4. Verification Conditions

Ziel: Entwickle ein Verfahren, das die Korrektheit von Programmen, formuliert als Gültigkeit

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

von Hoare-Tripeln, (vollständig) automatisch zeigt.

Idee:
- Mit dem vorherigen Kapitel ist

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

  äquivalent zu

  $$\models A \to pred(c, B),$$

  die Implikation soll also allgemeingültig sein.
- Beachte, dass $A \to pred(c,B)$ eine logische Formel ist, die komplett frei von dem Programm c ist. Das heißt, die Programmanalyse über die Implikation geschieht rein in Logik.
- Die Hoffnung ist, dass Solver (SAT/SMT-Solver, Theorembeweiser, Constraint-Solver) die Allgemeingültigkeit solcher Implikationen automatisch zeigen können.

Probleme:
- Für while-Schleifen ist $pred(c,B)$ eine sehr schwierige Formel (lang + enthält (geschachtelte) Quantoren). Implikationen, die sich auf eine solche Formel beziehen, wird ein Solver nicht nachweisen können.
- Im Vorfeld eine einfachere Schleifeninvariante zu berechnen, ist ebenfalls unmöglich.

  Letztlich ist Verifikation unentscheidbar!

Ansatz: · Erwerb Guidance vom Programmieren.

· Annotiere das Programm mit Schleifeninvarianten, um die Aufgabe des Solvers zu vereinfachen.

Definition:

· Die Syntax annotierter While-Programme ist

$$c ::= skip \mid x := a \mid \underline{assume}\ b \quad // \text{ wie bisher}$$
$$\mid c_1 ; c_2 \mid \underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi}$$
$$\mid \underline{while}\ b\ \underline{do}\ \{I\}\ c\ \underline{od}$$

Dabei ist I eine Assertion.

· Wir bisher schreiben wir $\models \{A\}\, c\, \{B\}$ auch für annotierte Programme und meinen damit, $\models \{A\}\, drop(c)\, \{B\}$, das Hoare-Tripel für das Programm ohne Annotationen ($drop(c)$) ist gültig.

Beispiel (Fakultät):

$c \equiv$ <u>while</u> $x > 0$ <u>do</u>

$\{ y * x! = n! \wedge x \geq 0 \}$

$y := y * x;$

$x := x - 1;$

<u>od</u>

Ansatz (cont.): · Jeden annotierten Hoare-Tripel $\{A\}\, c\, \{B\}$ ordnen wir eine Menge an Zusicherungen zu, so dass Folgendes gilt:

Sind die Zusicherungen gültig, gilt auch das Hoare-Tripel.

-2-

· Die Menge an Zusicherungen nennt man

Verifikations-Conditions, geschrieben $vc(\{A\}c\{B\})$.

Vorgehen:

(i) Betrachte zunächst Programme c_1, c_2 ohne while und ohne ;.

Für das Hoare-Tripel $\{A\}\, c_1; c_2\, \{B\}$

können wir mit Dijkstras Sak

eine Zwischenzusicherung D berechnen:

$$\{A\}\, c_1;\, \{D\}\, c_2\, \{B\} \quad \text{mit} \quad D = pred(c_2, B).$$

Dann ist $\{A\}\, c_1; c_2\, \{B\}$ gültig gdw.

$\{A\}\, c_1\, \{D\}$ und $\{D\}\, c_2\, \{B\}$ gültig sind.

Auf diese Weise annotieren wir das gesamte Programm.

Nun müssen wir nur noch die Gültigkeit der einzelnen

$$\{A\}\, prim\, \{D\}$$

prüfen, wobei $prim ::= skip \mid x := a \mid \underline{assume}\ b$

eine primitive Anweisung ist.

Dazu prüfen wir die Verifikations-Condition

$$A \to pred(prim, B) \quad \text{auf Allgemeingültigkeit.}$$

(ii) Betrachte nun $\{A\}\, c; w\, \{B\}$,

wobei $w = \underline{while}\ b\ \underline{do}\ \{I\}\, c'\ \underline{od}$.

Wir zuvor suchen wir eine Annotation

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

so dass $\vdash \{A\}\, c\, \{D\}$ und $\vdash \{D\}\, w\, \{B\}$.

Damit wiederum

$$\vdash \{D\}\, w\, \{B\},$$

bilden wir (WHILE) + (CONSEQUENCE) und fordern

$$D \Rightarrow I, \quad \vdash \{I \wedge b\}\, c\, \{I\}, \quad I \wedge \neg b \Rightarrow B.$$

Da D gesucht ist, können wir auch gleich das schwächste $D := I$ wählen.

Zusammenfassung:

Die gesucht Zwischenzusicherung ist

$$\{I\}\, c;\, [I]\, w\, \{B\}.$$

Außerdem erhalten wir die Verpflichtung,

$$\{I \wedge b\}\, c\, \{I\}$$

mit Annotationen zu versehen und

$$I \wedge \neg b \Rightarrow B$$

zu zeigen.

Anschaulich:

- Wir unterteilen das Programm in Basis-Paths.
  Ein Basis-Path ist ein azyklisches Programm (ohne while, potentiell mit if), das zwischen zwei der folgenden Konstrukte liegt:
  Start / Ende des gegebenen Programms,
  Eintritt / Verlassen einer While-Schleife,
  Start / Ende eines Schleifenrumpfs.
- Der Begriff Pfad wird deutlich, wenn wir das Programm als Kontrollflussgraphen sehen und Schleifen

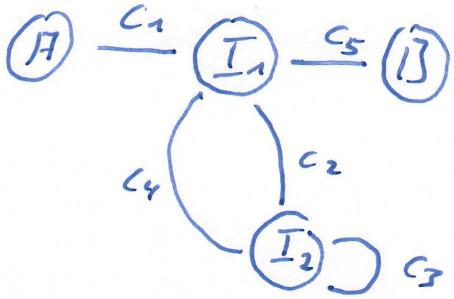
durch ihre Invarianten repräsentieren.

Beispiel:

Seien c_1 bis c_5 atomar.

```
{A}
 c1;
 while b1 do
   {I1}
    c2;
    while b2 do
      {I2}
        c3
    od
    c4;
 od
 c5;
{B}
```

als Kontrollflussgraph



Die Basic-Paths sind c_1 bis c_5.

- Die Verifikations-Conditions des Hoare-Tripels sind die Verifikations-Conditions von

$\{A\}\ c_1\ \{I_1\}$ $\qquad$ $\{I_2 \wedge \neg b_2\}\ c_4\ \{I_1\}$

$\{I_1 \wedge b_1\}\ c_2\ \{I_2\}$ $\qquad$ $\{I_1 \wedge \neg b_1\}\ c_5\ \{B\}$,

$\{I_2 \wedge b_2\}\ c_3\ \{I_2\}$ $\qquad$ wie sie in (i) definiert wurden.

- Im Grunde zerschneiden wir die ∞-vielen Pfade durch das Programm in endlich viele Teilstücke.

Definition:

Die Funktion vc berechnet zu jedem Hoare-Tripel $\{A\}\ c\ \{B\}$, wobei c ein annotiertes Programm ist, eine endliche Menge an Aussagen $vc(\{A\}\ c\ \{B\}) \subseteq \mathcal{L}$ wie folgt:

$$vc(\{A\}\ skip\ \{B\}) := \{A \to B\}$$

$$vc(\{A\}\ x := a\ \{B\}) := \{A \to B[x/a]\}$$

$$vc(\{A\}\ \underline{assume}\ b\ \{B\}) := \{A \to (b \to B)\}$$

$$vc(\{A\}\ \underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi}\ \{B\}) := vc(\{A \wedge b\}\ c_1\ \{B\}) \cup vc(\{A \wedge \neg b\}\ c_2\ \{B\})$$

$$vc(\{A\}\ \underline{while}\ b\ \underline{do}\ \{I\}\ c\ \underline{od}\ \{B\}) := vc(\{I \wedge b\}\ c\ \{I\}) \cup \{A \to I,\ I \wedge \neg b \to B\}$$

$$vc(\{A\}\ c_1; c_2\ \{B\}) := vc(\{A\}\ c_1\ \{pred(c_2, B)\}) \cup vc(\{pred(c_2, B)\}\ c_2\ \{B\}),$$

wobei pred auf while-Schleifen vermieden wird:

$$pred(w, B) := I, \quad \text{wobei } w = \underline{while}\ \_\ \underline{do}\ \{I\}\ \_\ \underline{od}.$$

Satz (Korrektheit):

$\models vc(\{A\}\ c\ \{B\})$ impliziert $\models \{A\}\ c\ \{B\}$.

Beweis mit Induktion nach der Struktur von Programmen.

Beispiel:

Betrachte $c \equiv$ <u>while</u> $x > 0$ <u>do</u>

$\{I\}$

$y := y * x;$

$x := x - 1;$

<u>od</u>

mit $I = y * x! = n! \wedge x \geq 0$

und $A = x = n \wedge n > 0 \wedge y = 1$

$B = y = n!$

$$
\begin{aligned}
& vc(\{A\}\, c\, \{B\}) \\
= \;& vc(\{I \wedge x>0\}\; y := y*x;\; x := x-1\; \{I\}) \\
& \cup \{A \to I,\; I \wedge \neg(x>0) \to B\} \\
= \;& vc(\{I \wedge x>0\}\; y := y*x\; \{\underbrace{pred(x := x-1, I)}_{I[x/x-1]}\}) \\
& \cup vc(\{I[x/x-1]\}\; x := x-1\; \{I\}) \\
& \cup \{A \to I,\; I \wedge \neg(x>0) \to B\} \\
= \;& \{I \wedge x>0 \to I[x/x-1][y/y*x],\\
& \quad I[x/x-1] \to I[x/x-1],\\
& \quad A \to I,\; I \wedge \neg(x>0) \to B\}.
\end{aligned}
$$

Diese Menge ist allgemeingültig,
also folgt mit obigem Satz $\vdash \{A\}\, c\, \{B\}$.
Diese Allgemeingültigkeit lässt sich schnell
mit SAT/SMT-Solvern prüfen (rise4fun.com/Z3). □

Lemma (Incompleteness):
$\models \{A\}\, c\, \{B\}$ impliziert nicht $\vdash vc(\{A\}\, c\, \{B\})$.

Beweis:
Betrachte $\vdash \{true\}$ while false do $\{false\}$ skip od $\{true\}$.
Es gilt:
$$
\begin{aligned}
& vc(\{true\} \text{ while false do } \{false\} \text{ skip od } \{true\}) \\
= \;& \{false \wedge false \to false,\; false \wedge \neg false \to true,\; true \to false\}
\end{aligned}
$$
Aber $true \to false$ ist nicht allgemeingültig,
obwohl das Hoare-Tripel valide ist. □