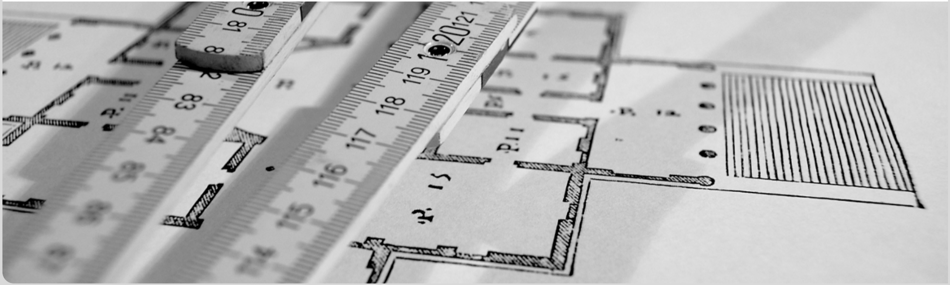
Grundbegriffe der Informatik
Tutorium 36

Termin 9 | 23.12.2016
Thassilo Helmold

KIT – Karlsruher Institut für Technologie



Inhalt

Algorithmen: Hoare-Kalkül

Abbildung: https://www.xkcd.com/1185

In the previous episode of GBI...

Rückblick: Prädikatenlogik

- Deutlich komplizierterer Aufbau als Aussagenlogik
- Auswertung mit Interpretation und Variablenbelegung
- Quantoren erlauben allgemeine Aussagen

Wahr oder Falsch?

Sei $Rel_{PL} = \{R, S\}$ mit $ar(R) = 2$ und $ar(S) = 1$,
$Fun_{PL} = \{f, g\}$ mit $ar(f) = 1$ und $ar(g) = 2$

- $R(y, g(x, y))$ ist präd.log. syntaktisch korrekt. W
- $f(S(x))$ ist präd.log. syntaktisch korrekt. F
  Eine Relation kann nicht innerhalb einer Funktion auftauchen, da sie kein Term, sondern eine atomare Formel darstellt.
- „Nicht alle Kinder spielen nicht“ $\equiv \forall x(child(x) \to play(x))$ F
  Es wird nur ausgesagt, dass es mindestens ein Kind gibt, das spielt.

Algorithmen: Hoare-Kalkül

Definition

Über die Eigenschaften von Algorithmen:

- Eine endliche Beschreibung...
- ...aus elementaren Aussagen...
- ... die deterministisch ausgeführt werden.
  (Manchmal auch gemischt mit (Pseudo-)Zufallselementen)
- Eine endliche Eingabe gibt endliche Ausgabe...
- in endlich vielen Schritten.
- Das funktioniert für beliebig große Eingaben und
- ist nachvollziehbar bzw. verständlich

Woher wissen wir, ob ein Algorithmus korrekt ist?

Korrektheit

Einige Algorithmen haben besonders hohe Anforderungen an die Korrektheit: Banking-Server, Airbag-Steuerprogramm, Herzschrittmacher, ...

Wie können wir die Korrektheit beweisen?

- Testen? Was, wenn wir einen Sonderfall vergessen?
- Alle Eingaben testen? Oft nicht möglich.
- Formal beweisen: Hoare-Kalkül

In der Praxis

Theoretisch müsste die komplette Werkzeugkette bewiesen werden: Programm, Compiler, Prozessor...
Oft wird bei Compilern nur "Proven in use" benutzt: Compiler, bei denen seit Jahren keine Fehler gefunden wurden.

Das Hoare-Kalkül

Definition

Ein *Hoare-Tripel* ist ein Tupel $(\{P\}\, S\, \{Q\})$ mit einem Programmstück S und prädikatenlogischen *Zusicherungen* P, Q.

$P =$ Bedingung vor der Ausführung
$Q =$ Zusicherung nach der Ausführung
$S =$ Programmstück

Dabei: Wir betrachten nur „relevante“ Interpretationen:

- Fester Grundbereich (explizit angegeben oder implizit ableitbar)
- Funktionen und Relationen „wie üblich“ interpretiert.
- Konstanten beliebig, als „Eingabe“ des Programms.
  Muss also für alle Möglichkeiten = Eingaben gelten.

Das Hoare-Kalkül

Definition

Ein Hoare-Tripel $\{P\}\ S\ \{Q\}$ ist gültig, wenn für jede relevante Interpretation und jede Variablenbelegung β gilt:
Wenn vor der Ausführung $val_{D,I,\beta}(P) = \mathbf{w}$ ist und wenn die Ausführung von S für I und β endet und hinterher Variablenbelegung β' vorliegt, dann gilt am Ende $val_{D,I,\beta'}(Q) = \mathbf{w}$.

Hoare-Kalkül

Das *Hoare-Kalkül* definiert Regeln, wie gültige *Hoare-Tripel* schrittweise abgeleitet werden können.

HT-A

Axiom HT-A

$$\{\sigma_{x/E}(Q)\}\ x \leftarrow E\ \{Q\}$$

Nach einer Zuweisung gilt jede Aussage für die Variable, welche vorher für die linke Seite der Zuweisung galt.

- $\sigma_{x/E}(Q)$ ist die Aussage, die dadurch entsteht, dass man in Q jedes freie Vorkommen von x durch E ersetzt.
- Achtung: $\sigma_{x/E}$ muss kollisionfrei sein!

Beispiel

$\{x+1=43\}\ y := x+1\ \{y=43\}$.

HT-E

Regel HT-E

Wenn $\{P\}\, S\, \{Q\}$ gültig ist, dann auch $\{P'\}\, S\, \{Q'\}$ mit $P' \implies P$ und $Q \implies Q'$.

Vorbedingungen können stärker, Nachbedingungen können schwächer werden.

HT-S

Regel HT-S

Wenn $\{P\}\ S_1\ \{Q\}$ und $\{Q\}\ S_2\ \{R\}$ gültig sind, dann auch $\{P\}\ S_1 S_2\ \{R\}$.

Hoare-Tripel können transitiv zusammengefasst werden.

Beispiel

zeige Ableitbarkeit von $\{x = a\}\ y \leftarrow x; z \leftarrow y\ \{z = a\}$

$$\boxed{\{\ x = a\ \}}$$

$$y \leftarrow x$$

$$\boxed{\{\ y = a\ \}}$$

$$\boxed{\{\ y = a\ \}}$$

$$z \leftarrow y$$

$$\boxed{\{\ z = a\ \}}$$

- auseinander ziehen
- HT-A: $\{y = a\}\ z \leftarrow y\ \{z = a\}$ ist ableitbar
- HT-A: $\{x = a\}\ y \leftarrow x\ \{y = a\}$ ist ableitbar
- HT-S: fertig

HT-I

$\{P\}$
if B
then
    $\{P \wedge B\}$
    S_1
    $\{Q\}$
else
    $\{P \wedge \neg B\}$
    S_2
    $\{Q\}$
fi
$\{Q\}$

Regel HT-I

if B **then** S_1 **else** S_2 **fi**

- Wenn $\{P \wedge B\}S_1\{Q\}$ gültig
- und $\{P \wedge \neg B\}S_2\{Q\}$ gültig
- dann auch
  $\{P\}$ **if** B **then** S_1 **else** S_2 **fi**$\{Q\}$ gültig

Beispiel $|x|$

$\{x \in \mathbb{R}\}$

if $x < 0$

then

$\{z \in \mathbb{R} \wedge x < 0\}$

$\{-x = |x|\}$

$z \leftarrow -x$

$\{z = |x|\}$

else

$\{z \in \mathbb{R} \wedge x \geqslant 0\}$

$\{x = |x|\}$

$z \leftarrow x$

$\{z = |x|\}$

fi

$\{z = |x|\}$

Aufgabe

$\{ \ x = a \wedge y = b \ \}$

if $x > y$

then

$\{ \ \ldots \ \}$

$z \leftarrow y$

$\{ \ \ldots \ \}$

else

$\{ \ \ldots \ \}$

$z \leftarrow x$

$\{ \ \ldots \ \}$

fi

$\{ \ z = \min(a, b) \ \}$

Lösung

$\{\ x = a \wedge y = b\ \}$

if $x > y$

then

$\{\ x = a \wedge y = b \wedge x > y\ \}$

$\{\ y = \min(a, b)\ \}$

$z \leftarrow y$

$\{\ z = \min(a, b)\ \}$

else

$\{\ x = a \wedge y = b \wedge \neg(x > y)\ \}$

$\{\ x = \min(a, b)\ \}$

$z \leftarrow x$

$\{\ z = \min(a, b)\ \}$

fi

$\{\ z = \min(a, b)\ \}$

Regel HT-W — für **while**-Schleifen

$\{ I \}$

while B

do

$\{ I \wedge B \}$

S

$\{ I \}$

od

$\{ I \wedge \neg B \}$

HT-W: $$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{I \wedge \neg B\}}$$

Zusicherung I heißt Schleifeninvariante

Gültigkeit „bleibt erhalten“

Schleifeninvarianten

Schleifeninvarianten …

- sind Aussagen, die bei jedem Schleifendurchgang gültig sind
- helfen, die Korrektheit eines Programmes zu beweisen
- muss man ebenfalls beweisen
- garantieren nicht die Korrektheit des Programms:
  Terminierung der Schleife muss zusätzlich gezeigt werden!

Beispiel

$\{\ x = a \wedge y = b\ \}$

$\{\ \ldots\ \}$

while $y \neq 0$

do

$\{\ \ldots\ \}$

$y \leftarrow y - 1$

$\{\ \ldots\ \}$

$x \leftarrow x + 1$

$\{\ \ldots\ \}$

od

$\{\ \ldots\ \}$

$\{\ x = a + b\ \}$

Wertetabelle für a=3 und b=4

i	x	y
0	3	4
1	4	3
2	5	2
3	6	1
4	7	0

Schleifeninvariante :

$$x_i + y_i = a + b$$

Beispiel

$$
\begin{array}{l}
\{\ x = a \wedge y = b\ \} \\
\{\ x + y = a + b\ \} \\
\textbf{while } y \neq 0 \\
\textbf{do} \\
\qquad \{\ x + y = a + b \wedge y \neq 0\ \} \\
\qquad \{\ x + 1 + y - 1 = a + b\ \} \\
\qquad y \leftarrow y - 1 \\
\qquad \{\ x + 1 + y = a + b\ \} \\
\qquad x \leftarrow x + 1 \\
\qquad \{\ x + y = a + b\ \} \\
\textbf{od} \\
\{\ x + y = a + b \wedge (y = 0)\ \} \\
\{\ x = a + b\ \}
\end{array}
$$

SIV mit Vollständiger Induktion

Wir zeigen mit vollständiger Induktion die Gültigkeit der Schleifeninvariante.
Dabei sei i die Anzahl der bisherigen Schleifendurchläufe.
Behauptung:

$$\forall\ i \in \{0, b\} : x_i + y_i = a + b$$

Induktionsanfang

Für $i = 0$ gilt $x_0 + y_0 = a + b$ nach Vorbedingung

Induktionsvorrausetzung

Für ein beliebig aber festes $i \in \{0, b\}$ gelte die Behauptung

SIV mit Vollständiger Induktion

Induktionsschluß

Zu zeigen

$$x_{i+1} + y_{i+1} = a + b$$

$$\begin{aligned} x_{i+1} + y_{i+1} &= x_i + 1 + y_i - 1 \\ &= x_i + y_i \\ &\overset{IV}{=} a + b \end{aligned}$$

Weitere Beispiele

Weitere Beispiele findet ihr hier: Übung 8, WS 15/16

Was ihr nun wissen solltet

- Wie das Hoare-Kalkül funktioniert
- Wie man mit dem Hoare-Kalkül ein Programm beweist.

Was nächstes Mal kommt

- Graphen - Alles Vernetzt
- Systematisches Suchen und Wandern - Algorithmen auf Graphen

Frohe Weihnachten und einen guten Start in das neue Jahr!

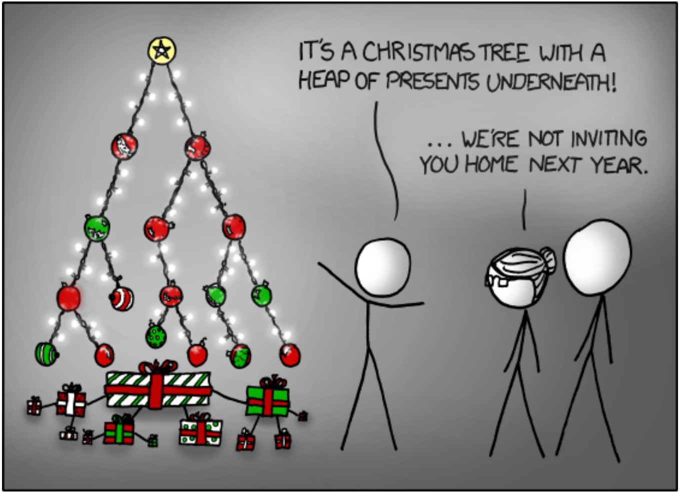


Abbildung: `https://www.xkcd.com/835`

Danksagung

Dieser Foliensatz basiert in Teilen auf Folien von:

Philipp Basler
Nils Braun
Dominik Doerner
Ou Yue