

Übungen zur Vorlesung
„Grundlagen der Programm- und Systementwicklung“

Aufgabe 6.1 Eigenschaften von Programmiersprachen (für die Zentralübungen)

Sollen semantische Eigenschaften von Programmiersprachen bewiesen werden, stellt sich das Problem der Einbettung der Semantik in einen Theorembeweiser. Seit Michael C.J. Gordon 1988 eine einfache while-Sprache (s.u.) behandelt hat und die Hoare-Regeln als Theoreme abgeleitet hat, ist die Technik inzwischen etabliert. Wir lernen sie an einem einfachen Beispiel kennen, an der Sprache L der *positiven booleschen Ausdrücke*, die aus Identifikatoren ID, Konjunktionen und Disjunktionen bestehen. Unser Ziel ist es, einen Monotoniesatz zu beweisen.

- Definieren Sie die Syntax von $L \subseteq (ID \cup \{\bullet, +, (\cdot)\})^*$ durch BNF-Regeln. ID sei gegeben.
- Definieren Sie die abstrakte Syntax von L durch einen Datentyp PBEX. ID sei gegeben.
- Definieren Sie induktiv über diesem Datentyp die Semantik von L.

Wenn der Theorembeweiser Ihrer Wahl einen Beweiskalkül für algebraische Spezifikationen besitzt, dann haben Sie die Beispielsprache jetzt eingebettet.

Auf der Sorte Bool sei die Relation \leq eingeführt durch

$$a \leq b = (\neg a \vee b)$$

Zur Übung: Beweisen Sie

$$(a \leq b) \Rightarrow (\neg b) \leq (\neg a)$$

$$(a \leq p) \wedge (b \leq q) \Rightarrow (a \circ b) \leq (p \circ q) \text{ für } \circ \in \{\wedge, \vee\}.$$

Belegungen sind Abbildungen, die jedem Identifikator einen booleschen Wert zuordnen. Auf der Sorte Env der Belegungen wird die Relation \leq punktweise definiert durch

$$\beta_1 \leq \beta_2 = \forall x: \beta_1(x) \leq \beta_2(x).$$

- Formulieren Sie den Satz, daß sich die Interpretationen der positiven booleschen Ausdrücke monoton zu den Belegungen verhalten, und beweisen Sie den Satz durch Terminduktion.

Entscheidend ist, daß Sie über dem Aufbau von PBEX beweisen und nicht über BOOL, denn in BOOL gilt der Monotoniesatz nicht.

Aufgabe 6.2 (H) Abstrakte Syntax als Datentyp (wird in der Zentralübung erläutert)

Die folgenden BNF-Regeln beschreiben die Syntax einer einfachen zuweisungsorientierten Sprache (vgl. Broy: Informatik III):

$\langle \text{expression} \rangle ::= \langle \text{identifizier} \rangle \mid 0 \mid 1 \mid (\langle \text{expression} \rangle [+ \mid - \mid *] \langle \text{expression} \rangle)$

$\langle \text{statement} \rangle ::= \mathbf{nop} \mid$

$\mathbf{abort} \mid$

$\langle \text{identifizier} \rangle := \langle \text{expression} \rangle \mid$

$\langle \text{statement} \rangle ; \langle \text{statement} \rangle \mid$

$\mathbf{if} \langle \text{expression} \rangle > 0 \mathbf{then} \langle \text{statement} \rangle \mathbf{fi} \mid$

$\mathbf{while} \langle \text{expression} \rangle > 0 \mathbf{do} \langle \text{statement} \rangle \mathbf{od}$

(Die oben angeführte Sprache von Gordon enthält über die hier angegebene Sprache hinaus beliebige Bedingungen, die durch Vergleichsoperationen auf Expressions gebildet sind, nicht jedoch **abort**.) Definieren Sie die abstrakte Syntax durch Datentypen.