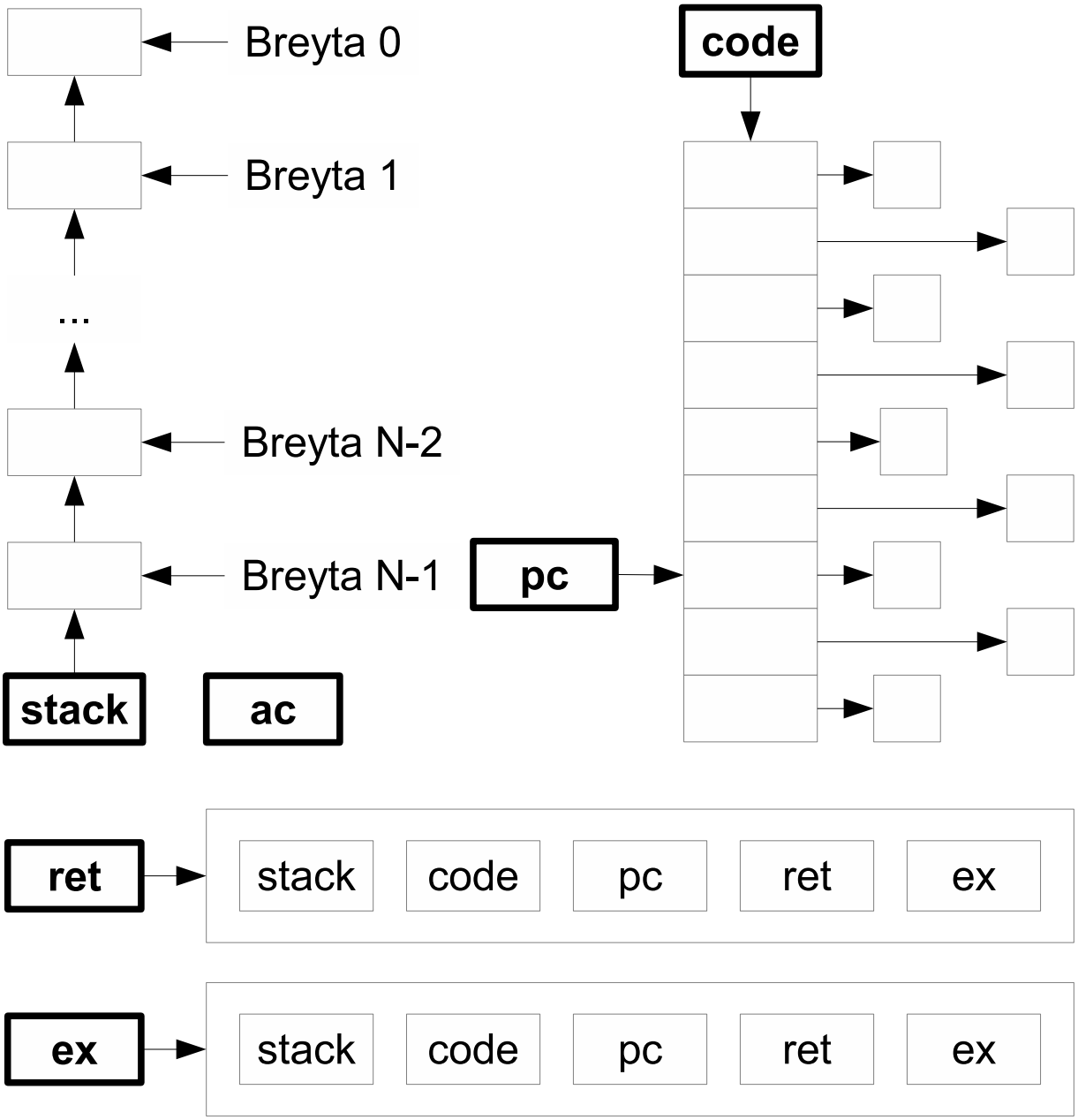
We will discuss this in lecture to strengthen the understanding of how programming language runtime environments work.

Í keyrslu Morpho sýndarvélarinnar eru nokkur gisti (*register*) í gangi — While running the Morpho virtual machine a few registers are maintained:

code. Fylki af Morpho vélarmálsskipunum sem verið er að framkvæma.

An array of Morpho machine language operations that are being executed.

pc. Vísir inn í **code** sem bendir á þá vélarmálsskipun sem verið er að framkvæma. Samanlagt skilgreina **code** og **pc** staðsetningu í vélarmálsþulu sem líta má að



Mynd 1: Staða í keyrslu Morpho — A State in the Morpho Runtime

sé sambærilegt við vendivistfang eða fallsbendi í forritunarmálum sem hafa einfaldara keyrsluumhverfi.

An index into **code** which points to the machine operation being executed. Together **code** and **pc** define a location in the machine code that is comparable to a return address or a function pointer in programming languages with a simpler runtime.

stack. Keðja af hlekkjum sem hver og einn inniheldur eina breytu. Þessi gildahlaði er umhverfið (*environment*) sem verið er að keyra í. Hér eru bæði staðværar breytur og breytur í efri földunarhæðum.

A chain of links each of which contains one variable. This stack of values is the environment in which the current function is executing. This chain contains both the local variables and the variables in enclosing scopes.

ret. Eðlilegt framhald úr núverandi vakningu (sjá neðar).

The normal continuation from the current activation (see below).

ex. Afbrigðilegt framhald úr núverandi vakningu (sjá neðar).

The abnormal continuation from the current activation (see below).

ac. Gisti sem fær nýtt gildi í hvert sinn sem Morpho segð skilar gildi (*accumulator*). Þegar fall skilar gildi er gildið sett í **ac** áður en snúið er til baka. úr fallinu.

A register that gets a new value each time a Morpho expression returns a value (*accumulator*). When a function returns a value, the value is put into **ac** before returning from the function.

Eitt lykilatriði í Morpho, og einnig í Scheme og mörgum svipuðum málum, er að þegar við köllum á fall þá sendum við fallinu bæði viðföng (á hlaðanum) og **framhald** (*continuation* á ensku). Framhaldið er í grundvallaratriðum bendir á vélarmálsþulu ásamt bendi á vakningarfærslu. En sama gildir um lokanir og hver er þá munurinn? Almennt er framhald gildi sem er dálítið keimlíkt og lokun en í stað þess að innihalda bendi á vakningarfærslu sem verður í næstu földunarhæð þegar lokunin er nytjuð (kallað er á lokunina), þá inniheldur framhald bendi á vakningarfærslu sem verður núverandi vakningarfærsla þegar framhaldið er nytjað (snúið er til baka úr núverandi falli). Halaendurkvæmni er þá útfærð með því að halaendurkvæmt kall fær sama framhald og sá sem kallar, í stað þess að búið sé til nýtt framhald til að snúa til baka til þess sem kallar. Lykilatriði er að framhaldið inniheldur stýrihlekk í stað þess að innihalda aðgangshlekk eins og lokun gerir.

One key issue in Morpho, and also in Scheme and many similar programming languages, is that when we call a function we give it both arguments (on the stack) and a *continuation*. The continuation is fundamentally a pointer to machine code along with a pointer to an activation record. But the same hold for closures and

what then is the difference? A continuation is a bit similar to a closure but instead of containing a pointer to an activation record that is in the enclosing nesting level when the closure is called, a continuation contains a pointer to an activation record that will become the current activation record on return from the function, i.e. when the continuation is used. Tail recursion is implemented by having a tail recursive call pass the current continuation to the function being called tail-recursively, instead of creating a new continuation to return to the function that calls. A key issue is that a continuation contains a control link (dynamic link), instead of containing an access link (static link), as a closure does.

Reyndar er ein afleiðing af þessari hönnun að aðgangshlekkir eru ekki lengur bendar á vakningarfærslur heldur eru bendar á hlekki í breytuhlaðanum. Þetta veldur minnissparnaði þegar breytur þurfa að lifa það af að fallið deyr sem bjó þær til, en þið þurfið ekki að muna það smáatriði og megið reikna með því að aðgangshlekkir séu bendar á vakningarfærslur. Í þessari hönnun getum við litið svo á að heil vakningarfærsla sé samanlagt innihaldið í gistunum **ret** og **ex**, ásamt byrjuninni á hlaðanum, sem gistið **stack** vísar á. Hlaðinn inniheldur öll viðföng fallsins og staðværar breytur fallsins ásamt öllum breytum í efri földunarhæðum. Þegar kallað er á fall verður til nýr hlaði, með því að skeyta hlekkjum við einhvern annan hlaða (aðgangshlekkinn), sem ræðst af földun þess falls sem verið er að kalla á.

One consequence of this design is that access links are not pointers to activation records but rather pointers to cells on the variable stack. This leads to memory savings when variables need to survive after the activation that created them dies, but you do not need to remember that detail and may simply assume that access links are pointers to activation records. With this design we may take the view that a whole activation record consists of the contents of the registers **ret** and **ex**, along with the first cells of the chain of variables that the register **stack** points to. The stack contains all the arguments to the function and the local variables of the function along with all variables in enclosing scopes. When a function is called a new stack is created by prepending cells to some other stack (the access link), which depends on the nesting of the function being called.

Á hverju andartaki í keyrslu Morpho eru tvö framhöld tiltæk. Annað framhaldið er í gistinu **ret** og er eðlilegt framhald sem nytjað er þegar núverandi fall skilar gildi á venjulegan hátt. Takið eftir að framhaldið inniheldur allt sem til þarf til að stilla sýndarvélina í skilgreint ástand nema gildið í gistinu **ac**. En þegar snúið er til baka úr kalli þarf fallið einmitt að tilgreina gildið sem fer í **ac** gistið, sem verður skilagildi fallsins. Hitt framhaldið er í gistinu **ex**, sem nytjað er þegar afbrigði gerist í keyrslu (þ.e. *exception*). Þá er gildi afbrigðisins (oft Java Exception eða Error gildi) sett í **ac** gistið og hægt að að grípa það í `catch` hluta af `try-catch` segð.

At each momemt while running Morpho there are two continuations available. One continuation is in the register **ret** and that is the normal continuation that is used when the current function returns a value in a normal fashion. Notice that the continuation contains everything needed to put the virtual machine into a well defined state except

the value in the register **ac**. But when returning from a function, the function needs to put the appropriate value into **ac**, which then becomes the return value from the function. The other continuation is in the register **ex**, which is used when an exception happens during the run. Then the value of the exception (often a Java Exception or Error value) is put into the **ac** register and this can then be caught in the `catch` part of a `try-catch` expression.

3 Notkun Morpho — Using Morpho

Keyra má Morpho í samtalsham (*interactive mode*) með skipuninni — Morpho can be run in interactive mode using the command

```
java -jar morpho.jar
```

Til að þýða Morpho forrit úr textaskrá `x.morpho` má nota eftirfarandi skipun — To compile a Morpho program in a text file `x.morpho` we can use the following command:

```
java -jar morpho.jar -c x.morpho
```

Út úr slíkri þýðingu gæti, til dæmis, komið út keyrsluhæf skrá `x.mexe`, sem þá má keyra með skipuninni — The result of such a compilation could, for example, be an executable file `x.mexe`, which can the be run using the command

```
java -jar morpho.jar x
```

Nánari uppýsingar er að finna í handbókinni fyrir Morpho.

More detailed information can be found in the manual for Morpho.