Freie Universität Berlin

Partielle Korrektheit von Programmen

... Beispiele an der Tafel

Freie Universität Berlin

Beispiel:

Partielle Korrektheit von Programmen

Nehmen wir an, wir möchten beweisen, dass folgendes Programm partiell korrekt ist:

```
# Eingabe
dividend = int(input( 'n=' ))
divisor = int(input( 'd=' ))

result = 0
rest = dividend
while rest >= divisor:
    rest = rest-divisor
    result = result+1

# Ausgabe
print( 'rest=',rest)
print( 'result=',result)
```

Das Programm berechnet die ganzahlige Division mit Rest von zwei positiven Zahlen

Wir werden in unserer Analyse keine Ausgabe berücksichtigen und die Eingabe als vorgegebene Konstanten betrachten.

Vor- und Nachbedingungen können wir mit Hilfe von assert-Anweisungen formulieren.

```
# Eingabe
dividend = int(input( 'n=' ))
divisor = int(input( 'd=' ))

assert dividend>=0 and divisor>0
result = 0
rest = dividend
while rest>=divisor:
        rest = rest-divisor
        result = result+1
assert dividend==(result*divisor+rest) and 0<=rest and rest<divisor
```

Vorbedingung {P}

Div-Programm

Nachbedingung {Q}

Unser Programm ist partiell korrekt, wenn die Programmformel {P} Div-Programm {Q} gültig ist.

{dividend≥0 ∧divisor>0} {P}

```
result = 0
rest = dividend
while rest>=divisor:
        rest = rest-divisor
        result = result+1
```

{dividend==result·divisor+rest ∧0≤rest<divisor} {Q}

Wir fangen rückwärts mit Q an und versuchen, eine Schleifeninvariante für die while-Anweisung zu finden, dann beweisen wir mit Hilfe der Sequenzregel den Rest.

{dividend≥0 ∧divisor>0} {P}

```
result = 0
rest = dividend
while  rest>=divisor:
        rest = rest-divisor
        result = result+1
```

{B}

{dividend==result·divisor+rest ∧0≤rest<divisor} {Q}

Zuerst müssen wir eine Schleife-Invariante finden, sodass folgendes gilt:

INV ∧¬B ⇔ dividend==result·divisor+rest ∧0≤rest<divisor

weil ¬B ≡ rest<divisor gilt,

dann eine mögliche Invariante ist:

INV ≡dividend==result·divisor+rest ∧0≤rest

Partielle Korrektheit von Programmen

jetzt müssen wir für unsere Schleife die while-Regel beweisen

$\{INV \wedge B\}$ Schleifen-Rumpf $\{INV\}$

in unserem Fall:

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest \wedge rest \geq divisor\}$

```
rest = rest-divisor
result = result+1
```

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest\}$

Partielle Korrektheit von Programmen

Wir fangen wieder rückwärts an und versuchen, zuerst eine Vorbedingung P_1 für die letzte Zuweisung zu finden und dann eine Vorbedingung P_2 für die erste Zuweisung zu finden mit Hilfe des Zuweisungsaxioms.

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest \wedge rest \geq divisor\}$

$\{P_2\}$

```
rest = rest-divisor
```

$\{P_1\}$

```
result = result+1
```

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest\}$

Partielle Korrektheit von Programmen

{dividend==result·divisor+rest ∧0≤rest ∧rest≥divisor} $\{P_2\}$

{dividend==(result+1)·divisor+rest-divisor∧0≤rest-divisor} $\{P_2\}$

rest = rest-divisor

{dividend==(result+1)·divisor+rest∧0≤rest } $\{P_1\}$

result = result+1

{dividend==(result)·divisor+rest ∧0≤rest }

{dividend==result·divisor+rest ∧0≤rest ∧rest≥divisor}

Da die Vorbedingung des Schleifenrumpfs das Prädikat P_2 impliziert, haben wir aufgrund der zweiten Konsequenz-Regel die Gültigkeit der Teilformel bewiesen.

{dividend==result·divisor+rest ∧rest≥divisor } {P_2}

rest = rest-divisor

{dividend==(result+1)·divisor+rest ∧0≤rest } {P_1}

result = result+1

{dividend==(result)·divisor+rest ∧0≤rest }

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest \wedge rest \geq divisor\}$

rest = rest-divisor

$\{dividend == (result+1) \cdot divisor + rest \wedge 0 \leq rest\}$

result = result+1

$\{dividend == (result) \cdot divisor + rest \wedge 0 \leq rest\}$

Mit Hilfe der Sequenzregel erhalten wir:

$\{dividend == result \cdot divisor + rest \wedge 0 \leq rest \wedge rest \geq divisor\}$

rest = rest-divisor

result = result+1

$\{dividend == (result) \cdot divisor + rest \wedge 0 \leq rest\}$

Freie Universität Berlin

Partielle Korrektheit von Programmen

Wir haben eine Schleifeninvariante gefunden und damit auch nach der while-Regel die Gültigkeit der Programmformel

{INV}

{dividend==result·divisor+rest ∧ 0≤rest }

```
while rest>=divisor:
    rest = rest-divisor
    result = result+1
```

{dividend==result·divisor+rest ∧ 0≤rest<divisor }

{INV∧¬B}

bewiesen.

Freie Universität Berlin

Partielle Korrektheit von Programmen

Zum vollständigen Beweis der partielle Korrektheit brauchen wir nur noch zu zeigen, dass nach der Ausführung der ersten zwei Zuweisungen aus den Anfangsvorbedingungen die Schleifen-Invariante hergeleitet werden kann.

```
{dividend>=0 ∧ divisor>0}
        result = 0
        rest = dividend
{dividend==result·divisor+rest ∧ 0≤rest }
        while rest>=divisor:
            rest = rest-divisor
            result = result+1
{dividend==result·divisor+rest ∧ 0≤rest<divisor }
```

Freie Universität Berlin

Wieder mit dem Zuweisungsaxiom erhalten wir

{dividend>=0 ∧divisor>0}

result = 0

{dividend==**result**·divisor +dividend∧0≤dividend}

{dividend==result·divisor+**dividend** ∧0≤**dividend**}

rest = dividend

{dividend==result·divisor+rest ∧0≤rest }

while rest>=divisor:

rest = rest-divisor

result = result+1

{dividend==result·divisor+rest ∧0≤rest<divisor }

{dividend>=0 ∧ divisor>0}

{dividend==dividend∧0≤dividend}

result = 0

{dividend==**result**·divisor +dividend∧0≤dividend}

{dividend==result·divisor+**dividend** ∧0≤**dividend**}

rest = dividend

{dividend==result·divisor+rest ∧0≤rest }

while rest>=divisor:

rest = rest-divisor

result = result+1

{dividend==result·divisor+rest ∧0≤rest<divisor }

{dividend>=0 ∧ divisor>0}

{true∧0≤dividend}

{dividend==dividend∧0≤dividend}

result = 0

{dividend==**result**·divisor +dividend∧0≤dividend}

{dividend==result·divisor+**dividend** ∧0≤**dividend**}

rest = dividend

{dividend==result·divisor+rest ∧0≤rest }

while rest>=divisor:

rest = rest-divisor

result = result+1

{dividend==result·divisor+rest ∧0≤rest<divisor }

{dividend>=0 ∧ divisor>0}

{0≤dividend}

{true∧0≤dividend}

{dividend==dividend∧0≤dividend}

result = 0

{dividend==**result**·divisor +dividend∧0≤dividend}

{dividend==result·divisor+**dividend** ∧0≤**dividend**}

rest = dividend

{dividend==result·divisor+rest ∧0≤rest }

while rest>=divisor:

rest = rest-divisor

result = result+1

{dividend==result·divisor+rest ∧0≤rest<divisor }

Freie Universität Berlin

Partielle Korrektheit von Programmen

{dividend>=0 ∧ divisor>0}

⇓

{dividend>=0}

result = 0

rest = dividend

{dividend==result·divisor+rest ∧ 0≤rest }

wir beweisen die Gültigkeit der Teilprogrammformel mit Hilfe der Konsequenzregel

while rest>=divisor:

rest = rest-divisor

result = result+1

{dividend==result·divisor+rest ∧ 0≤rest<divisor }

Freie Universität Berlin

Partielle Korrektheit von Programmen

Zum Schluss verwenden wir die Sequenzregel und beweisen damit unsere Programmformel.

{dividend>=0 ∧ divisor>0}

result = 0
rest = dividend

{dividend==result·divisor+rest ∧ 0≤rest }

while rest>=divisor:
rest = rest-divisor
result = result+1

{dividend==result·divisor+rest ∧ 0≤r

{dividend>=0 ∧ divisor>0}

result = 0
rest = dividend
while rest>=divisor:
rest = rest-divisor
result = result+1

{dividend==result·divisor+rest ∧ 0≤rest<divisor }

Nochmal alle Vor- und Nachbedingungen in Python

```
# Eingabe
dividend = int(input('n='))
divisor = int(input('d='))

assert dividend>=0 and divisor>0
result = 0
assert result*divisor==0 and dividend>=0
rest = dividend
assert dividend==(result*divisor+rest) and rest>=0
while rest>=divisor:
    assert dividend==(result*divisor+rest) and rest>=0 and rest>=divisor
    rest = rest-divisor
    assert dividend==((result+1)*divisor+rest) and 0<=rest
    result = result+1
assert dividend==(result*divisor+rest) and 0<=rest and rest<divisor
# Ausgabe
print('rest=',rest)
print('result=',result)
```

Terminierung

Bis jetzt haben wir nur die partielle Korrektheit bewiesen.

Die totale Korrektheit kann nur bewiesen werden, wenn die Terminierung gezeigt wird.

Das Problem entsteht, wenn wir mindestens eine Schleife haben.

Betrachten wir folgende while-Schleife

```
while B:
    S1
    ...
    S2
```

Wir müssen nachweisen, dass nach endlich vielen Schleifendurchläufen ¬B erfüllt ist.

Terminierung

Eine Funktion τ heißt Terminierungsfunktion der while-Schleife

```
while B:  S
```

wenn sie folgende drei Bedingungen erfüllt:

1) Seien $v_1,\ldots, v_n$ die in B und S vorkommenden Variablen und Konstanten. Dann ist $\tau: Z^n \to Z$ mit $\tau(v_1,\ldots, v_n) \in Z$.

2) Ist vor einem Schleifendurchlauf $\tau(v_1,\ldots, v_n)=t$ erfüllt, dann ist nach dem Schleifendurchlauf $\tau(v_1,\ldots, v_n)<t$ erfüllt.

3) Es existiert ein Wert $t^* \in Z$, so dass vor jedem Schleifendurchlauf $\tau(v_1,\ldots, v_n) \geq t^*$ gilt.

Existiert eine Terminierungsfunktion für eine Schleife, dann terminiert die Schleife.

Freie Universität Berlin

Iterationsregel für totale Korrektheit

$$\frac{\{INV \wedge B \wedge t=z\}\ S\ \{INV \wedge t<z\},\ INV \rightarrow t \geq 0}{\{INV\}\ \text{while}\ B:\ S\ \ \{INV \wedge \neg B\}}$$

Erweiterung der while-Regel mit zusätzlichen Bedingungen über die Laufvariable

Durch {t=z} S {t<z} nimmt t bei jedem Schleifendurchlauf ab

INV→t≥0 stellt sicher, dass eine untere Schranke gibt

Das bedeutet, die Schleife muss irgendwann enden.