2. Aufgabenblatt zur Vorlesung

Nichtsequentielle und Verteilte Programmierung SoSe 2018

Tutorium No. 04 bei Johannes Nixdorf

Anton Oehler, Mark Niehues

Aufgabe 1 Nebenläufiges Zählen

(a) Durch das cachen der Variablen n, kann diese beliebige Werte zwischen *K* und *-K* am Ende des Programmablaufes annehmen. Bei Annahme schwacher Fairness ist davon auszugehen, dass einer der Threads zu einem Zeitpunkt, da `n` nicht mehr null ist, seiner eigenen Cache Variablen diesen Wert zuweist. Was zwischen dem Cachen und dem zurückschreiben im nächsten atomaren Schritt eines Threads passiert, hat keine Auswirkungen auf den End-Zustand des Programms. Dementsprechend hängt der End-Zustand nur davon ab, an welcher stelle die Variable `n` gecached bzw. in diesem Fall könnte man auch sagen *eingefroren* wird.

(b)

```
1  import java.util.Random;
2
3  class Test1 extends Thread{
4    int K;
5    int start;
6    public Test1(int K, int start){
7     this.K = K;
8     this.start = start;
9    }
10
11         public void run(){
12          int a;
13           try{
14            for(int i = 1; i <= K; i++){
15             a = Aufgabe1.n;
16             Aufgabe1.n = a + 1;
17
18             if(Aufgabe1.n == start){ // Defines the starting
     state for Test2
19               this.sleep(100); // Wait so the other Thread is
     calle and caches n
20             }
21
22             System.out.println("---> Addition: n = " + Aufgabe1
     .n);
23            }
24           } catch(Exception e) {
25            System.out.println(e);
26           }
27          }
28      }
```

```
30 class Test2 extends Thread{
31   int K;
32   public Test2(int K){
33    this.K = K;
34   }
36         public void run(){
37           int a;
38          try{
39      this.sleep(50); //  Wait for initial loops of Test 1
41            for(int i = 1; i <= K; i++){
42             a = Aufgabe1.n;
43             this.sleep(100); // Wait so Test 1 continues and
    finishes
44              Aufgabe1.n = a - 1;
45              System.out.println("---> Subtraktion: n = " +
    Aufgabe1.n);
46             }
47           } catch(Exception e) {
48            System.out.println(e);
49           }
50          }
51      }
53 public class Aufgabe1 {
55  public static int n = 0;
57  public static void main(String[] args) {
58   int K = 10;
60   for( int i = 0; i<= K; i++){
61    // Produce a few different final states depending on the
    iteration Thread 1 caches n
62    n = 0; // initial state
64    Test1 Thread1 = new Test1(K, i);
65    Test2 Thread2 = new Test2(K);
67    Thread1.start();
68    Thread2.start();
70    try{ // Wait for Threads to finish
71     Thread1.join();
72     Thread2.join();
73    } catch (InterruptedException e){
74     System.out.println(e);
75    }
77    System.out.println("Endzustand: n = " + n);
78   }
80  }
```

```
81 }
```

Aufgabe 2 Parallele Suche

Es lässt sich zeigen, dass Programm a und b aus den selben Gründen nicht korrekt sind, während Programm c korrekt ist. Die Ursache für das Fehlerhafte verhalten der ersten beiden Programme bildet der Umstand, dass die `found` Variable auch mit `false` überschrieben wird, falls ein Thread keine Nullstelle gefunden hat. Dadurch kann das Finden der Nullstelle bei ungünstigem Schedulen untergehen. Mit diesem Fall ist auf Grund der angenommenen schwachen Fairness zu rechnen. Ein Szenario zu Programm a wäre besipielsweise (das Szenario zu b ist analog):

Listing 1: Beispielhaftes Szenario, dass die Korrektheit von a widerlegt.

```
1  Sei n Nullstelle, sodass f(n) = 0
2  Zustand: (i, j, found)
3
4  ...  (n-1, n-1, false)
5  a4 -> (n, n-1, false)
6  a5 -> (n, n-1, true)
7  b4 ->  (n, n-1, false) # Kritischer Übergang, der zum fehlerhaften
       Verhalten führt
8  a2 ->  (n, n-1, false)
9  a3 -> (n+1, n-1, false) # Nullstelle übersprungen
10 ...
```

In dem letzten Programm kann die Variable, die anzeigt, dass eine Nullstelle gefunden wurde nicht mehr mit `false` überschrieben werden sobald sie einmal den Wert `true` zugewiesen bekommen hat. Dadurch terminieren beide Threads zwangsläufig, sobald einer der beiden die Nullstelle gefunden hat.

Aufgabe 3 Das Froschpuzzle

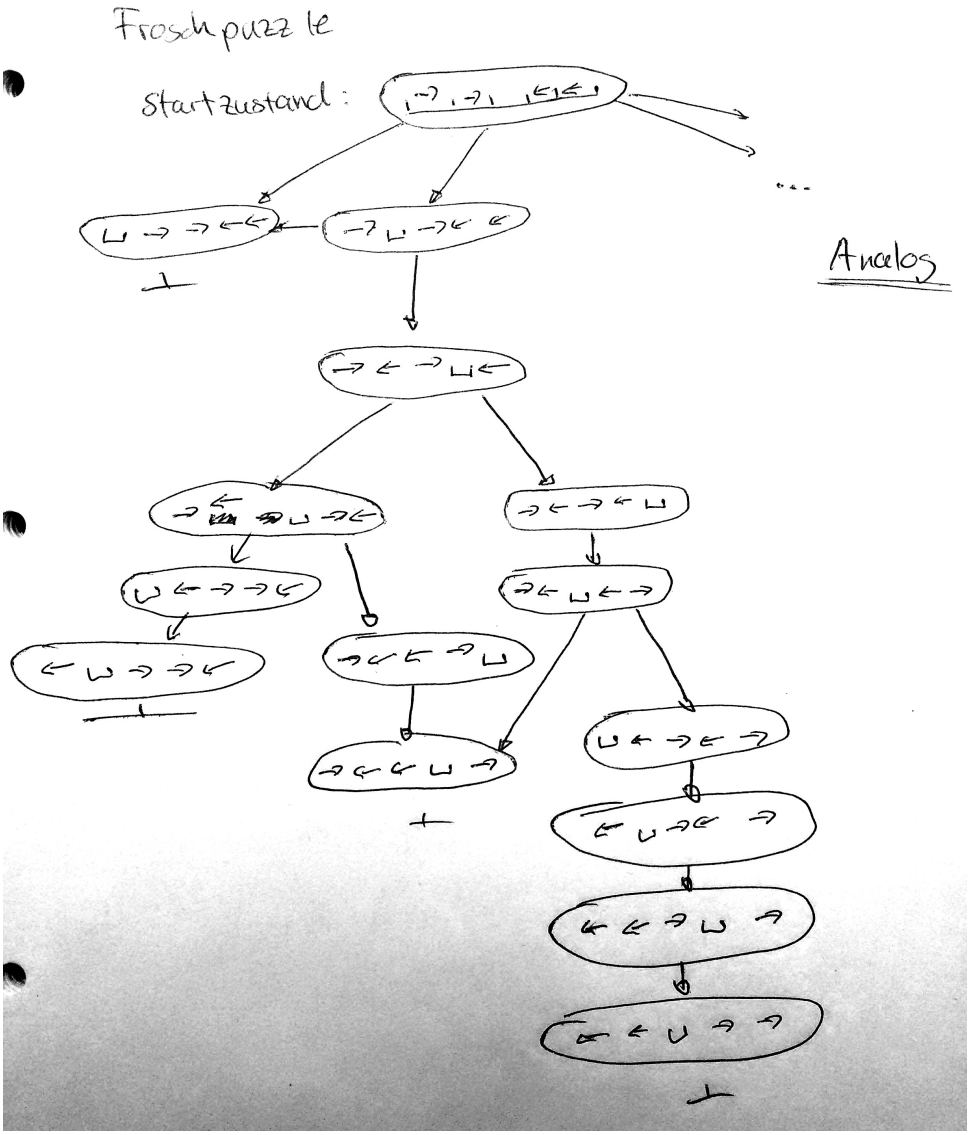

Abbildung 1: Zustandsdiagramm des Froschpuzzles