Algorithmen und Programmierung 2, SS 2016 — 4. Übungsblatt

Abgabe bis Freitag, 13. Mai 2016, 12:00 Uhr

20. Elimination von `continue`, 8 Punkte

    Schreiben Sie zu den beiden folgenden Beispielen äquivalente Programme ohne die Anweisungen `break` und `continue` und ohne die `else`-Klausel bei der `while`-Schleife.

```
while B:
    S
    if C:
        T
        if D:
            U
        else:
            continue
        V
    W
```

```
while B:
    S
    if C:
        T
    else:
        U
        break
    V
else:
    W
```

    Dabei sind B, C, D Bedingungen, und $S, T, \ldots$ steht für Anweisungen oder Folgen von Anweisungen. (Die `else`-Klausel der `while`-Schleife wird ausgeführt, wenn die Schleife wegen der Bedingung B verlassen wird und nicht durch die `break`-Anweisung.)

21. Ägyptische Multiplikation, 14 Punkte

    (a) Beweisen Sie die partielle Korrektheit der ägyptischen Multiplikation (Aufgabe 17 vom 3. Blatt) mit dem Hoare-Kalkül unter der Annahme $a_0, b_0 \geq 0$. (Sie dürfen dabei das Programm leicht umschreiben, ohne die Bedeutung zu ändern; zum Beispiel können Sie bei der Anweisung `a=a//2` eine Fallunterscheidung machen, ob a gerade oder ungerade ist.) Geben Sie jedesmal an, welche Regeln des Hoare-Kalküls Sie verwenden.

    (b) Beweisen Sie die totale Korrektheit.

22. Suche, 8 Punkte, Programmieraufgabe

    (a) Schreiben Sie eine einfache PYTHON-Funktion `finde` nach der folgenden durch Vor- und Nachbedingungen ausgedrückten Spezifikation. Verwenden Sie nicht einfach die eingebaute Methode (Funktion) `index` oder die Abfrage „n `in` A".

    $\{\, \mathit{type}(A) = \mathit{list} \,\}$
    `i = finde(A,n)`
    $\{\, (\mathit{type}(i) = \mathit{int} \wedge A[i] = n) \;\vee\; (i = \texttt{None} \wedge \forall j\colon 0 \leq j < \texttt{len}(A) \Rightarrow A[j] \neq n) \,\}$

    (b) Schreiben Sie die Funktion so um, dass sie nur mit `while`-Schleifen auskommt. (Eventuell müssen Sie in einer späteren Aufgabe die Korrektheit des Programms beweisen.)

23. Tribonacci-Zahlen, 0 Punkte

    Erweitern Sie die Funktion `Tr(n)` aus der Vorlesung vom 25. April[1] so, dass sie die Tribonacci-Zahlen T_n auch für auch für negative Parameter n berechnet (und nicht bloß 0 ausgibt). Das Ergebnis soll so definiert sein, dass die Gleichung $T_{n+1} = T_n + T_{n-1} + T_{n-2}$ für *alle* ganzen Zahlen n gilt.

[1] `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/tribonacci.py`