# (Seminar 08. Juli) Semantik für PDL, Ableiten in PDL, Hoare-Kalkül

## 1 Semantik für PDL

Wir stellen uns vor: eine Menge $E$ von Elementar-Aussagen. eine Menge $M$ von Maschinenzuständen,

Dabei ist jedes $m \in M$ eine Funktion, die jeder Aussage $A \in E$ einen Wahrheitswert $\mathsf{Wert}(A, m)$ zuordnet. Ein Zustand ist also eindeutig beschrieben durch die Menge der in ihm wahren Elementaraussagen. Das heißt $M = 2^E$.

Beispiel: zwei Elementar-Aussagen $E = \{e_x, e_y\}$ mit $e_x = (x > 0), e_y = (y > 0)$. Das ergibt 4 Zustände, $M = 2^E = \{s_0, s_x, s_y, s_{xy}\}$ mit $s_0 = \emptyset, s_x = \{e_x\}, s_y = \{e_y\}, s_{xy} = \{e_x, e_y\}\}$.

Jedes Programm $p$ ist eine Relation auf $M$.

Beispiel: $p = (x := x + 1)$ ist die Relation

$$\{(s_0, s_0), (s_0, s_x), (s_x, s_x), (s_y, s_y), (s_y, s_{xy}), (s_{xy}, s_{xy})\}$$

So etwas heißt Kripke-Struktur.

Wir interpretieren $p \cup q$ als Vereinigung der entsprechenden Relationen, $p; q$ als Verkettung und $p^*$ als transitive Hülle. Das heiß Standardmodell (für die PDL-Axiome).

Es gibt auch Nichtstandard-Modelle: die erfüllen alle PDL-Axiome, aber $p^*$ kann noch mehr enthalten als die transitive Hülle.

## 2 Ableiten in PDL

Beispiel: Einen formalen Beweis für die Aufgabe 112 erhalten wir durch Kontraposition von Axiom PDL-6

$$\begin{aligned} A \vee \langle p\rangle\langle p^* A\rangle &\rightarrow \langle p^*\rangle A \\ \neg\langle p^*\rangle A &\rightarrow \neg(A \vee \langle p\rangle\langle p^*\rangle A) \\ [p^*]A &\rightarrow \neg A \wedge \neg\langle p()\langle p^*\rangle A) \\ [p^*]A &\rightarrow \neg A \wedge [p]\neg(\langle p^*\rangle A) \\ [p^*]A &\rightarrow \neg A \wedge [p][p^*]A \end{aligned}$$

Beispiel: Wir wollen nun ableiten:

$$\langle p^*; p^*\rangle A \rightarrow \langle p^*\rangle A$$

Aus $\langle p^*; p^*\rangle A$ folgt nach Axiom 4:

$$\langle p^*\rangle\langle p^*\rangle A$$

Nach Axiom 7 (für $A := \langle p^*\rangle A$) folgt

$$\langle p^*\rangle A \vee \langle p^*\rangle(\neg\langle p^*\rangle A \wedge \langle p\rangle\langle p^*\rangle A)$$

Nach Axiom 2 ist das äquivalent zu

$$\langle p^*\rangle(A \vee (\neg\langle p^*\rangle A \wedge \langle p\rangle\langle p^*\rangle A))$$

Wir zeigen, daß das rechte Argument des $\vee$ immer $\mathsf{False}$ (also widersprüchlich) ist: Nach Axiom 6 gilt

$$\langle p\rangle\langle p^*\rangle A \rightarrow \langle p^*\rangle A$$

also kann nicht gleichzeitig das erste Argument des $\wedge$ wahr sein.

# 3 Hoare-Logik

Wir erinnern an Hoares Notation $\{A\}p\{B\} =_{\text{def}} A \to [p]B$.

In Programmen schreiben wir Zusicherungen (Assertions)

```
#include <assert.h>
assert (A)
p;
assert (B)
```

Tatsächlich wird das Programm p selbst komplizierter sein und wiederum viele Zusicherungen enthalten. Wir beweisen dann über viele Zwischenschritte $A \to [p]B$, mit Hilfe von Erweiterungen von PDL. Wir können Schlußregeln für die Hoare-Notation ableiten:

Für Programmverzweigungen: $\dfrac{\{B \wedge A\}p\{C\}, \{B \wedge \neg A\}q\{C\}}{\{B\} \textbf{ if } A \textbf{ then } p \textbf{ else } q\{C\}}$

Für Schleifen: $\dfrac{\{A \wedge B\}\ p\ \{B\}}{\{B\} \textbf{ while } A \textbf{ do } p\ \{\neg A \wedge B\}}$

$B$ heißt die Schleifen-Invariante, weil diese Aussage **vor, während und nach** der Schleife gilt.

Beispiel: Wir wollen den größten gemeinsamen Teiler von zwei natürlichen Zahlen $a, b$ bestimmen:

```
x := a; y := b;
while  x > 0  and  y > 0  do
   if  x >= y  then  x := x - y
               else  y := y - x
return (x + y);
```

Nennen wir $t = \gcd(a, b)$. Die Invariante ist: $x$ und $y$ sind Vielfache von $t$. Das gilt anfangs und ist offensichtlich invariant: $t \mid x \wedge t \mid y \to t \mid x - y$.

Daraus folgt aber noch nicht, daß schließlich der richtige Wert geliefert wird. (Es könnte sein, daß die Ausgabe z. B. $5t$ ist.) Wir wollen zeigen, daß am Programmende sowohl $a$ als auch $b$ Vielfache von $x + y$ sind.

Wir erweitern das Programm um Variablen $c, d, e, f$ mit der Invariante $a = cx + dy, b = ex + fy$. Übungsaufgabe: schreibe die notwendigen Anweisungen für die neuen Variablen, und beweise:

```
x := a; y := b; c := 1; d := 0; e = 0; f = 1;
while  x > 0  and  y > 0  do
   if  x >= y  then  x := x - y; d := d + c; f := f + e;
               else  y := y - x; c := c + d; e := e + f;
return (x + y);
```

Zum Schluß ist ja $x = 0$ oder $y = 0$, dann gilt $x + y \mid a \wedge x + y \mid b$. Anders gesagt, $x + y \mid t$, weil $t$ der größte gemeinsame Teiler von $a$ und $b$ ist. Von vorhin wußten wir aber $t \mid x + y$. Also bleibt nur $t = x + y$.

Da wir $c, d, e, f$ bei der Rechnung eigentlich nicht benutzen, können wir sie schließlich wieder aus dem Programm streichen.

Durch diesen Trick ist das Programm als **partiell** korrekt bewiesen (d. h. **falls** es hält, dann mit dem richtigen Ergebnis). Wir müssen noch nachweisen, **daß** es hält. Diese Bedingung läßt sich nicht in PDL ausdrücken. Wir können immer nur über alle erreichbare Knoten des Berechnungsbaums quantifizieren, aber nicht sagen, wieviele das nun sein sollen oder wie tief sie liegen dürfen.

Zum Beweis der **totalen Korrektheit** (= Termination und partielle Korrektheit) gibt es andere Methoden. Für **while**-Schleifen führen wir eine **Schranke** mit, das ist ein Ausdruck $S$, der nur ganzzahlige Werte annimmt und für den gilt

1. der Wert der Schranke nach dem Schleifenkörper ist gesunken:
   $\{B \wedge h = S\}p\{S < h\}$

2. wenn die Schranke $< 0$ ist, wird die Schleife verlassen:
   $B \wedge S < 0 \to \neg A$

Im Fall des gcd-Algorithmus genügt als Schranke $x + y$. (Beweis!)

Kalküle für Programmkorrektheit sind Gegenstand der Vorlesung Programmverifikation, Fragen der Termination werden auch in der Termersetzung behandelt (es gibt dort noch viel kompliziertere, aber trotzdem wohlfundierte Ordnungen, bezüglich denen wir Schranken bauen können). Zu beiden Themen siehe die Vorlesungen im Hauptstudium von Dr. Hartwig
(Skripte: `http://www.informatik.uni-leipzig.de/~rhartwig/rh_skri.html`)