Freie Universität Berlin

Prof. Dr. Claudia Müller-Birn, Barry Linnert

Objektorientierte Programmierung, SoSe 17

Übung 06

TutorIn: Thierry Meurers
Tutorium 10

Stefaan Hessmann, Jaap Pedersen, Mark Niehues

7. Juni 2017

---

1 Aufgabe 1

1.1 Beweise der Türme

Wir haben den Beweis etwas abgekürzt, indem wir den Code auf die hanoi Methode beschränkt haben (siehe 1.2) und analoge Elemente in den If-Bedingungen nicht behandelt haben. Trotzdem ist der Beweis sehr unübersichtlich, sorry besser gings nicht!

```
1  A ist sortiert ⇔ {∀0 < i < len(A) − 1 : A[i] < A[i + 1] ∨ len(A) < 2}
3  INV = {S[0], S[1], S[2] sind sortiert ∧ len(S[0]) + len(S[1]) + len(S[2]) = n ∧ kleinstes Element ist S[0][−1]}
5  Q = {len(H) = len(A) = 0 ∧ len(Z) = n ∧ Z sortiert}
7  def hanoi(n):
10   assert (n > 0)
11   A = list(range(n,0,-1))
12   H = []
13   Z = []
15 P = {n > 0 ∧ len(H) = len(Z) = 0 ∧ len(A) = n ∧ H, Z, A sortiert}
17   if (n % 2 == 0):
19     {P ∧ B} = {P ∧ n%2 = 0}
20     S = [A,H,Z]
21     {P ∧ B} = {P ∧ n%2 = 0 ∧ S = [A, H, Z]}
23  ⇒ INV mit Kongruenzregel, da :
24  len(H) = len(Z) = 0 ∧ len(A) = n ⇒ ∧len(S[0]) + len(S[1]) + len(S[2]) = n ∧ S = [A, H, Z]
25  H, Z, A sortiert ∧ S = [A, H, Z] ⇒ S[0], S[1], S[2] sind sortiert
26  Kleinstes Element klar weil alle Scheiben in A und A ist sortiert
28  while A != [] or H != []:
29   {INV ∧ B_while} = {INV ∧ (len(S[0]) > 0 ∨ len(S[1]) > 0)}
31   # verschiebt kleinste Scheibe um einen Platz
```

```
  S[1].append(S[0].pop())

  {S[0], S[1], S[2] sind sortiert ∧ len(S[0])−1 + len(S[1])+1 + len(S[2]) = n∧
  kleinstes Element ist S[1][−1]∧
  (len(S[0]) > −1 ∨ len(S[1]) > 1)} ≡ Q*

  # gibt es andere verschiebbare Scheiben? wenn ja, verschiebe
  if S[0] != [] and (S[2] == [] or S[0][-1] < S[2][-1]):
    {Q* ∧ B} = {Q* ∧ len(S[0]) > 0 ∧ (len(S[2]) = 0 ∨ S[0][−1] < S[2][−1])}
    ⇔ {S[0], S[1], S[2] sind sortiert ∧ len(S[0]) − 1 + len(S[1]) + 1 + len(S[2]) = n∧
    kleinstes Element ist S[1][−1] ∧ len(S[0]) > 0 ∧ (len(S[2]) = 0 ∨ S[0][−1] < S[2][−1])}

    S[2].append(S[0].pop())
    {S[0], S[1], S[2] sind sortiert ∧ len(S[0])−2 + len(S[1]) + 1 + len(S[2]+1) = n∧
    kleinstes Element ist S[1][−1] ∧ len(S[0]) > −1∧
    (len(S[2]) = 0 ∨ S[0][−1] < S[2][−1])}

    S = [S[1],S[2],S[0]]
    {S[0], S[1], S[2] sind sortiert ∧ len(S[2]) − 2 + len(S[0]) + 1 + len(S[0]) + 1) = n∧
    kleinstes Element ist S[0][−1] ∧ len(S[2]) > −1∧
    (len(S[1]) = 0 ∨ S[2][−1] < S[1][−1])}
          ⇒ INV

  elif S[2] != [] and (S[0] == [] or S[2][-1] < S[0][-1]):
          # Analog zum oberen Beispiel mit anderen Indizes für S
    ...

 {INV + ¬B_while}
 = {S[0], S[1], S[2] sind sortiert ∧ len(S[0]) + len(S[1]) + len(S[2]) = n∧
 kleinstes Element ist S[0][−1] ∧ len(A) = 0 ∧ len(H) = 0}
 ⇔ {S[0], S[1], S[2] sind sortiert ∧ len(S[0]) + len(S[1]) + len(S[2]) = n∧
 kleinstes Element ist S[0][−1] ∧ len(S[0]) = n ∧ S[0] = Z}

 ⇒ Q = {len(H) = len(A) = 0 ∧ len(Z) = n ∧ Z sortiert}

  else:

    # Analog zum oberen Teil, allerdings ist die Zuweisung von Z und H anders,
    # weshalb die Indizes von S sich unterscheiden.
    ...
```

1.2 Angepasstes Code Snippet

Listing 1: Angepasster Code Schnipsel, der bewiesen werden muss.

```
# ////////////// die Hanoi Loesung //////////////
def hanoi(n):


  assert (n > 0)
  A = list(range(n,0,-1))
  H = []
  Z = []

  if (n % 2 == 0):

 S = [A,H,Z]
 while A != [] or H != []:

  # verschiebt kleinste Scheibe um einen Platz
  S[1].append(S[0].pop())

  # gibt es andere verschiebbare Scheiben? wenn ja, verschiebe
  if S[0] != [] and (S[2] == [] or S[0][-1] < S[2][-1]):
    S[2].append(S[0].pop())
```

```
22      S = [S[1],S[2],S[0]]

24    elif S[2] != [] and (S[0] == [] or S[2][-1] < S[0][-1]):
25      S[0].append(S[2].pop())

27      S = [S[1],S[2],S[0]]

29    else:
30     S = [S[1],S[2],S[0]]

32    else:

34  S = [A,Z,H]

36  while A != [] or H != []:
37    # verschiebt kleinste Scheibe um einen Platz
38    S[1].append(S[0].pop())

40    # gibt es andere verschiebbare Scheiben? wenn ja, verschiebe
41    if S[0] != [] and (S[2] == [] or S[0][-1] < S[2][-1]):
42      S[2].append(S[0].pop())

44      S = [S[1],S[2],S[0]]

46    elif S[2] != [] and (S[0] == [] or S[2][-1] < S[0][-1]):

48      S[0].append(S[2].pop())

50      S = [S[1],S[2],S[0]]

52    else:
53     S = [S[1],S[2],S[0]]
```

1.3 Ursprünglicher Code

Listing 2: Ursprünglicher Code aus dem KVV

```
1  import math

3  # ////////////// Print-Funktion //////////////
4  def printStapel(A,H,Z):
5    """ zeigt den Status der Tuerme an, hat nur Lesezugriffe auf die Stapel """
6    # Anzahl der Scheiben pro Stab
7    nA = len(A)
8    nH = len(H)
9    nZ = len(Z)
10   N = nA + nH + nZ      # Gesamtzahl der Scheiben
11   # fuellt Staebe auf gleiche Hoehe auf
12   Anew = [chr(9474) for i in range(N-nA)] + [ str(x) for x in (A[::-1])]
13   Hnew = [chr(9474) for i in range(N-nH)] + [ str(x) for x in (H[::-1])]
14   Znew = [chr(9474) for i in range(N-nZ)] + [ str(x) for x in (Z[::-1])]
15   # ermittelt groesste Ziffernzahl
16   breite = int(math.log10(N))+1
17   string = 3*(' ' * breite + chr(9474)) + '\n'
18   for i in range(N):
19     for x in [Anew,Hnew,Znew]:
20       string += ' '*(breite - len(x[i]) +1) + x[i]
21     string += '\n'
22   string += 3*(chr(9473) * breite + chr(9527)) + '\n'
23   string += (' ' * breite + 'A') + (' ' * breite + 'H') + (' ' * breite + 'Z') + '\n'
24   print(string)


27 # ////////////// die Hanoi Loesung //////////////
```

```
28 def hanoi(n):
29   global step
30
31   assert (n > 0)
32   A = list(range(n,0,-1))
33   H = []
34   Z = []
35   if (n % 2 == 0):
36     stapel = [A,H,Z]
37   else:
38     stapel = [A,Z,H]
39   printStapel(A,H,Z)                                    # nur Lesezugriffe
40
41   while A != [] or H != []:
42     # verschiebt kleinste Scheibe um einen Platz
43     x = stapel[0].pop()
44     stapel[1].append(x)
45     step += 1
46     printStapel(A,H,Z)                                  # nur Lesezugriffe
47     # gibt es andere verschiebbare Scheiben? wenn ja, verschiebe
48     if stapel[0] != [] and (stapel[2] == [] or stapel[0][-1] < stapel[2][-1]):
49       stapel[2].append(stapel[0].pop())
50       step += 1
51     elif stapel[2] != [] and (stapel[0] == [] or stapel[2][-1] < stapel[0][-1]):
52       stapel[0].append(stapel[2].pop())
53       step += 1
54     printStapel(A,H,Z)                                  # nur Lesezugriffe
55     stapel = [stapel[1],stapel[2],stapel[0]]
56
57
58
59 # ////////////// externer Aufruf //////////////
60 step = 0                                               # Anzahl der Zuege
61
62 try:
63   n = int(input('\nGeben Sie eine Scheibenzahl n>0 ein: '))
64   hanoi(n)
65 except ValueError:
66   print('Es muss eine natuerliche Zahlsein')
67
68 print ("Anzahl der Zuege:", step)
```