8. Übungsblatt zu Logik, Semantik und Verifikation SS 2001

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

Abgabe: Donnerstag, 7. Juni in der Vorlesungspause

Aufgabe 8.1: Unstetige Funktionen (5) Geben Sie eine Funktion $F \in (\mathbb{N} \to \mathbb{N}_\perp) \to (\mathbb{N} \to \mathbb{N}_\perp)$ an, die monoton aber nicht stetig ist.

Aufgabe 8.2: Implementierung der denotationalen Semantik für IMP (20) Sie sollen einen Interpretierer in SML schreiben, der die denotationale Semantik von IMP implementiert. Dabei sind die folgenden Deklarationen gegeben:

```
type con = int
type loc = string

datatype aexp = Con of con | Loc of loc | Add of aexp * aexp

datatype bexp = LE of aexp * aexp

datatype com = Assign of loc * aexp
             | Seq    of com * com
             | If     of bexp * com * com
             | While  of bexp * com

type sigma = loc -> int

fun fix f x = f (fix f) x
```

Gehen Sie wie folgt vor:

(a) Implementieren Sie die denotationale Semantik für arithmetische Ausdrücke:

```
val da : aexp -> sigma -> int
```

(b) Implementieren Sie die denotationale Semantik für Boolesche Ausdrücke:

```
val db : bexp -> sigma -> bool
```

(c) Implementieren Sie die Hilfsfunktion für Schleifen:

```
val gamma : (sigma -> bool) * (sigma -> sigma) -> (sigma -> sigma)
            -> (sigma -> sigma)
```

(d) Implementieren Sie die denotationale Semantik für Kommandos:

```
val dc : com -> sigma -> sigma
```

Aufgabe 8.3: Programmieren in IMP (5) Sei die Prozedur

```
fun eval c n = dc c (fn l => if l="N" then n else 0) "Y"
val eval : com -> int -> int
```

gegeben, die das Kommando `c` evaluiert. Dabei erfolgt die Eingabe über die Lokation `"N"` und die Ausgabe über die Lokation `"Y"`. Geben Sie ein Kommando `c` an, so dass

```
eval c
```

zu einer Prozedur evaluiert, die die Fakultätsfunktion berechnet. Denken Sie daran, dass IMP keine Multiplikation besitzt!

Aufgabe 8.4: IMP' (20) Wir betrachten eine Variante IMP' von IMP, in der es zusätzlich until-Schleifen (`do c until b`) und Negation für Boolesche Ausdrücke ($\neg$`b`) gibt. Die Ausführung eines until-Kommandos

```
do c until b
```

führt `c` mindestens einmal aus, und die Ausführung von `c` wird solange wiederholt, bis `b` gilt. Tipp: Die until-Schleife entspricht `do c while (¬b)` in C.

(a) Geben Sie ein zu `do c until b` äquivalentes IMP'-Kommando ohne until-Schleifen an.

(b) Geben Sie ein zu `while b do c` äquivalentes IMP'-Kommando ohne while-Schleifen an.

(c) Geben Sie für die operationale Semantik von IMP' die Inferenzregeln für die Negation von Booleschen Ausdrücken an.

(d) Geben Sie für die denotationale Semantik von IMP' die Gleichung für die Negation von Booleschen Ausdrücken an.

(e) Geben Sie für die operationale Semantik von IMP' die Inferenzregeln für until-Schleifen an.

(f) Geben Sie für die denotationale Semantik von IMP' die Gleichung für until-Schleifen zusammen mit einer passenden Hilfsfunktion Γ' an. Tipp: Probieren Sie die denotationale Semantik mit einer Implementierung in SML aus!

(g) Durch Regelinduktion kann man für IMP' zeigen, dass

$$\forall c \in Con' \, \forall \sigma \in \Sigma \, \forall \sigma' \in \Sigma : \; \sigma \vdash c \Rightarrow \sigma' \quad \Rightarrow \quad \mathcal{C}[\![c]\!]\sigma = \sigma'$$

gilt. Dieser Beweis ist bis auf die Regeln für until-Schleifen mit dem Beweis für IMP identisch. Geben Sie die Beweisteile für die neuen Regeln an.

(h) Durch strukturelle Induktion über c kann man für IMP' zeigen, dass

$$\forall c \in Con' \, \forall \sigma \in \Sigma \, \forall \sigma' \in \Sigma : \; \mathcal{C}[\![c]\!]\sigma = \sigma' \quad \Rightarrow \quad \sigma \vdash c \Rightarrow \sigma'$$

gilt. Geben Sie den Beweisteil für until-Schleifen an.