Verifikation von while-Schleifen Beweisskizzen und Annotierte Programme

Martin Wirsing

in Zusammenarbeit mit
Matthias Hölzl, Piotr Kosiuczenko, Dirk Pattinson

04/03

Ziele

- Anwendung des Hoare-Kalkül zur Verifikation von **`while`**-Programmen
- Beweisskizzen zur Darstellung von Hoare-Kalkül-Beweisen
- Annotierte Programme in Java zum Darstellen und Testen von Zusicherungen

Zusicherungen

Zusicherung: Vorbedingung

{ x==a & y==b }

```
while (x > 0)
{ y = y+1; x = x-1;

}
```

{ y== a+b }

Zusicherung: Nachbedingung

Durch logische Variable (wie a,b) kann man sich auf „alte" Werte beziehen

Was berechnet dieses Programm? **Die Summe von a und b?**

Zusicherungen in Java

```
        -20
int a = 23; b = 25;
int x = a, y = b;
assert x==a & y==b;

while (x > 0)
{ y = y+1; x = x-1;

}
assert y== a+b;
```

Zusicherung

Für a=23 und b=25 gilt, dass das Programm die Summe von a und b berechnet.

Für a= -20 und b=25 gilt dies NICHT!

Mit dem assert-Statement können Zusicherungen getestet, aber nicht verifiziert werden!

Realisierung in Java

- In Java kann man Zusicherungen der Form {P} durch die Anweisung

  `assert P`

  in das Programm einführen.
- Durch > javac -source 1.4 <Dateiname>.java

  > java -ea <Dateiname>

  werden Programme mit Zusicherungen übersetzt und ausgeführt.
- Ist bei der Programmausführung eine der Zusicherungen falsch, so bricht das Programm mit einem Fehler ab (und die Zeile mit der ungültigen Zusicherung wird angegeben). Ist alles richtig, rechnet das Programm normal.

Hoare-Kalkül zur Verifikation

{ x==a & y==b & a >=0 }

```
while (x > 0)
{ y = y+1; x = x-1;

}
```

{ y== a+b }

Mit den Regeln des Hoare-Kalkül beweist man die Gültigkeit dieser Hoare-Formel und damit die partielle und totale Korrektheit des Programms!

Die Summe von a und b? Ja, wenn a >=0 !

Iteration: Hoare-Regeln

Partielle Korrektheit:

Invariante

$$\frac{\{b\ \&\ I\}\ S\ \{I\}}{\{I\}\ \textbf{while}\ (b)\ \ S\ \{(!b)\ \&\ I\}} \qquad (\text{Iteration}_{partiell})$$

Schritt 1: Finde Invariante

{ x==a & y==b & a >=0}

⇓

{ x+y ==a+b & x >=0}

```
while (x > 0)
{ y = y+1; x = x-1;

}
```

{ x+y ==a+b & x >=0 & !(x>0)}

⇓

{ y== a+b }

Die Summe von a und b?

Welche invariante Beziehung gilt für die Programmvariablen in der Schleife?

x+y ändert sich nicht

x bleibt immer >= 0

Also I := x+y ==a+b & x >=0

Schritt 2: Anwendung Iterationsregel part. Korrektheit

Sei I := x+y ==a+b & x >=0

Dann gilt:

x>0 & x+y ==a+b & x >=0 ⇒ x-1+y+1 ==a+b & x -1>=0

{x-1+y+1 ==a+b & x -1>=0} y=y+1; {x-1+y ==a+b & x-1 >=0}

{x-1+y ==a+b & x-1 >=0} x=x-1; {x+y ==a+b & x >=0}

Daraus folgt mit Abschwg u. seq. Komp.:

{x>0 & I} y=y+1; x=x-1; {I}

Die Blockregel impliziert: {x>0 & I} {y=y+1; x=x-1;} {I}

Daraus folgt mit Iterationsregel: {I} **while**(x>0){y=y+1; x=x-1;} {!(x>0)&I}

Wegen x+y==a+b & x >=0 & !(x>0) ⇒ x == 0 & y==a+b

folgt die partielle Korrektheit des Programms.

Iteration: Hoare-Regeln

Totale Korrektheit:

{b & I} S {I}

{b & I & t == z} S {t < z} //t wird echt kleiner

I ⇒ t >= 0 //t nie negativ

{I} **while** (b) S {(!b) & I} $(\text{Iteration}_{total})$

t – ein Integer-Ausdruck für die Terminierung der while-Schleife

z – eine „logische“ Variable, die nicht in I, b, S oder t vorkommt, also durch S nicht verändert wird.

Schritt 3: Totale Korrektheit

Es gilt: {I & x>0} {y=y+1; x=x-1;} {I}

für I := x+y ==a+b & x >=0

Zu zeigen: Terminierung mittels geeignetem t d.h.

1) {I & x>0 & t==z} {y=y+1; x=x-1;} {t<z}

2) I ⇒ t>=0

Wähle t := x

Dann gilt 2) denn x+y ==a+b & x >=0 ⇒ x >= 0

Es gilt auch 1) denn es gilt

x+y ==a+b & x>=0 & x>0 & x==z ⇒ x-1<z

{x-1<z} {y=y+1; x=x-1;} {x<z}

Beweisskizze

- Eine **Beweisskizze für partielle / totale Korrektheit** ist ein mit Zusicherungen ergänztes Programm S, bei dem **jede** Teilanweisung R von S mit Vor- und Nachbedingung der Form {P} R {Q} annotiert ist, sodaß {P} R {Q} im Hoare-Kalkül ableitbar ist. Folgen zwei Zusicherungen {P} {Q} hintereinander, muss gelten P ⇒ Q

- Insbesondere sind annotiert

while - Anweisung	mit {P} (while(b) {b&P} S {P}) {!b&P}
if- Anweisung	mit {P}(if(b) {b&P} S1 {Q} **else** {!b&P} S2{Q}){Q}
Block	mit {P} {{P} S {Q}} {Q}
„Doppelformel“	mit {P} ⇒ {P1} S {Q1} ⇒ {Q} so daß P ⇒ P1, Q1 ⇒ Q

⇒ kann auch weggelassen werden

Beweisskizzen

Beispiele:

1. Zuweisung

```
{true}
  ⇓
{x * x >=0}
x = x * x;
{x >=0}
```

also ist auch

```
{true}
x = x * x;
{x >=0}
```

Beweisskizze

Beweisskizzen

2. if-then-else-Regel

```
{x == A}
(if (x>=0)
   {x>=0 & x == A}
       ⇓
   {x == A & x ==|A|}
    y = x;
   {x == A& y ==|A|}
else
   {x < 0 & x == A}
       ⇓
   {x == A & -x ==|A|}
    y = -x;
   {x == A & y ==|A|})
{x == A & y ==|A|}
```


Beweisskizzen

3. .... Beispiel

```
{true}
  ⇓
{1 <= 11}
int n = 1;
{n <= 11}
int end = 10;
{n <= end+1}  //Invariante
(while (n <= end)  {n <= end & n <= end+1}
{       {n <= end & n <= end+1}
                  ⇓
        {n+1 <= end+1}
             n = n+1;
        {n <= end+1}
}       {n <= end+1}   )
{!(n <= end) & n <= end+1}
              ⇓
      {n == end+1}
```

Beweis der partiellen Korrektheit von

```
{true}
int n=1; int end=10;
while (n <= end) n = n+1;
{n == end+1}
```


Beweisskizzen

3. .... Beispiel

```
    { a >=0}
  int x = a, y = b;
 { x==a & y==b & a >=0}
        ⇓
 { x+y == a+b & x >=0}
( while (x > 0)   { x>0 & I}
{    { x>0 & I}
        ⇓
 {x-1+y+1 == a+b & x -1>=0}
 y = y+1;
 {x-1+y == a+b & x-1 >=0}
 x = x-1;
 { x+y == a+b & x >=0}
} { x+y == a+b & x >=0}       )
{ x+y == a+b & x >=0 & !(x>0)}  ⇒   { y== a+b }
```

Beweis der partiellen Korrektheit von

```
{a >=0}
int x=a, y=b;
while (x>0) {y=y+1; x=x -1;}
{y == a+b}
```

mit der Invariante
I := x+y == a+b & x >=0

Annotierte Programme

- Ein annotiertes Programm ist ein Programm, das Zusicherungen als Kommentare enthält (aber möglicherweise nicht vollständig annotiert ist).
- Ein annotiertes Programm `{P} S {Q}` heißt partiell korrekt, wenn es
  1. partiell korrekt ist bzgl. `P,Q` und wenn
  2. für jede innere Zusicherung Q1 von S
     Q1 gültig ist in jedem Zustand, der erhalten wird durch (partielle) Ausführung von S bis zu der Stelle von `Q1` mit Start in einem Zustand, in dem P gilt.

Annotierte Summe

```
int a = 23, b = 25;
int x = a, y = b;
assert x==a & y==b;

while (x > 0)
{ y = y+1; x = x -1;

}
 assert y== a+b;
```

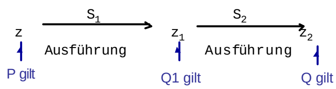
Für a=23 und b=25 gilt, dass das Programm die Summe von a und b berechnet.

Für a= -20 und b=25 gilt dies NICHT!

Annotierte Programme

Beispiel

S ≡ {P} S_1 {Q1} S_2 {Q}



Sei z ein Zustand, in dem P gilt.

Dann müssen Q1 in z_1 und Q in z_2 gelten

Annotierte Programme

Beispiele

- Jede Beweisskizze ist ein korrektes annotiertes Programm
  z.B. `int n = 1; {n==1} n = 2 * n; {n==2}`
- Jede gültige Hoare-Formel {P} S {Q} ist ein partiell korrektes annotiertes Programm
- Jedes Programm S kann als annotiertes Programm der Form {**true**} S {**true**} aufgefaßt werden
- `int n = 1; {n>0} n = 2 * n; {n==2}`

Beispiel: Summe

```
int a = 23, b=25;
int x = a, y = b;
assert x==a & y==b & a >=0;

assert x+y == a+b & x >=0;
( while (x > 0)
{    assert x>0 & x+y ==a+b & x>=0;

assert x-1+y+1 ==a+b & x -1>=0;
y = y+1;

assert x-1+y == a+b & x-1 >=0;
x = x-1;

assert x+y == a+b & x >=0;
}                               )
assert x+y == a+b & x >=0 & !(x>0);  ⇒   assert y== a+b ;
```

Vollständige Annotation von

```
{a >=0}
int x=a, y=b;
while (x>0) {y=y+1; x=x -1;}
{y == a+b}
```

mit der Invariante
I := x+y ==a+b & x >=0

Beispiel: Division x/y und Rest

Sei x ≥ 0, y > 0, I ≡ x == quo * y + rem & rem >=0

Dann ist

```
int quo = 0, rem = x;
{I}   //Invariante
while (rem >= y)
{   rem = rem – y; quo = quo + 1;
}
{x == quo * y + rem  & 0 <= rem < y}
```

Ein total korrektes annotiertes Programm (Beweis Übung)

Realisierung in Java mittels „assert“

Division mit Rest:

```
int x = 1000, y = 37;
int quo = 0, rem = x;
assert x == quo * y + rem & rem >=0;
while (rem >= y)
{ rem = rem-y; quo = quo+1;
  assert x == quo * y + rem & rem >= 0;
}
assert x == quo * y + rem  &  0 <= rem  &  rem < y;
```


Realisierung von Zusicherungen in Java

- Gelten alle Zusicherungen bei der Ausführung des Programms, so terminiert das Programm normal (und nur die print-Anweisungen werden ausgegeben).

  þ Java unterstützt das Testen von Programmen durch die Möglichkeit der dynamischen Überprüfung von Zusicherungen.

! Testen liefert keinen Beweis der Korrektheit, sondern im Fehlerfall den Nachweis von Fehlern!

Zusammenfassung

- Der Hoare-Kalkül dient zur Verifikation der partiellen und totalen Korrektheit (kleiner) Programme.
- Beweisskizzen sind eine übersichtliche Präsentation von formalen Beweisen mit dem Hoare-Kalkül.
- Annotierte Programme unterstützen die Dokumentation von Programmen und können in Java mit der `assert`-Anweisung geschrieben und getestet, aber nicht bewiesen werden.