Verification Condition Generation

Barie, Viktor

TU Kaiserslautern

November 28, 2014

Inhaltsverzeichnis

Inhaltsverzeichnis

Programmverifikation

Wie verifiziert man dieses Programm?

$$x := 1$$

$$y := x + 1$$

Programmverifikation

Wie verifiziert man dieses Programm?

$$x := 1$$

$$y := x + 1$$

$$\{y = 2\}$$

Programmverifikation

Wie verifiziert man dieses Programm?

$$
\begin{array}{l}
\quad x := 1 \\
\quad \{x := 1\} \\
\quad y := x + 1 \\
\{y = 2\}
\end{array}
$$

Programmverifikation

Wie verifiziert man dieses Programm?

$\{true\}$
$\quad x := 1$
$\quad \{x := 1\}$
$\quad y := x + 1$
$\{y = 2\}$

Programmverifikation

if $x \leq 0$ *then*

$x := 1$

else

SKIP

Programmverifikation

if $x \leq 0$ *then*

$x := 1$

else

SKIP

$\{x > 0\}$

Programmverifikation

if $x \leq 0$ *then*

$x := 1$

else

SKIP

$\{x > 0\}$

$\{x > 0\}$

Programmverifikation

$\{x > 0\}$

if $x \leq 0$ *then*

$x := 1$

else

$\{x > 0\}$

SKIP

$\{x > 0\}$

$\{x > 0\}$

Programmverifikation

$\{x > 0\}$

if $x \leq 0$ *then*

$x := 1$

$\{x > 0\}$

else

$\{x > 0\}$

SKIP

$\{x > 0\}$

Programmverifikation

$$
\begin{array}{l}
\{x > 0\} \\
\quad \textit{if } x \leq 0 \textit{ then} \\
\qquad \{\textit{true}\} \\
\qquad x := 1 \\
\qquad \{x > 0\} \\
\quad \textit{else} \\
\qquad \{x > 0\} \\
\qquad \textit{SKIP} \\
\qquad \{x > 0\}
\end{array}
$$

Programmverifikation

$\{(x \leq 0 \ \wedge \mathit{true}) \vee (\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{\mathit{true}\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

Programmverifikation

$\{true\}$
$\{(x \leq 0 \ \wedge true) \vee (\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{true\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

Programmverifikation

$$\textit{while } x \leq 0 \textit{ do}$$

$$x := x + 1$$

Programmverifikation

$$\textit{while } x \leq 0 \textit{ do}$$

$$x := x + 1$$

$$\{x > 0\}$$

Programmverifikation

$$\textit{while } x \leq 0 \textit{ do}$$

$$x := x + 1$$

$$\{x \leq 0 \vee x > 0\}$$

$$\{x > 0\}$$

Programmverifikation

$$
\begin{array}{l}
\textit{while } x \leq 0 \textit{ do} \\
\quad \{x \leq 0\} \\
\quad x := x + 1 \\
\quad \{x \leq 0 \vee x > 0\} \\
\{x > 0\}
\end{array}
$$

Programmverifikation

$\{x \leq 0 \vee \neg x \leq 0\}$
while $x \leq 0$ *do*
$\{x \leq 0\}$
$x := x + 1$
$\{x \leq 0 \vee x > 0\}$
$\{x > 0\}$

Programmverifikation

$\{true\}$
$\quad \{x \leq 0 \vee \neg x \leq 0\}$
$\quad$ *while* $x \leq 0$ *do*
$\quad\quad \{x \leq 0\}$
$\quad\quad x := x + 1$
$\quad\quad \{x \leq 0 \vee x > 0\}$
$\{x > 0\}$

Programmverifikation

@Vorbedingung P: $(0 \leq l) \wedge (u < |a|)$

```
    bool LinearSearch(int[] a, int l, int u, int e) {
    int i := l;
        @Invariante I: (l ≤ i) ∧ (∀j((l ≤ j < i) → (a[j] ≠ e)))
        while (i ≤ u) do{
            if (a[i] = e) return true;
            i := i+1
        }
    return false;
    }
```

@Nachbedingung Q: $rv \leftrightarrow \exists i((l \leq i \leq u) \wedge (a[i] = e))$

Inhaltsverzeichnis

Hoare-Logik

1: $\dfrac{}{\{Q\}\ SKIP\ \{Q\}}$

Hoare-Logik

$$1: \frac{}{\{Q\}\ SKIP\ \{Q\}}$$

$$2: \frac{}{\{Q[u/t]\}\ u := t\ \{Q\}}$$

Hoare-Logik

$$1: \frac{}{\{Q\}\ SKIP\ \{Q\}}$$

$$2: \frac{}{\{Q[u/t]\}\ u := t\ \{Q\}}$$

$$3: \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1\ ;\ c_2\ \{Q\}}$$

Hoare-Logik

$$1: \frac{}{\{Q\}\ SKIP\ \{Q\}}$$

$$2: \frac{}{\{Q[u/t]\}\ u := t\ \{Q\}}$$

$$3: \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1\ ;\ c_2\ \{Q\}}$$

$$4: \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{if}\ \ b\ \mathit{then}\ c_1\ \mathit{else}\ c_2\ \{Q\}}$$

Hoare-Logik

$$1: \frac{}{\{Q\}\ SKIP\ \{Q\}}$$

$$2: \frac{}{\{Q[u/t]\}\ u := t\ \{Q\}}$$

$$3: \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1\ ;\ c_2\ \{Q\}}$$

$$4: \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{if}\ b\ \mathit{then}\ c_1\ \mathit{else}\ c_2\ \{Q\}}$$

$$5: \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \mathit{while}\ b\ \mathit{do}\ c\ \{P \wedge \neg b\}}$$

Hoare-Logik

$$1: \frac{}{\{Q\}\ SKIP\ \{Q\}}$$

$$2: \frac{}{\{Q[u/t]\}\ u := t\ \{Q\}}$$

$$3: \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1\ ;\ c_2\ \{Q\}}$$

$$4: \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \mathit{if}\ b\ \mathit{then}\ c_1\ \mathit{else}\ c_2\ \{Q\}}$$

$$5: \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \mathit{while}\ b\ \mathit{do}\ c\ \{P \wedge \neg b\}}$$

$$6: \frac{(P' \rightarrow P)\ \{P\}\ c\ \{Q\}\ (Q \rightarrow Q')}{\{P'\}\ c\ \{Q'\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\frac{\{x \leq 1 \wedge x \leq 0\} x := x + 1 \{x \leq 1\}}{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\frac{\{x \leq 0\} x := x + 1 \{x \leq 1\}}{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\dfrac{\dfrac{}{\{x + 1 \leq 1\} x := x + 1 \{x \leq 1\}}}{\{x \leq 0\} x := x + 1 \{x \leq 1\}}$$
$$\overline{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\dfrac{\dfrac{\dfrac{}{\{x + 1 \leq 1\} x := x + 1 \{x \leq 1\}}}{\{x \leq 0\} x := x + 1 \{x \leq 1\}}}{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\dfrac{\dfrac{(x \leq 0) \rightarrow (x \leq 1) \qquad \overline{\{x + 1 \leq 1\} x := x + 1 \{x \leq 1\}}}{\{x \leq 0\} x := x + 1 \{x \leq 1\}}}{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Hoare-Logik

$$\{x \leq 0\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x > 0\}$$

$$\dfrac{\dfrac{(x \leq 0) \rightarrow (x \leq 1) \qquad \{x + 1 \leq 1\} x := x + 1 \{x \leq 1\}}{\{x \leq 0\} x := x + 1 \{x \leq 1\}}}{\{x \leq 1\} \mathit{while}\ x \leq 0\ \mathit{do}\ x := x + 1 \{x \leq 1 \wedge x > 0\}}$$

Inhaltsverzeichnis

Verification Conditions

$P \rightarrow WP(Q, c_i)$

- Vorbedingung P
- Nachbedingung Q
- Ausführungen des Programms c_i
- Berechnung der Weakest Precondition WP

Berechnung von WP

$$
\begin{aligned}
&WP(Q, c_1 \ldots c_n) && \Leftrightarrow WP(WP(Q, c_n), c_1 \ldots c_{n-1}) \\
&WP(Q, SKIP) && \Leftrightarrow Q \\
&WP(Q[x], x := e) && \Leftrightarrow Q[e] \\
&WP(Q, \textit{if } b \textit{ then } c_1 \textit{ else } c_2) && \Leftrightarrow b \rightarrow WP(Q, c_1) \wedge \neg b \rightarrow WP(Q, c_2) \\
&WP(Q, \textit{while } \{I\} \; b \textit{ do } c) && \Leftrightarrow (b \rightarrow WP(I, c)) \wedge (\neg b \rightarrow Q))
\end{aligned}
$$

Berechnung mit WP

$\{true\}$
$\{(x \leq 0 \ \wedge true) \vee$
$(\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{true\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

Berechnung mit WP

$\{true\}$
$\{(x \leq 0 \ \wedge true) \vee$
$(\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{true\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

- $b \rightarrow WP(Q, c_1) \wedge \neg b \rightarrow WP(Q, c_2)$

Berechnung mit WP

$\{true\}$
$\{(x \leq 0 \ \wedge true) \vee$
$(\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{true\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

- $b \rightarrow WP(Q, c_1) \wedge \neg b \rightarrow WP(Q, c_2)$
- Ausrechnen und Umformen ergibt Äquivalenz beider Formeln

Berechnung mit WP

$\{true\}$
$\{(x \leq 0 \ \wedge true) \vee$
$(\neg x \leq 0 \wedge x > 0)\}$
if $x \leq 0$ *then*
$\{true\}$
$x := 1$
$\{x > 0\}$
else
$\{x > 0\}$
SKIP
$\{x > 0\}$
$\{x > 0\}$

- $b \rightarrow WP(Q, c_1) \wedge \neg b \rightarrow WP(Q, c_2)$
- Ausrechnen und Umformen ergibt Äquivalenz beider Formeln
- VC des Programms:
  $true \rightarrow$
  $([(x \leq 0) \rightarrow (1 > 0)] \wedge$
  $[(x > 0) \rightarrow (x > 0)])$

Probleme solcher Verification Conditions

Problem:

- VC's können in ihrer Größe explodieren
- Formulierung der Invarianten nicht trivial

Basic Paths

Lösung: Basic Paths

- Zerlegung des Programms in mehrere kleine Programmausschnitte
- Starten an der Vorbedingung oder einer Invariante
- Enden an einer Invariante oder der Nachbedingung
- Guards werden ersetzt durch Annahmen (*assume*)
  - Ein Basic Path mit Annahme *assume* b
  - Ein Basic Path mit Annahme *assume* $\neg b$

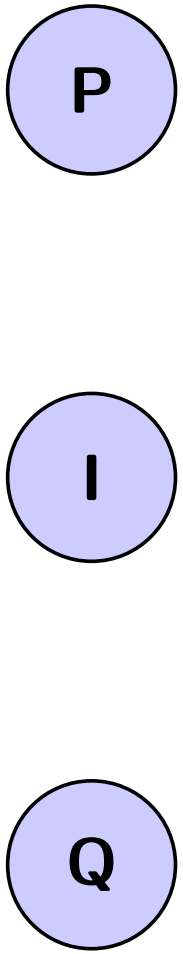
Berechnung von WP mit Basic Paths

Anpassung an WP:

$$
\begin{aligned}
WP(Q, c_1 \ldots c_n) &\Leftrightarrow WP(WP(Q, c_n), c_1 \ldots c_{n-1}) \\
WP(Q, SKIP) &\Leftrightarrow Q \\
WP(Q[x], x := e) &\Leftrightarrow Q[e] \\
WP(Q, assume\ b) &\Leftrightarrow b \rightarrow Q
\end{aligned}
$$

Basic Paths eines Programms mit While-Schleife

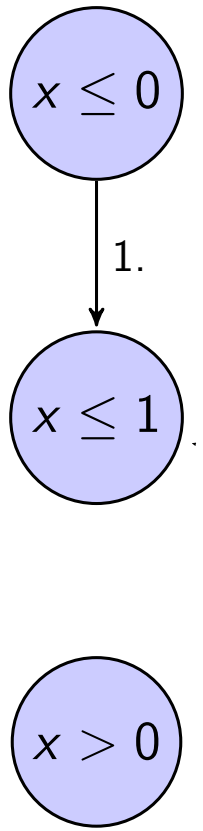
$\{x \leq 0\}$
$\quad \{I : x \leq 1\}$
$\quad\quad$ *while* $x \leq 0$ *do*
$\quad\quad\quad x := x + 1$
$\{x > 0\}$



Basic Paths eines Programms mit While-Schleife

$\{x \leq 0\}$
$\quad \{I : x \leq 1\}$
$\quad\quad$ *while* $x \leq 0$ *do*
$\quad\quad\quad x := x + 1$
$\{x > 0\}$

VC1: $x \leq 0 \rightarrow x \leq 1$

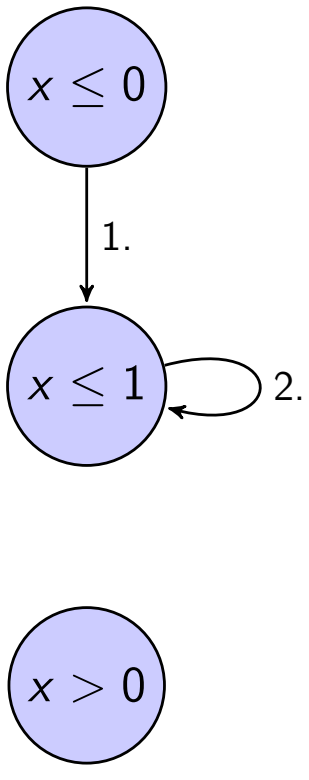


Basic Paths eines Programms mit While-Schleife

$\{x \leq 0\}$
$\quad \{I : x \leq 1\}$
$\quad\quad$ *while* $x \leq 0$ *do*
$\quad\quad\quad x := x + 1$
$\{x > 0\}$

VC1: $x \leq 0 \rightarrow x \leq 1$

VC2: $x \leq 1 \rightarrow (x \leq 0 \rightarrow x \leq 0)$



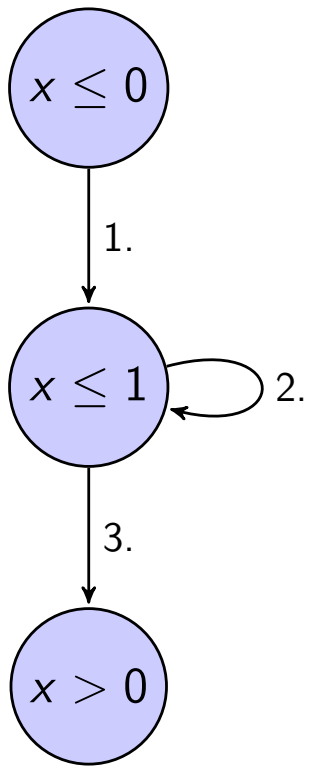
Basic Paths eines Programms mit While-Schleife

$\{x \leq 0\}$
$\quad \{I : x \leq 1\}$
$\quad\quad$ *while* $x \leq 0$ *do*
$\quad\quad\quad x := x + 1$
$\{x > 0\}$

VC1: $x \leq 0 \rightarrow x \leq 1$

VC2: $x \leq 1 \rightarrow (x \leq 0 \rightarrow x \leq 0)$

VC3: $x \leq 1 \rightarrow (x > 0 \rightarrow x > 0)$



P-Invariant und P-Induktiv

Theorem

Wenn für jeden Basic Path $\{F\}c_1 \ldots c_n\{G\}$ von Programm P die Verification Condition gültig ist, dann sind die Annotationen von P P-Induktiv.

Beweis durch Induktion über die Anzahl an Basic Paths in einem Lauf eines Programms P

Inhaltsverzeichnis

ESC

- Behandlung von Guards ähnlich wie bei Basic Paths

ESC

- Behandlung von Guards ähnlich wie bei Basic Paths
- Assert-Statements sind Annotationen des Programmierers

ESC

- Behandlung von Guards ähnlich wie bei Basic Paths
- Assert-Statements sind Annotationen des Programmierers
- Behandlung von Schleifen

ESC

- Behandlung von Guards ähnlich wie bei Basic Paths
- Assert-Statements sind Annotationen des Programmierers
- Behandlung von Schleifen

  while {Invariante I} b do c end; wird überführt in

  $assert\ I;$
  $x_1 := y_1; \ldots; x_n := y_n$
  $assume\ I;$
  $((assume\ b; c; assert\ I; assume\ false)\ [\!]\ assume\ \neg b)$

ESC

- Behandlung von Guards ähnlich wie bei Basic Paths
- Assert-Statements sind Annotationen des Programmierers
- Behandlung von Schleifen

  while {Invariante I} b do c end; wird überführt in

  $assert\ I;$
  $x_1 := y_1; \ldots; x_n := y_n$
  $assume\ I;$
  $((assume\ b; c; assert\ I; assume\ false)\ [\!]\ assume\ \neg b)$

- Wertzuweisungen: $x := e$ wird zu $assume\ x' = e$

WP unter ESC

- Programm S liegt in passiver Form vor

WP unter ESC

- Programm S liegt in passiver Form vor
- Mögliche Zustände sind
  - N.S: Programm S kann normal terminieren
  - W.S: Programm S kann schief gehen

WP unter ESC

- Programm S liegt in passiver Form vor
- Mögliche Zustände sind
  - N.S: Programm S kann normal terminieren
  - W.S: Programm S kann schief gehen

Definition

S	$N.S$	$W.S$
$assert\ e$	e	$\neg e$
$assume\ e$	e	$false$
$A; B$	$N.A \wedge N.B$	$W.A \vee (N.A \wedge W.B)$
$A \,[\!]\, B$	$N.A \vee N.B$	$W.A \vee W.B$

ESC

$$VC_{esc}.S := \neg(W.S) \wedge (N.S \rightarrow Q)$$

ESC

$$VC_{esc}.S := \neg(W.S) \wedge (N.S \rightarrow Q)$$

Theorem

Wenn S in passiver Form ist, dann ist

- $|N.S|$ *in* $\mathcal{O}(|S|)$, *und*
- $|W.S|$ *in* $\mathcal{O}(|S^2|)$.

Proof.

Durch strukturelle Induktion über S. □

ESC

- K. R. M. Leino weist darauf hin:
  - Für eine eingeschränkte Programmklasse,
  - mit der Weakest Liberal Precondition (WLP) gilt:
  - $VC_{esc}.S = WP.S.Q$

ESC

- K. R. M. Leino weist darauf hin:
  - Für eine eingeschränkte Programmklasse,
  - mit der Weakest Liberal Precondition (WLP) gilt:
  - $VC_{esc}.S = WP.S.Q$
- WP terminiert unter erfüllter Vorbedingung immer
- WLP muss nicht terminieren

ESC

- K. R. M. Leino weist darauf hin:
  - Für eine eingeschränkte Programmklasse,
  - mit der Weakest Liberal Precondition (WLP) gilt:
  - $VC_{esc}.S = WP.S.Q$
- WP terminiert unter erfüllter Vorbedingung immer
- WLP muss nicht terminieren

Lemma

$\forall Q(WP(Q,S)) \equiv \ WP(true,S) \wedge \ WLP(Q,S)$

ESC

- K. R. M. Leino weist darauf hin:
  - Für eine eingeschränkte Programmklasse,
  - mit der Weakest Liberal Precondition (WLP) gilt:
  - $VC_{esc}.S = WP.S.Q$
- WP terminiert unter erfüllter Vorbedingung immer
- WLP muss nicht terminieren

Lemma

$\forall Q(WP(Q,S)) \equiv \ WP(true,S) \wedge \ WLP(Q,S)$

Zur Erinnerung:
$VC_{esc}.S := \neg(W.S) \wedge (N.S \rightarrow Q)$

Inhaltsverzeichnis

Funktionsaufrufe in Basic Path

Problem:

Rekursion führt zu unbeschränkt vielen Aufrufen

Funktionsaufrufe in Basic Path

Problem:

Rekursion führt zu unbeschränkt vielen Aufrufen

Lösung: Summaries

- Summaries ersetzen diese Aufrufe innerhalb eines Basic Paths
- Summaries sind auf den Funktionsaufruf angepasste Nachbedingungen

Funktionsaufrufe in Basic Path

Problem:

Rekursion führt zu unbeschränkt vielen Aufrufen

Lösung: Summaries

- Summaries ersetzen diese Aufrufe innerhalb eines Basic Paths
- Summaries sind auf den Funktionsaufruf angepasste Nachbedingungen

Beispiel:

$f(x)$ mit Nachbedingung $\{rv > x\}$ und einem Aufruf $y := f(k+1)$

Funktionsaufrufe in Basic Path

Problem:
Rekursion führt zu unbeschränkt vielen Aufrufen

Lösung: Summaries

- Summaries ersetzen diese Aufrufe innerhalb eines Basic Paths
- Summaries sind auf den Funktionsaufruf angepasste Nachbedingungen

Beispiel:
$f(x)$ mit Nachbedingung $\{rv > x\}$ und einem Aufruf $y := f(k+1)$

Im zugehörigen Basic Path wird der Aufruf zu
assume $y = rv > (k+1)$

Pointer im Hoare-Kalkül

Problem: Hoare-Kalkül nicht für Pointer ausgelegt

Pointer im Hoare-Kalkül

Problem: Hoare-Kalkül nicht für Pointer ausgelegt

Beispiel: Betrachte folgendes Hoare-Tripel mit Pointer Aliasing

$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$

Pointer im Hoare-Kalkül

Problem: Hoare-Kalkül nicht für Pointer ausgelegt

Beispiel: Betrachte folgendes Hoare-Tripel mit Pointer Aliasing

$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$

- Korrekt für $\{P : y^* = 5\}$ oder $\{P : x = y\}$

Pointer im Hoare-Kalkül

Problem: Hoare-Kalkül nicht für Pointer ausgelegt

Beispiel: Betrachte folgendes Hoare-Tripel mit Pointer Aliasing

$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$

- Korrekt für $\{P : y^* = 5\}$ oder $\{P : x = y\}$
- Anwendung der Wertzuweisung ergibt:

$$\begin{aligned} P &:= \quad Q[x^* = 5] \\ P &:= \quad 5 + y^* = 10 \\ P &:= \quad y^* = 5 \end{aligned}$$

Pointer im Hoare-Kalkül

Problem: Hoare-Kalkül nicht für Pointer ausgelegt

Beispiel: Betrachte folgendes Hoare-Tripel mit Pointer Aliasing

$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$

- Korrekt für $\{P : y^* = 5\}$ oder $\{P : x = y\}$
- Anwendung der Wertzuweisung ergibt:

$$\begin{aligned} P := \quad & Q[x^* = 5] \\ P := \quad & 5 + y^* = 10 \\ P := \quad & y^* = 5 \end{aligned}$$

- Der Fall $x = y$ ist verloren gegangen

Erweiterung des Hoare-Kalküls

Lösung:

- Betrachtung des Speichers als ein Objekt M

Erweiterung des Hoare-Kalküls

Lösung:

- Betrachtung des Speichers als ein Objekt M
- Einführung von Regeln die M manipulieren
  - $upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
  - $sel(M, E_2)$ liest M an Adresse E_1
  - $$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

Erweiterung des Hoare-Kalküls

Lösung:

- Betrachtung des Speichers als ein Objekt M
- Einführung von Regeln die M manipulieren
  - $upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
  - $sel(M, E_2)$ liest M an Adresse E_1
  - $sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$
- Anpassung der Wertzuweisungsregel

$$\{P : Q[upd(M, E_1, E_2)]/M\} E_1^* := E_2 \{Q\}$$

Angepasste Wertzuweisungsregel

$$\{P : Q[upd(M, E_1, E_2)]/M\}E_1^* := E_2\{Q\}$$

$$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$$
$$P : Q[upd(M, x, 5)/M]$$

Angepasste Wertzuweisungsregel

$$\{P : Q[upd(M, E_1, E_2)]/M\}E_1^* := E_2\{Q\}$$

$$\{P\}x^* := 5\{Q : x^* + y^* = 10\}$$

$$P : Q[upd(M, x, 5)/M]$$

$$P : x^* + y^* = 10[upd(M, x, 5)/M]$$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$P : x^* + y^* = 10[upd(M, x, 5)/M]$

$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$$P : x^* + y^* = 10[upd(M, x, 5)/M]$$

$$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$$

$$P : sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10$$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$P : x^* + y^* = 10[upd(M, x, 5)/M]$

$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$

$P : sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10$

$P : 5 + sel(upd(M, x, 5), y) = 10$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$P : x^* + y^* = 10[upd(M, x, 5)/M]$

$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$

$P : sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10$

$P : 5 + sel(upd(M, x, 5), y) = 10$

$P : sel(upd(M, x, 5), y) = 5$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$P : x^* + y^* = 10[upd(M, x, 5)/M]$

$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$

$P : sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10$

$P : 5 + sel(upd(M, x, 5), y) = 10$

$P : sel(upd(M, x, 5), y) = 5$

$P : (x = y \wedge 5 = 5) \vee (x \neq y \wedge sel(M, y) = 5)$

Angepasste Wertzuweisungsregel

$upd(M, E_1, E_2)$ aktualisiert M an Adresse E_1 mit dem Wert E_2
$sel(M, E_2)$ liest M an Adresse E_1

$$sel(upd(M, E_1, E_2), E_3) = \begin{cases} E_2, & \text{falls } E_1 = E_3 \\ sel(M, E_3), & \text{falls } E_1 \neq E_3 \end{cases}$$

$P : x^* + y^* = 10[upd(M, x, 5)/M]$

$P : sel(M, x) + sel(M, y) = 10[upd(M, x, 5)/M]$

$P : sel(upd(M, x, 5), x) + sel(upd(M, x, 5), y) = 10$

$P : 5 + sel(upd(M, x, 5), y) = 10$

$P : sel(upd(M, x, 5), y) = 5$

$P : (x = y \wedge 5 = 5) \vee (x \neq y \wedge sel(M, y) = 5)$

$P : x = y \vee y^* = 5$

Quellen

Hoare:

- K. R. Apt and E.-R. Olderog. Programmverifikation. 1994.
  item C. A. R. Hoare. An axiomatic basis for computer programming. Commun. ACM,
- Ranjit Jhala. Floyd-hoare logic. University Lecture, 2013.

Verification Conditions:

- Aaron R. Bradley and Zohar Manna. The Calculus of Computation: Decision Procedures with Applications to Verification

ESC:

- Cormac Flanagan and James B. Saxe. Avoiding exponential explosion: Generating compact verification conditions.
- K. Rustan M. Leino. Efficient weakest preconditions. Technical Report MSR-TR-2004-34, Microsoft Research, April 2004.