3. Hoare-Logik

Ziel:
- Zeige/Beweise, dass ein Programm den Effekt erzielt, für den es gemacht wurde.
- Bisher haben wir nur einzelne Initialzustände betrachtet.
- Um Korrektheit zu zeigen, betrachte alle (geeigneten) Eingaben.

Vorsicht:
- Zeige Aussagen der Form

  Falls $x \leq 1$ vor Ausführung des Programms c gilt,

  dann gilt nach Ausführung $y > 2$

  — sofern das Programm terminiert.

- Das ist partielle Korrektheit,

  wobei partiell bedeutet, dass das Programm nicht terminieren muss.

  Im Fall von Divergenz werden keinerlei Garantien gegeben.
- Um Terminierung auszuzeichen, betrachtet man totale Korrektheit.

  Totale Korrektheitsaussagen haben die Form:

  Falls vor der Ausführung von Programm c $x \leq 1$ gilt,

  dann terminiert c und liefert bei Beendigung $y > 2$.

  Totale Korrektheit werden wir nicht betrachten.
- Partielle Korrektheitsaussagen werden als Hoare-Tripel

$$\{x \leq 1\}\ c\ \{y > 2\}$$

  dargestellt, nach Sir Tony Hoare (* 1934, Turing-Award 1980).

  In so einem Hoare-Tripel ist:

  ↳ $x \leq 1$ die Vorbedingung,

  die die Startzustände σ von Interesse charakterisiert.

↳ c das Programm

↳ $y > 2$ die Nachbedingung, die angibt,

welche Eigenschaften Endzustände σ' mit $(c, \sigma) \Downarrow \sigma'$

erfüllen.

Über Nicht-Endzustände wird keine Aussage gemacht.

- Vor- und Nachbedingungen (Engl. Pre- / Postconditions) heißen auch Assertions.

Bemerkung:

- Das ist die Hoare Schule der (manuellen) Programmverifikation
Sie geht zurück auf Turings Paper "Checking a large routine" von 1949.
Die Pioniere sind Robert Floyd (1936-2001, Turing-Award 1970) und Tony Hoare.
- Mit zunehmender Automatisierung ist der Ansatz heute Mainstream und wird genutzt unter Anderem bei
Cadence, Facebook, Amazon, ARM, INTEL, IBM und INRIA:)
- Ursprünglich war der Fokus nicht allein
Zum Nachweis von Programmeigenschaften entwickelt worden.
Es ging auch darum, die Bedeutung von Programmierkonstrukten zu klären.
Die Bedeutung eines Konstrukts ist dadurch gegeben als
↳ Axiome für Befehle und
↳ Regeln für Kompositionsoperatoren.
Daher spricht man auch von axiomatischer Semantik.
- Heute ist der Begriff Programmlogiken (Program Logics) gängiger.

Wir brauchen eine Sprache für Assertions.

Definition:

Assertions sind Formeln der Prädikatenlogik erster Stufe über der Signatur von While-Programmen:

$$A ::= 0 \mid 1 \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg A \mid A_1 \wedge A_2 \mid \exists c : A.$$

Dabei ist

- i eine ganzzahlige Variable, die von den Programmvariablen unterschiedlich ist.
- a_1, a_2 sind arithmetische Ausdrücke, die neben den Programmvariablen auch ganzzahlige Variablen enthalten dürfen.
- Assertions sind abgeschlossen bzgl. ganzzahligen Variablen, soll heißen, die treten nur quantifiziert auf.

Bemerkung:

- Man kann die Signatur des Programms erweitern, um in Assertions weitere Funktionssymbole und Prädikatssymbole zur Verfügung zu haben.
- Ebenso könnte man den Datenbereich erweitern; Auxiliary Information, Ghost Variables.
- Wir schreiben statt $[\![A]\!]\sigma = 1$ auch $S, \sigma \models A$. Da S ja fest ist, schreiben wir auch $\sigma \models A$. Man sagt, σ erfüllt A.

Definition:

Ein Hoare-Tripel (auch partielle Korrektheitsaussage genannt) hat die Form

$$\{A\}\, c\, \{B\}$$

mit A, B Assertions und c einem Programm.

Das Tripel ist gültig, geschrieben als $\models \{A\}\, c\, \{B\}$, falls

$$\forall \sigma, \sigma' \in State.\quad \sigma \models A \wedge (c, \sigma) \Downarrow \sigma' \quad \Rightarrow \quad \sigma' \models B.$$

Beispiel: einfach besser zu lesen als 1

- $\models \{true\}$ **while** true **do** skip **od** $\{A\}$ gilt für alle A, da das Programm nicht terminiert.
- $\models \{true\}$ **while** $x \leq 2$ **do** skip **od** $\{x > 2\}$ gilt, da das Programm nur für $x > 2$ terminiert.
- $\models \{s = 0 \wedge n = 1\}$ **while** $n < 101$ **do** $s := s + n;\ n := n + 1$ **od** $\{s = \sum_{i=1}^{100} i\}$

Bemerkung:

- Die Definition funktioniert auch für nicht-deterministische Programme.
- Wir haben den Begriff gültig nur für die Struktur S definiert. Genauer wäre es, S-gültig zu sagen. In der Logik bedeutet gültig nämlich gültig in allen Strukturen. Da es hier nicht zur Verwechslung kommen kann, bleiben wir bei gültig.

3.1 Der Hoare-Kalkül

Ziel: • Gültigkeit von Hoare-Tripeln ist ein semantisches Problem.

• Um Gültigkeit algorithmisch zu prüfen
(wir wollen das automatisieren),
brauchen wir ein syntaktisches Gegenstück.

• Wir führen jetzt ein Beweissystem (Kalkül) ein,
dessen Theoreme genau die gültigen Hoare-Tripel sind.

Definition (Hoare '69):
Die Axiome und Regeln von Hoares Beweissystem,
kurz Hoare-Regeln, lauten wie folgt:

(SKIP) $\dfrac{}{\{A\}\ \mathrm{skip}\ \{A\}}$ $\qquad$ $\dfrac{}{\{A[x/a]\}\ x := a\ \{A\}}$ (ASSIGN)

(SEQ) $\dfrac{\{A\}\ c_1\ \{C\} \quad \{C\}\ c_2\ \{B\}}{\{A\}\ c_1; c_2\ \{B\}}$ $\qquad$ $\dfrac{\{A \wedge b\}\ c_1\ \{B\} \quad \{A \wedge \neg b\}\ c_2\ \{B\}}{\{A\}\ \mathrm{if}\ b\ \mathrm{then}\ c_1\ \mathrm{else}\ c_2\ \mathrm{fi}\ \{B\}}$ (IF)

(WHILE) $\dfrac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \mathrm{while}\ b\ \mathrm{do}\ c\ \mathrm{od}\ \{A \wedge \neg b\}}$ $\qquad$ $\dfrac{A \to A' \quad \{A'\}\ c\ \{B'\} \quad B' \to B}{\{A\}\ c\ \{B\}}$ (CONSEQUENCE)

Falls es in obigem Kalkül einen Beweis für $\{A\}\ c\ \{B\}$ gibt,
schreiben wir $\vdash \{A\}\ c\ \{B\}$.