TÖL304G
Forritunarmál
Lausnir 2

Snorri Agnarsson

2. september 2024

Hópverkefni

1. Skrifið Scheme fall `last` sem tekur lista sem viðfang, sem ekki má vera tómur, og skilar aftasta gildinu í listanum. Til dæmis skal segðin `(last '(1 2 3))` skila `3`.

   Svar:

```
;; Notkun: (last x)
;; Fyrir:  x er listi, ekki tómur.
;; Gildi:  Aftasta gildið í listanum.
(define (last x)
  (if (null? (cdr x))
      (car x)
      (last (cdr x))
  )
)
```

2. Skrifið Scheme fall `remove-last` sem tekur lista sem viðfang, sem ekki má vera tómur, og skilar lista allra gilda nema aftasta í viðfangslistanum. Til dæmis skal segðin `(remove-last '(1 2 3))` skila `(1 2)`.

   Svar:

```
;; Notkun: (remove-last x)
;; Fyrir:  x er listi, ekki tómur.
```

```
;; Gildi:  Listi sem inniheldur öll gildin úr x nema
;;         síðasta, í sömu röð og í x.
(define (remove-last x)
  (if (null? (cdr x))
      '()
      (cons (car x) (remove-last (cdr x)))
  )
)
```

3. Skrifið Scheme fall `product`, sem tekur eitt viðfang x, sem skal vera listi talna $x_1, \ldots, x_n$, og skilar $\Pi_{i=1}^{n} x_i$. Þið skuluð leyfa að listinn sé tómur og skila viðeigandi gildi í því tilviki. Fallið skal vera halaendurkvæmt.[1]

   Svar:

```
;; Notkun: (product x)
;; Fyrir:  x er listi talna.
;; Gildi:  Margfeldi talnanna í x.
(define (product x)
  ;; Notkun: (help x p)
  ;; Fyrir:  x er listi talna, p er tala
  ;; Gildi:  Margfeldi p og talnanna í x.
  (define (help x p)
    (if (null? x)
        p
        (help (cdr x) (* p (car x)))
    )
  )
  (help x 1)
)
```

4. Skrifið Scheme fall `myappend` sem tekur tvo lista, `x` og `y` sem viðföng, og skilar lista sem inniheldur fremst öll gildin úr `x` (í sömu röð og í `x`) og síðan öll gildin úr `y` (í sömu röð og í `y`). Fallið `myappend` skal útfæra með því að nota föllin `last` og `remove-last`, að ofan, og einnig má nota `define`, `if`, `cons`, `null?`, en ekki önnur föll eða lykilorð. Fallið `myappend` verður eðlilega halaendurkvæmt, þannig að oftast þegar kallað er á það mun það enda á að kalla á sjálft sig. Tímaflækja þessa falls er hins vegar ekkert til að hrópa húrra fyrir. Athugið líka að þótt `myappend` sé halaendurkvæmt þá er næstum öruggt

[1] Það dugar að útreikningarnir séu framkvæmdir af halaendurkvæmu hjálparfalli, jafnvel þótt `product` sé ekki beint halaendurkvæmt. Markmiðið er að takmarka dýpt hlaðans sem forritið notar fyrir milliniðurstöður, sem við munum sjá að er hlaði svokallaðra vakningarfærslna (*activation records*).

að hjálparfallið `remove-last` verður trúlega ekki halaendurkvæmt þannig að heildarlausnin er þá ekki halaendurkvæm.

Svar:

```
;; Notkun: (myappend x y)
;; Fyrir:  x og y eru listar.
;; Gildi:  Listi sem inniheldur fremst öll gildin
;;         úr x og síðan öll gildin úr y, í sömu
;;         röð og í upphaflegu listunum.
(define (myappend x y)
  (if (null? x)
      y
      (myappend (remove-last x) (cons (last x) y))
  )
)
```

Einstaklingsverkefni

Klárið Dafny föllin þrjú sem eru ókláruð á þessari vefsíðu[2]. Skilið PDF útprenti af lausninni í Gradescope og skilið einnig (fremst í sama útprenti) permalink á lausnina ykkar. Þið getið fengið permalink á lausnina, þegar hún er tilbúin, með því að styðja á hnappinn sem merktur er með keðju, sækja of langa permalinkinn þar og nota tinyurl.com[3] til að smíða styttri, nothæfan, permalink.

Til hliðsjónar getið þið kíkt á þessa vefsíðu[4], sem við munum kíkja á í fyrirlestri.
Svar: Sjá hér[5] á vefnum. Forritstextinn fyrir lausnina er til dæmis eftirfarandi:

```
// For k>=0 this function returns 1+2+3+...+k.
// This is the sum of the first k integers >0.
// If k==0 then this sum is 0.
// In older versions of Dafny a function like this
// is not executable but can take part in program
// verification and the Dafny compiler "understands"
// the body of the function.
function SumInts( k: int ): int
  requires k >= 0
{
  if k == 0 then 0 else SumInts(k-1)+k
```

[2] https://tinyurl.com/ydusfckf
[3] https://tinyurl.com
[4] https://tinyurl.com/y23c9ku3
[5] https://tinyurl.com/42px7zxf

```
}

// Compute SumInts using a loop and prove
// that SumInts(k) == (k+1)*k/2.
method SumIntsLoop( k: int ) returns( s: int )
  requires k >= 0
  ensures s == (k+1)*k/2
  ensures s == SumInts(k)
{
  // Finish programming the body and do not
  // use recursion
  var i := 0;
  s := 0;
  while i != k
    invariant 0 <= i <= k
    decreases k-i
    invariant s == (i+1)*i/2
    invariant s == SumInts(i)
  {
    i := i+1;
    s := s+i;
  }
}

// Compute SumInts using recursion and prove
// that SumInts(k) == (k+1)*k/2.
method SumIntsRecursive( k: int ) returns( s: int )
  requires k >= 0
  ensures s == (k+1)*k/2
  ensures s == SumInts(k)
{
  // Finish programming the body and use recursion
  // and no loop.
  // Only call SumIntsRecursion.

  if( k == 0 ) { return 0; }
  s := SumIntsRecursive(k-1);
  s := s+k;
}

// Compute SumInts using tail recursion and prove
// that SumInts(k) == (k+1)*k.
```

```
method SumIntsTailRecursive( i: int, r: int, k: int ) returns(
  requires 0 <= i <= k
  decreases k-i
  requires r == (i+1)*i/2
  requires r == SumInts(i)
  ensures s == (k+1)*k/2
  ensures s == SumInts(k)
{
  // Finish programming the body and use tail recursion
  // and no loop.
  // Only call SumIntsTailRecursion.

  if( i == k ) { return r; }
  s := SumIntsTailRecursive(i+1,r+i+1,k);
}

method Main()
{
  var s1 := SumIntsLoop(5);
  var s2 := SumIntsRecursive(5);
  var s3 := SumIntsTailRecursive(0,0,5);
  var s4 := SumIntsTailRecursive(4,10,5);
  print s1;
  print s2;
  print s3;
  print s4;
}
```