Dijkstras Guarded Commands - Bewachte Anweisungen und Zusicherungen

Manfred Broy

Anweisung

- Eine Anweisung
  - ist ein Element einer Programmiersprache
  - dient der Änderung des Zustands
  - definiert eine Zustandsänderung
- Anweisungen arbeiten in der Regel über attributierten Zuständen; dann ist der Zustandsraum gegeben durch
  - eine endliche Zahl von Attributen (auch Programmvariable genannt)
  - jedes Attribut hat einen Typ (eine Sorte)
- Ein Zustand ist dann gegeben
  - durch die Belegung der Programmvariablen mit Werten
  - wobei jeder Wert dem Typ der entsprechenden Variable entspricht

Beispiel: Attributierter Zustand

- Deklaration der Attribute

  x : Var Nat, y : Var Bool, z : Var Integer

- Belegung

x	y	z
17	true	-1333

Guarded Comands - Bewachte Anweisungen

Syntaktische Form	Bezeichnung
nop	Leere Anweisung
abort	Divergierende Anweisung (Anweisung, die nie terminiert)
$x_1, ..., x_n := E_1, ..., E_n$	Kollektive Zuweisung
$S_1; S_2$	Sequentielle Komposition
if C_1 **then** S_1 [] C_2 **then** S_2 ... [] C_n **then** S_n **fi**	Satz bewachter Anweisungen
do C_1 **then** S_1 [] C_2 **then** S_2 ... [] C_n **then** S_n **od**	Wiederholungsanweisung

Zusicherungen

- Eine Zusicherung ist eine prädikatenlogische Formel (ein Ausdruck der Sorte bool) über den Attributen eines Programms, die (Programmvariable) eine Aussage über den Zustand darstellt.
- Beispiel:

$$x > 10 \wedge y \wedge z < 1000$$

- Zusicherungen können mit Hilfe der sogenannten Zusicherungslogik (auch „Hoare Logik“) dazu verwendet werden, die Korrektheit von Programmen logisch zu beweisen.

Korrektheit

- Eine mit Zusicherungen annotierte Anweisung S

  $$\{ P \}\ S\ \{ Q \}$$

  heißt korrekt (bzgl. der Vorbedingung P und der Nachbedingung Q) falls für jeden Zustand folgende Aussage zutrifft:

  Wird die Anweisung S in einem Zustand ausgeführt, für den die Vorbedingung P gilt, so wird - falls die Ausführung der Anweisung S terminiert - ein Zustand erreicht, in dem die Nachbedingung Q gilt.
- Wir sprechen bei

  $$\{ P \}\ S\ \{ Q \}$$

  auch von einem Hoare-Triple.

Verifikation

- Durch Zusicherungen können wir
  - imperative Programme spezifizieren
  - Aussagen zu ihrer Korrektheit formulieren
- In einer Verifikation weisen wir nach oder überprüfen wir, dass eine Anweisung korrekt ist.
- Methoden des Nachweises
  - Test
  - Inspektion
  - Logische Verifikation
  - Überprüfung aller erreichbaren Zustände (bei endlichen Automaten) durch Model-Checking

Annotierte Anweisungen

- Eine (zusammengesetzte) Anweisung heißt vollständig (durch Zuweisungen) annotiert, wenn vor und nach jeder Anweisung eine Zusicherung steht.
  - Die Annotation, die zu Beginn steht, heißt Vorbedingung.
- Ein annotiertes Programm heißt korrekt (annotiert), wenn für jede Ausführung des Programms ausgehend von einem Zustand, in dem die Vorbedingung gilt, alle durchlaufenen Zusicherungen in den jeweils dann eingenommen Zuständen gelten.

Bubblesort auf Sequenzen - annotiert mit Zusicherungen

var seq nat a, z , s;
{a = ⟨⟩ ∧ z = s}
do z ≠ ⟨⟩ then
{sorted(a) ∧ a°z ≈ s ∧ z ≠ ⟨⟩ }
if a == ⟨⟩ then {a = ⟨⟩ ∧ z ≈ s ∧ z ≠ ⟨⟩ }

a, z := ⟨first(z)⟩, rest(z)
{sorted(a) ∧ a°z ≈ s}
[] a =/= ⟨⟩ ∧ last(a) ≤ first(z) then {sorted(a) ∧ a°z ≈ s ∧ last(a) ≤ first(z) ∧ z ≠ ⟨⟩}
a, z := a°⟨first(z)⟩, rest(z)
{sorted(a) ∧ a°z ≈ s }
[] a =/= ⟨⟩ ∧ last(a) > first(z) then {sorted(a) ∧ a°z ≈ s ∧ last(a) > first(z) ∧ z ≠ ⟨⟩}
a, z := lrest(a), ⟨first(z)⟩°⟨last(a)⟩°rest(z)
{sorted(a) ∧ a°z ≈ s ∧ z ≠ ⟨⟩}
fi {sorted(a) ∧ a°z ≈ s }
od
{sorted(a) ∧ a°z ≈ s ∧ z = ⟨⟩ }
{sorted(a) ∧ a ≈ s}

Definition der Hilfsprädikate

fct sorted = (seq nat s) bool:
    if #s < 2 then true
               else first(s) ≤ first(rest(s)) ∧ sorted(rest(s))
    fi

a ≈ s = (∀ nat k: k#a = k#s)

mit
    k#⟨⟩ = 0
    k#⟨j⟩°s = if k == j then 1+ k#s else k#s fi

Sinn annotierter Programme

- Ein annotiertes Programm kann durch (manuelle) Inspektion der Zusicherungen auf Korrektheit überprüft werden.
- Dazu wird für jede Anweisung überprüft,
  - dass für jede Ausführung des Programms ausgehend von einem Zustand, in dem die Vorbedingung der Anweisung gilt, nach Ausführung der Anweisung die Zusicherung nach der Anweisung gilt.
- Für jede Art von Anweisungen gibt es formale Regeln, die diesen Zusammenhang herstellen (Zusicherungskalkül, Hoare-Kalkül)

Die Regel für Zuweisungen

- Für Zuweisungen gilt („Zuweisungsaxiom“):

$$\{ Q[E/x] \} \quad x := E \quad \{ Q \}$$

falls E nur einfache Namen für Variablen enthält (kein Aliasing, keine Referenzen auf Variable).

Beispiel:

$$\{ 0 < x+1 \} \quad x := x+1 \quad \{ 0 < x \}$$

$$\{\text{sorted}(a) \wedge a \circ z \approx s \wedge \text{last}(a) \leq \text{first}(z) \}$$
$$\{\text{sorted}(a \circ \langle \text{first}(z) \rangle) \wedge a \circ \langle \text{first}(z) \rangle \circ \text{rest}(z) \approx s \}$$
$$a, z := a \circ \langle \text{first}(z) \rangle, \text{rest}(z)$$
$$\{\text{sorted}(a) \wedge a \circ z \approx s \}$$

Die Regel für bedingte Anweisungen

- Für bedingte Anweisungen gilt:

  $\{ Q \}$
  if C then $\{ C \wedge Q \}$ S1 $\{ R \}$
  [] $\neg$ C then $\{ \neg C \wedge Q \}$ S2 $\{ R \}$
  fi
  $\{ R \}$

  falls
  $\{ C \wedge Q \}$ S1 $\{ R \}$
  und
  $\{ \neg C \wedge Q \}$ S2 $\{ R \}$
  gilt

- Nebenbedingungen: C enthält keine Seiteneffekte

Beispiel: Die Regel für bedingte Anweisungen

$\{ x > 1 \wedge y > 1 \}$

if $x \leq y$ then $\{ x \leq y \wedge x > 1 \wedge y > 1 \}\ z := x\ \{ z = \min(x, y) \wedge x > 1 \wedge y > 1 \}$

[] $x > y$ then $\{ x > y \wedge x > 1 \wedge y > 1 \}\ z := y\ \{ z = \min(x, y) \wedge x > 1 \wedge y > 1 \}$

fi

$\{ z = \min(x, y) \wedge x > 1 \wedge y > 1 \}$

falls

$\{ x \leq y \wedge x > 1 \wedge y > 1 \}\ z := x\ \{ z = \min(x, y) \wedge x > 1 \wedge y > 1 \}$

und

$\{ x > y \wedge x > 1 \wedge y > 1 \}\ z := y\ \{ z = \min(x, y) \wedge x > 1 \wedge y > 1 \}$

gilt

Die Regel für aufeinander folgende Zusicherungen

- Sind in einem annotierten Programm zwei Zusicherungen nicht durch eine Anweisung getrennt sondern folgen sie unmittelbar aufeinander, so gilt die Annotation

$$\{ Q \}$$
$$\{ R \}$$

falls $\{ Q \}$ und

$$Q \Rightarrow R$$

gilt

Die Regel für die Wiederholungsanweisung: Invariante

- Gilt für eine Wiederholungsanweisung

  do C then S od

  die Annotation

  $\{C \wedge Q\}\ S\ \{Q\}$

  dann ist folgendes Programm korrekt annotiert

  $\{Q\}$

  do C then $\{C \wedge Q\}\ S\ \{Q\}$ od

  $\{\neg C \wedge Q\}$
- Die Zusicherung Q heißt dann Invariante.
- Durch Invariante lassen sich Wiederholungsanweisungen verifizieren.
- Nebenbedingungen: C enthält keine Seiteneffekte
- Achtung: Terminierung ist hier nicht gefordert!

Invarianten - Bubblesort

$\{a = \langle\rangle \wedge z = s\}$
$\{sorted(a) \wedge a \circ z \approx s\}$
do $z \neq \langle\rangle$ then
$\{sorted(a) \wedge a \circ z \approx s \wedge z \neq \langle\rangle\}$
...
$\{sorted(a) \wedge a \circ z \approx s\}$
od
$\{sorted(a) \wedge a \circ z \approx s \wedge z = \langle\rangle\}$

Invariante: $Q = sorted(a) \wedge a \circ z \approx s$