Korrektheitsbeweise für Algorithmen

Jürgen Stuber

2013-10-02

Korrektheit

Zwei Aspekte:

1. partielle Korrektheit: falls der Programmablauf endet, ist das Ergenis richtig

Korrektheit

Zwei Aspekte:

1. partielle Korrektheit: falls der Programmablauf endet, ist das Ergenis richtig
2. Terminierung: der Programmablauf endet

Partielle Korrektheit

- Vorbedingung (precondition):
  Eigenschaft der Eingabedaten, wird als wahr vorausgesetzt

Partielle Korrektheit

- Vorbedingung (precondition):
  Eigenschaft der Eingabedaten, wird als wahr vorausgesetzt
- Nachbedingung (postcondition):
  Eigenschaft der Ausgabedaten, muss aus der Vorbedingung nach Programmablauf folgen

Partielle Korrektheit

- Vorbedingung (precondition):
  Eigenschaft der Eingabedaten, wird als wahr vorausgesetzt
- Nachbedingung (postcondition):
  Eigenschaft der Ausgabedaten, muss aus der Vorbedingung nach Programmablauf folgen

- Programm: Funktion, Methode, Algorithmus, ...

Partielle Korrektheit

- Vorbedingung (precondition):
  Eigenschaft der Eingabedaten, wird als wahr vorausgesetzt
- Nachbedingung (postcondition):
  Eigenschaft der Ausgabedaten, muss aus der Vorbedingung nach Programmablauf folgen

- Programm: Funktion, Methode, Algorithmus, ...
- Eingabedaten: Parameter, sichtbare Variablen (z.B. globale)

Partielle Korrektheit

- Vorbedingung (precondition):
  Eigenschaft der Eingabedaten, wird als wahr vorausgesetzt
- Nachbedingung (postcondition):
  Eigenschaft der Ausgabedaten, muss aus der Vorbedingung nach Programmablauf folgen

- Programm: Funktion, Methode, Algorithmus, ...
- Eingabedaten: Parameter, sichtbare Variablen (z.B. globale)
- Ausgabedaten: Rückgabewert, sichtbare Variablen

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm
G: Nachbedingung

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm
G: Nachbedingung

Axiom für Nichtstun:

$$\{F\}\, \texttt{skip}\, \{F\}$$

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm
G: Nachbedingung

Axiom für Nichtstun:

$$\{F\}\, \texttt{skip}\, \{F\}$$

Axiom für Zuweisung:

$$\{F[e/x]\}\, x := e\, \{F\}$$

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm
G: Nachbedingung

Axiom für Nichtstun:

$$\{F\}\, \texttt{skip}\, \{F\}$$

Axiom für Zuweisung:

$$\{F[e/x]\}\, x := e\, \{F\}$$

$F[e/x]$: Ersetze alle Vorkommen von x in F durch e

Hoare-Kalkül

Formale Korrektheitsbeweise für imperative Programme (Hoare 1969):

$$\{F\}\, P\, \{G\}$$

F: Vorbedingung, logische Formel, die Variablennamen enthalten darf
P: Programm
G: Nachbedingung

Axiom für Nichtstun:

$$\{F\}\, \texttt{skip}\, \{F\}$$

Axiom für Zuweisung:

$$\{F[e/x]\}\, x := e\, \{F\}$$

$F[e/x]$: Ersetze alle Vorkommen von x in F durch e

Beispiel: $\{\mathtt{x} + 1 < n\}\, \mathtt{x} := \mathtt{x} + 1\, \{\mathtt{x} < n\}$

Hoare-Kalkül Regeln

Anpassen der Vor- und Nachbedingungen:

$$\frac{A \Rightarrow B \quad \{B\}\, P\, \{C\} \quad C \Rightarrow D}{\{A\}\, P\, \{D\}}$$

Hoare-Kalkül Regeln

Anpassen der Vor- und Nachbedingungen:

$$\frac{A \Rightarrow B \quad \{B\}\, P\, \{C\} \quad C \Rightarrow D}{\{A\}\, P\, \{D\}}$$

Sequenz:

$$\frac{\{A\}\, P\, \{B\} \quad \{B\}\, Q\, \{C\}}{\{A\}\, P; Q\, \{C\}}$$

Hoare-Kalkül Regeln

Anpassen der Vor- und Nachbedingungen:

$$\frac{A \Rightarrow B \quad \{B\}\, P\, \{C\} \quad C \Rightarrow D}{\{A\}\, P\, \{D\}}$$

Sequenz:

$$\frac{\{A\}\, P\, \{B\} \quad \{B\}\, Q\, \{C\}}{\{A\}\, P; Q\, \{C\}}$$

`if-then-else`:

$$\frac{\{A \wedge c\}\, P\, \{B\} \quad \{A \wedge \neg c\}\, Q\, \{B\}}{\{A\}\, \texttt{if}\ c\ \texttt{then}\ P\ \texttt{else}\ Q\, \{B\}}$$

Hoare-Kalkül Regeln

`while`:

$$\frac{\{A \wedge c\}\, P\, \{A\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

A: Schleifeninvariante (loop invariant)

Beispiel

Terminierung

Schleifen müssen enden

Terminierung

Schleifen müssen enden

`while`-Regel für totale Korrektheit:

$$\frac{\{A \wedge c \wedge t = z\}\, P\, \{A \wedge t < z\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

Terminierung

Schleifen müssen enden

`while`-Regel für totale Korrektheit:

$$\frac{\{A \wedge c \wedge t = z\}\, P\, \{A \wedge t < z\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

t: Term

Terminierung

Schleifen müssen enden

`while`-Regel für totale Korrektheit:

$$\frac{\{A \wedge c \wedge t = z\}\, P\, \{A \wedge t < z\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

t: Term

z: zusätzliche Variable, steht für den Wert von t vor der Iteration

Terminierung

Schleifen müssen enden

`while`-Regel für totale Korrektheit:

$$\frac{\{A \wedge c \wedge t = z\}\, P\, \{A \wedge t < z\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

t: Term
z: zusätzliche Variable, steht für den Wert von t vor der Iteration
$<$: Wohlordnung, d.h. absteigende Ketten $t_1 > t_2 > t_3 > \ldots$ müssen enden

Terminierung

Schleifen müssen enden

`while`-Regel für totale Korrektheit:

$$\frac{\{A \wedge c \wedge t = z\}\, P\, \{A \wedge t < z\}}{\{A\}\, \texttt{while}\, c\, \texttt{do}\, P\, \{A \wedge \neg c\}}$$

t: Term
z: zusätzliche Variable, steht für den Wert von t vor der Iteration
$<$: Wohlordnung, d.h. absteigende Ketten $t_1 > t_2 > t_3 > \ldots$ müssen enden

Z.B. natürliche Zahlen, viele andere Möglichkeiten (schön: Erweiterung auf Multimengen)

Anmerkungen

- "weakest precondition": von der Nachbedingung zurückrechnen, welche Vorbedingung gebraucht wird (geht aber nicht bei Schleifeninvarianten)

Anmerkungen

- "weakest precondition": von der Nachbedingung zurückrechnen, welche Vorbedingung gebraucht wird (geht aber nicht bei Schleifeninvarianten)
- Großer Aufwand, wird in der Praxis nur an wenigen kritischen Stellen meist auf kleine Programmteile angewendet

Anmerkungen

- "weakest precondition": von der Nachbedingung zurückrechnen, welche Vorbedingung gebraucht wird (geht aber nicht bei Schleifeninvarianten)
- Großer Aufwand, wird in der Praxis nur an wenigen kritischen Stellen meist auf kleine Programmteile angewendet
  - sicherheitskritische Teile

Anmerkungen

- "weakest precondition": von der Nachbedingung zurückrechnen, welche Vorbedingung gebraucht wird (geht aber nicht bei Schleifeninvarianten)
- Großer Aufwand, wird in der Praxis nur an wenigen kritischen Stellen meist auf kleine Programmteile angewendet
  - sicherheitskritische Teile
  - einzelne Algorithmen und Datenstrukturen

Anmerkungen

- "weakest precondition": von der Nachbedingung zurückrechnen, welche Vorbedingung gebraucht wird (geht aber nicht bei Schleifeninvarianten)
- Großer Aufwand, wird in der Praxis nur an wenigen kritischen Stellen meist auf kleine Programmteile angewendet
  - sicherheitskritische Teile
  - einzelne Algorithmen und Datenstrukturen
- Programmiersprache Eiffel unterstützt Vor- und Nachbedingungen

Weitere Ansätze

Hoare-Kalkül nur ein Ansatz für sequentielle imperative Programme
(der älteste und einfachste)

Weitere Ansätze

Hoare-Kalkül nur ein Ansatz für sequentielle imperative Programme
(der älteste und einfachste)

Viele andere, für viele verschieden Situationen und Anwendungsfälle
`https://en.wikipedia.org/wiki/Formal_verification`
(der englischsprachige Artikel!)