## 10. Übungsblatt zu Logik, Semantik und Verifikation SS 2001

Prof. Dr. Gert Smolka, Dr. Christian Schulte
www.ps.uni-sb.de/courses/prog-lsv01/

Abgabe: 18. Juni in der Vorlesungspause

**Aufgabe 10.1: Verifikation (8)** Sei $I \in \mathit{Assn}$. Betrachten Sie die folgende Korrektheitsaussage:

$\{ X \geq 0 \}$
```
Y:=0; Z:=0;
```
$\{ I \}$
```
while Z≤X-1 do
   Y:=Y+Z+Z+1; Z:=Z+1;
```
$\{ Y = X * X \}$

(a) Geben Sie die Verifikationsbedingungen an (in möglichst expliziter Form).

(b) Geben Sie eine Invariante $I$ an, für die alle Verifikationsbedingungen gelten.

**Aufgabe 10.2: Korrektheit der Hoare-Regeln (10)** Ergänzen Sie den Beweis von Satz 8.2.1 im Skript um die Teile für Zuweisung, Konditional und Schleifen.

**Aufgabe 10.3: Korrektheit von $VC'$ (8)** Ergänzen Sie den Beweis von Satz 8.3.1 im Skript um die Teile für Zuweisung und Sequentialisierung.

**Aufgabe 10.4: Monotonie von $VC'$ (8)** Ergänzen Sie den Beweis von Satz 8.5.1 im Skript um die Teile für Zuweisung und Sequentialisierung.

**Aufgabe 10.5: Vollständigkeit von $VC'$ (8)** Ergänzen Sie den Beweis von Satz 8.5.2 im Skript um die Teile für Zuweisung und Konditional.

**Aufgabe 10.6: Eigenschaften von SVB-Funktionen (8)** Ergänzen Sie den Beweis von Proposition 8.6.3 im Skript um den Teil für Behauptung (4).

**Aufgabe 10.7: Berechnen von Verifikationsbedingungen (Freiwillig)** Im folgenden wollen wir den Generator für Verifikationsbedingungen implementieren. Dabei sind die folgenden Datentypen

```
type con = int
type loc = string
datatype aexp = Con  of con
              | Loc  of loc
              | Add of aexp * aexp
              | Mul  of aexp * aexp
datatype bexp = True
              | False
              | LE  of aexp * aexp
              | Not of bexp
              | And of bexp * bexp
type     assn = bexp
datatype com  = Skip
              | Assign of loc * aexp
              | Seq    of com * com
              | If     of bexp * com * com
              | While  of assn * bexp * com
```

vorgegeben. Im Vergleich zu Aufgabe 8.2 sind neu: `skip`, Zusicherungen (`assn`), Invarianten für Schleifen, Multiplikation und erweiterte Bedingungen (damit man auch schöne Programme und Invarianten hinschreiben kann).

(a) Unsere Zusicherungen sind sehr primitiv. Programmieren Sie Prozeduren

```
val dis  : bexp -> bexp -> bexp
val imp  : bexp -> bexp -> bexp
val cond : bexp -> bexp -> bexp -> bexp
```

die Disjunktion, Implikation und Konditional mit Hilfe von Negation und Konjunktion repräsentieren. Dabei ist das Konditional $\text{cond}(b, b_1, b_2)$ definiert als $(b \wedge b_1) \vee (\neg b \wedge b_2)$.

(b) Programmieren Sie Substitution für arithmetische und Boolesche Ausdrücke:

```
val substa : aexp -> aexp -> loc -> aexp
val substb : bexp -> aexp -> loc -> bexp
```

Zum Beispiel: $(\texttt{substa}\ a\ a'\ x) = a[a'/x]$. Glück gehabt: Wenig Arbeit, da es keine Konstruktoren für Disjunktion und Implikation gibt.

(c) Programmieren Sie eine Prozedur

```
val vc : assn -> com -> assn -> assn list
```

die zu einer Vorbedingung, einem Kommando, und einer Nachbedingung die Liste der Verifikationsbedingungen berechnet.