

Verifikation



Juni 1996: Erster Start der Ariane 5 Rakete

Gewinn durch Nichtverifikation eines Codestücks: ein paar \$

Verlust durch Selbstzerstörung der Rakete: ca. \$ 1 200 000 000

Weitere gute Gründe für Verifikation:

- Zeitverlust durch Debugging ist oft größer als der für Verifikation
- Möglichkeit des Nachweises der Korrektheit für den Auftraggeber
- sicherere Wiederverwendbarkeit ohne erneutes Nachprüfen
- Fehler in selten benutzten Programmteilen werden selten gefunden

Verschiedene Fehlerarten

- Anforderungsfehler/Spezifikationsfehler
Fehlerhafte, unvollständige, falsche, inkonsistente Anforderungen
- Entwurfsfehler
Falsch verstandene oder inkonsistente Anforderungen
- Implementierungsfehler
falsche Übersetzung von Entwurf ins Programm
- Wiederverwendungsfehler
unterlassene erneute Spezifikation / Test / Verifikation
- Features

Verifikation



Edsger W. Dijkstra:
„A discipline of Programming“, 1976

„Testing can be used to show the presence of bugs, never their absence.“

Erfinder des wp-Kalküls (wp = weakest precondition)

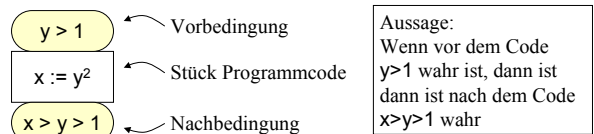
Arten der „Qualitätssicherung“

- **Testen**
 - Testeingabedaten und erwartete Ergebnisse zusammenstellen
 - Programm laufen lassen und Ergebnis mit Erwartung vergleichen
 - Korrektheit kann nicht garantiert werden
 - Anforderungsfehler und Entwurfsfehler können entdeckt werden
- **Verfeinerung**
 - Programm wird Schritt für Schritt gemäß Spezifikation entwickelt, so daß es zu jedem Zeitpunkt in jeder Version korrekt ist
- **Verifikation**
 - mathematischer Beweis, daß (**formale**) Spezifikation erfüllt wird
 - Entdeckung von Anforderungsfehlern nicht möglich

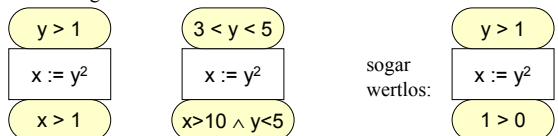
Zusicherungen (Assertions)

- **garantieren** an bestimmten Stellen im Programm bestimmte Eigenschaften
- Sind **prädikatenlogische Aussagen** über die Werte der Programmvariablen an den Stellen im Programm,
- **Vorbedingung** und **Nachbedingung** des gesamten Programs sind also eine prädikatenlogische Spezifikation
- Verfeinerung zieht mit dem Entwickeln des Programms Zusicherungen ein
- Verifikation prüft, ob ein Programm bestimmte Zusicherungen erfüllt, d.h. ob Nachbedingungen aus Vorbedingungen **ableitbar**

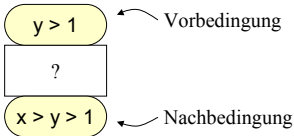
Beispiel für Zusicherungen



Zusicherungen können unterschiedlich stark sein:



Zusicherungspaar = Spezifikation

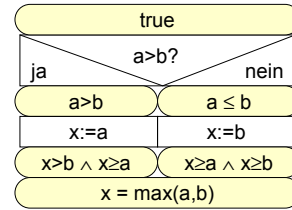


Hierdurch wird ein Programm spezifiziert. (Viele Programme leisten das Gewünschte)

Notation für Zusicherungen

Einige Programmiersprachen bieten Konstrukte wie `assert(x>1)` (\Rightarrow Programmabbruch, wenn nicht $x>1$)

Wir verwenden Struktogramme mit Ovalen:

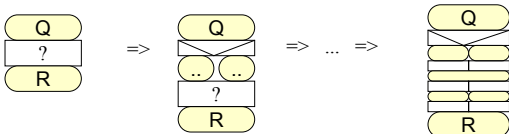


Programmierung mit Verfeinerung

• Hoare-Tripel = $\{Q\} S \{R\}$ (Vorb., Code, Nachbed.)

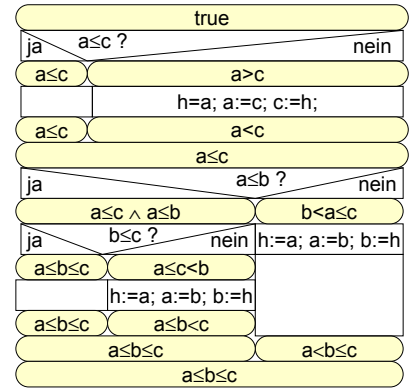
- Verfeinerung:
 - kein S gegeben, nur $\{Q\} . \{R\}$
 - Aufgabe des Programmierers: ein Programm S schreiben

Wenn vor dem Programm die Anfangsbedingung Q erfüllt ist, muß das Programm **terminieren** und nach der Terminierung die **Endebedingung R** erfüllen.



Beispiel:

Sortiere 3 Zahlen



Korrektheit von Programmen

partielle Korrektheit kann man dadurch zeigen, daß die gewünschte Nachbedingung erfüllt wird:

z.B. `x=1`
`while true {y:=x^2}`
`y:=x^2`
 Nach jedem Schleifendurchlauf gilt $y=x^2$, aber das Programm terminiert nicht.

totale Korrektheit verlangt außerdem den Beweis der Terminierung

das ist leider nicht immer möglich (\Rightarrow Halteproblem Info III)

außerdem schwierig: `x=2`
`while (x=Summe von 2 Primzahlen) x+=2`
`x>2`

Korrektheit von Programmen

Ein Programm heißt spezifikationstreu oder total korrekt, wenn

es gibt ein n , und zum Zeitpunkt z_0 gilt P
 und
 zum Zeitpunkt z_n gilt Q und das Programm ist terminiert

Ein Programm heißt partiell korrekt, wenn

zum Zeitpunkt z_0 gilt P
 und
 wenn das Programm zum Zeitpunkt z_n terminieren würde dann würde Q gelten

Unberechenbare Probleme

- Feststellen, ob 2 Programme die gleiche Wirkung haben
- Feststellen, ob ein gegebenes Programm überhaupt terminiert
- Feststellen, ob ein Programm eine Spezifikation erfüllt
- Ein Programm zu einer Spezifikation generieren
- Feststellen, ob 2 Spezifikationen die gleiche Klasse von Programmen festlegen

Diese Aufgaben lassen sich nur in Einzelfällen lösen, sind aber nicht für beliebige Programme und Spezifikationen algorithmisch lösbar.

Verifikation von imperativen Programmen

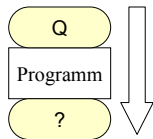
- imperative Programme haben lineare Kontrollstrukturen
- aus Korrektheit von Teilen folgt Korrektheit der Sequenz

Verifikationskalküle

- verwenden **Regelwerke** (rein Syntaktische) zur Ableitung zu verifizierender Aussagen - Regeln sind rein mechanisch anwendbar
- können **korrekt** sein: alles, was abgeleitet werden kann, ist wahr
- können **vollständig** sein: alles, was wahr ist, kann abgeleitet werden
- z.B. Floyd-Hoare-Kalkül (nach Charles Hoare),
odr wp-Kalkül (nach Esdger Dijkstra)

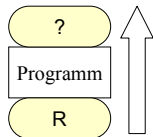
Vorwärts- und Rückwärtsanalyse

Vorwärts:



- gegeben Q und Programm
- bestimme „gute“ Nachbedingung

Rückwärts:



- gegeben R und Programm
- bestimme „gute“ Vorbedingung

Vorwärts- und Rückwärtsanalyse

Vorwärts:

- Beginne am Anfang der Hauptprozedur, steige ab, wo es geht
- Versuche, aus gültiger Vorbedingung eine Nachbedingung zu zeigen
- Kenntnis des Programms kann zur Ableitung der passenden Nachbedingungen benutzt werden
- Aber: nicht zielgerichtet, da man viele Nachbedingungen aus einer Vorbedingung ableiten kann

Rückwärts:

- Beginne am Ende der Hauptprozedur, steige ab, wo es geht
- Versuche aus zu beweisender Nachbedingung Vorbedingung zu finden
- Suche nach der schwächsten Vorbedingung (weakest precondition) die beweist die Nachbedingung „gerade noch“ und wird zur Nachbedingung der vorangehenden Anweisung

wp-Kalkül und Floyd/Hoare-Kalkül

- **Floyd/Hoare-Kalkül** wird vorwärts gerechnet
Problematik: - Nachbedingungen werden immer komplexer
- trickreich: Nur „nötige“ Teile übrig lassen
- **wp-Kalkül** wird rückwärts gerechnet

Schreibweise:

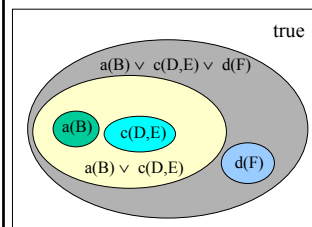
$wp(\text{Programm}, \text{Nachbedingung}) = \text{Vorbedingung}$

z.B. $wp(x=y, x=1) = y=1$

bedeutet: wenn nach Ausführung von $x=y$ gelten soll $x=1$, dann muß vorher **mindestens** $y=1$ gegolten haben

weak, weaker, weakest

Vor- und Nachbedingungen sind Aussagen, die eine Halbordnung bilden mit der Relation \rightarrow („daraus folgt“)



false entspricht leerer Menge

Vereinigung (\cup) entspricht der Disjunktion (oder \vee)

Schnittmenge (\cap) entspricht der Konjunktion (und \wedge)

Ein paar besondere Anweisungen

grundsätzlich gilt für alle P:

$\{P\}$ **leer** $\{P\}$ wenn nichts passiert, ändert sich die Bedingung nicht
 $\text{wp}(\text{leer}, P) = P$

$\{\text{false}\}$ **Fehler** $\{P\}$ Fehler dürfen nicht passieren
 $\text{wp}(\text{Fehler}, P) = \text{false}$

$\{?\}$ **abort** $\{\text{false}\}$ egal welche Vorbedingung, nach Abbruch gilt nichts mehr
 $\text{wp}(\text{abort}, \text{false}) = \text{false}$

Ein paar einfache Rechenregeln

Ausgeschlossenes Wunder:

$$\text{wp}(A, \text{false}) = \text{false}$$

Distributivität der Konjunktion:

$$\text{wp}(A, Q) \wedge \text{wp}(A, R) = \text{wp}(A, Q \wedge R)$$

Distributivität der Disjunktion:

$$\text{wp}(A, Q) \vee \text{wp}(A, R) = \text{wp}(A, Q \vee R)$$

Monotonie:

$$\begin{array}{l} \text{wenn } Q \rightarrow R \\ \text{dann } \text{wp}(A, Q) \rightarrow \text{wp}(A, R) \end{array}$$

wp Zuweisungsregel

$\text{wp}(\text{variable} := \text{ausdruck}, \text{Nachbedingung})$

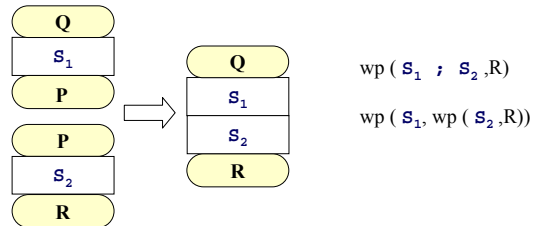
= Nachbedingung [variable / ausdruck]

in der Nachbedingung werden die Variablen durch die Ausdrücke substituiert, man nimmt also sozusagen den Effekt der Anweisung zurück

Zum Beispiel: $\text{wp}(v := v+1, v = m) = (v=m)[v/(v+1)]$
 $= v+1 = m$
 $= v = m-1.$

Sequenzregel allgemein

Zwei Programmstücke S_1 und S_2 können zu einem Programmstück $S_1 ; S_2$ zusammengesetzt werden, wenn die Nachbedingung von S_1 mit der Vorbedingung von S_2 identisch ist.

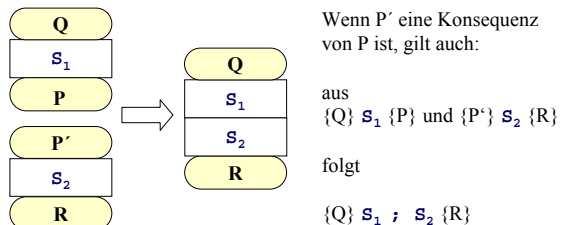
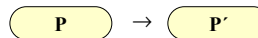


Hintereinanderausführung

Beispiel: Vertauschen der Werte der Variablen x und y:

$\text{wp}(h := x; x := y; y := h, x=X \wedge y=Y)$
 $= \text{wp}(h := x; x := y, \text{wp}(y := h, x=X \wedge y=Y))$
 $= \text{wp}(h := x; x := y, x=X \wedge h=Y)$
 $= \text{wp}(h := x, \text{wp}(x := y, x=X \wedge h=Y))$
 $= \text{wp}(h := x, y=X \wedge h=Y)$
 $= y=X \wedge x=Y$

Sequenzregel und Konsequenzregel



IF-Regel

$$\begin{aligned}
 & \text{wp}(\text{if } b \text{ then } A, Q) \\
 = & (b \rightarrow \text{wp}(A, Q)) \wedge (\neg b \rightarrow Q)
 \end{aligned}$$

$(b \rightarrow \text{wp}(A, Q))$ und $(\neg b \rightarrow Q)$
 wenn b gilt, dann muß als Vorbedingung wp(A, Q) gelten und wenn b nicht gilt, dann muß als Vorbedingung Q gelten

IF-THEN-ELSE-Regel

$$\begin{aligned}
 & \text{wp}(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) \\
 = & (b \rightarrow \text{wp}(S_1, Q)) \wedge (\neg b \rightarrow \text{wp}(S_2, Q))
 \end{aligned}$$

$(b \rightarrow \text{wp}(S_1, Q))$ wenn b gilt, dann muß als Vorbedingung wp(S₁, Q) gelten,
 $(\neg b \rightarrow \text{wp}(S_2, Q))$ wenn b nicht gilt, dann muß als Vorbedingung wp(S₂, Q) gelten,
 also das Mindeste, was gelten muß, damit nach S₁ Q gilt also das Mindeste, was gelten muß, damit nach S₂ Q gilt

IF-THEN-ELSE-Regel

Alternativ können wir schreiben

$$\begin{aligned}
 & \text{wp}(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) \\
 = & (b \wedge \text{wp}(S_1, Q)) \vee (\neg b \wedge \text{wp}(S_2, Q))
 \end{aligned}$$

es muß b und die Vorbedingung wp(S₁, Q) gelten,

oder aber $\neg b$ und die Vorbedingung wp(S₂, Q)

Das ist klar, weil immer gilt:

$$(A \wedge B) \vee (\neg A \wedge C) \Leftrightarrow (A \rightarrow B) \vee (\neg A \rightarrow C)$$

Allgemeine Fallunterscheidung: case-Regel

$$\begin{aligned}
 & \text{wp}(\text{switch } b \\
 & \quad \text{case } b_1 : S_1 ; \\
 & \quad \text{case } b_2 : S_2 ; \\
 & \quad \dots \\
 & \quad \text{case } b_n : S_n ; \\
 & \quad \text{else } S_{n+1}, Q)
 \end{aligned}$$

(Annahme: Semantik der switch-Anweisung wie in Java / C / C++)

$$= (\forall i: (b_i \rightarrow \text{wp}(S_i; \dots; S_n, Q))) \wedge ((\neg \exists i: b_i) \rightarrow \text{wp}(S_{n+1}, Q))$$

wenn irgend ein b_i gilt, dann muß als Vorbedingung wp(S₁; ...; S_n, Q) gelten, wobei
 $S_i; \dots; S_{k1}; \dots; S_{km}; \dots; S_n = S_i; \dots; S_{k1}; \dots; S_{kq}$
 wenn S_{kq} = break

Schleifenumwandlung

Bemerkung: Der Einfachheit halber läßt sich jede Schleife als while-Schleife schreiben:

do A while b	A while b do A
do A until b	A while $\neg b$ do A
for (A,b,C) D	A while b do { D ; C }

Verifikation von Schleifen

Betrachten wir die Schleife **while B do A**

und schreiben sie um zu

if B { A ; if B { A ; if B { A ; if B { ... } } } }

dann ist wp(**while B do A**, Q) =

$$\begin{aligned}
 & \neg B \rightarrow Q \wedge \\
 & B \rightarrow \text{wp}(A, \neg B \rightarrow Q \wedge \\
 & \quad B \rightarrow \text{wp}(A, \neg B \rightarrow Q \wedge \\
 & \quad \quad B \rightarrow \text{wp}(A, \dots)))
 \end{aligned}$$

nicht besonders handlich

Beispiel für Schleifenverifikation

wp (**while** ($x > 0$) **do** $x = x - 1$, $x = 0$) =

$$\begin{aligned} \neg(x > 0) &\rightarrow x = 0 \wedge \\ (x > 0) &\rightarrow [\neg(x - 1 > 0) \rightarrow x - 1 = 0 \wedge \\ &\quad (x - 1 > 0) \rightarrow [\neg(x - 1 - 1 > 0) \rightarrow x - 1 - 1 = 0 \wedge \\ &\quad \quad (x - 1 - 1 > 0) \rightarrow [\dots]]]] \end{aligned}$$

oder vereinfacht: wp (**while** ($x > 0$) **do** $x = x - 1$, $x = 0$) =

$$\begin{aligned} x \leq 0 &\rightarrow x = 0 \wedge \\ x > 0 \wedge x \leq 1 &\rightarrow x = 1 \wedge \\ x > 1 \wedge x \leq 2 &\rightarrow x = 2 \wedge \\ x > 2 \wedge x \leq 3 &\rightarrow x = 3 \dots \end{aligned} \quad \text{oder noch einfacher: } x \geq 0$$

Rekurrenente Schleifenregel

$$\begin{aligned} \text{wp}(\text{while } B \text{ do } A, Q) = H(Q) &= \forall k \geq 0: H_k(Q) \\ &= H_0(Q) \wedge H_1(Q) \wedge H_2(Q) \wedge \dots \end{aligned}$$

Die Schleifenzustandsfunktion H ist als Rekurrenz definiert:

$$\begin{aligned} H_0(Q) &= \neg B \rightarrow Q \\ H_1(Q) &= B \rightarrow \text{wp}(\text{if } B \text{ then } A, H_{k-1}(Q)) \end{aligned}$$

Zur Lösungen suchen wir also ein H, für das gilt:

$$\begin{aligned} H &\rightarrow \neg B \rightarrow Q && \text{und} \\ H &\rightarrow B \rightarrow \text{wp}(\text{if } B \text{ then } A, H) \end{aligned}$$

Schleifenregel mit Invariante

wp (**while** B **do** A, Q)

Falls es eine Invariante I gibt, in die sich die Nachbedingung auftrennen läßt:

$$Q = I \wedge \neg B \quad \text{und es gilt}$$

$$I \wedge B \rightarrow \text{wp}(\text{while } B \text{ do } A, I)$$

dann terminiert die Schleife und es gilt

$$I \wedge \text{wp}(\text{while } B \text{ do } A, \text{true}) \rightarrow \text{wp}(\text{while } B \text{ do } A, \underbrace{I \wedge \neg B}_{\text{Nachbedingung}})$$

Beispiel für Rekurrenente Schleifenregel

wp (**while** ($x > 0$) **do** $x = x - 1$, $x = 0$) =

$$H_0(x=0) \wedge H_1(x=0) \wedge H_2(x=0) \wedge \dots$$

$$\text{wobei } H_0(x=0) = \neg(x > 0) \rightarrow x = 0$$

$$\text{und } H_{k+1}(x=0) = (x > 0) \rightarrow \text{wp}(x = x - 1, H_k(x=0))$$

Jetzt suchen wir ein H, das die Rekurrenzrelation erfüllt.

Wie wäre es mit $H = x \geq 0$? Wir prüfen nach:

$$\begin{aligned} x \geq 0 &\rightarrow H_0(x=0) \checkmark \\ \text{und } x \geq 0 &\rightarrow (x > 0) \rightarrow \underbrace{\text{wp}(x = x - 1, x \geq 0)}_{x - 1 \geq 0 \text{ bzw. } x \geq 1} ? \end{aligned}$$

$$\text{also prüfen: } x \geq 0 \rightarrow ((x > 0) \rightarrow x \geq 1) \quad \checkmark \text{ (für } x \in \mathbb{N})$$

Prozeduren

Grundsätzlich ist es möglich,
jedes Programm in eins ohne Prozeduren umzuwandeln:

Einfach an die Aufrufstelle einbetten.

Dabei müssen ggf. Variablen, Parameter, etc. umbennt werden.

=> Verifikation von Prozeduren weniger mittels wp sondern durch Definition eines „Prozedurvertrags“ (=Spezifikation)

Beispielprogramm

Wir wollen verifizieren:

```
{n ≥ 0}
s = 0;
for (i = 1; i <= n; i++)
    s += i;
{s = (n+1)n/2 }
```

1. Umwandlung in
„Normalform“

```
{n ≥ 0}
s = 0;
i = 1;
while (i <= n)
{
    s = s + i;
    i = i + 1;
}
{s = (n+1)n/2 }
```

Eine Beispielverifikation

Wir suchen also

$\text{wp}(s=0; i=1; \text{while}(i \leq n) \{s=s+i; i=i+1\}; s=(n+1)n/2)$

= (Sequenzregel)

$\text{wp}(s=0; i=1; \text{wp}(\text{while}(i \leq n) \{s=s+i; i=i+1\}; s=(n+1)n/2))$

erst mal die Schleife:

$H_0(s=(n+1)n/2) = \neg i \leq n \rightarrow s=(n+1)n/2$

$H_{k+1}(s=(n+1)n/2) = i \leq n \rightarrow \text{wp}(s=s+i; i=i+1; H_k(s=(n+1)n/2))$



Eine Beispielverifikation (Forts.)

Welches H erfüllt die Rekurrenz?

Wie wäre es mit $H = s=(i-1)i/2 \wedge i \leq n+1$

$H_0(s=(n+1)n/2) = \neg i \leq n \rightarrow s=(n+1)n/2$

$H_{k+1}(s=(n+1)n/2) = i \leq n \rightarrow \text{wp}(s=s+i; i=i+1; H_k(s=(n+1)n/2))$

$\text{wp}(s=s+i; i=i+1; H_k(s=(n+1)n/2))$

= $\text{wp}(s=s+i; \text{wp}(i=i+1; H_k(s=(n+1)n/2)))$

$\text{wp}(i=i+1; s=(i-1)i/2 \wedge i \leq n+1) = s=(i+1)i/2 \wedge i \leq n$

$\text{wp}(s=s+i; s=(i+1)i/2 \wedge i \leq n) = s+i=(i+1)i/2 \wedge i \leq n$

gilt nun $H \rightarrow \neg i \leq n \rightarrow s=(n+1)n/2$

und $H \rightarrow i \leq n \rightarrow s+i=(i+1)i/2 \wedge i \leq n$?



Eine Beispielverifikation (Forts.)

wir prüfen nun, ob $H \rightarrow H_0$ und ob $H \rightarrow H_k$

$H \rightarrow H_0$? $s=(i-1)i/2 \wedge i \leq n+1 \rightarrow \neg i \leq n \rightarrow s=(n+1)n/2$

wenn $i \leq n$, haben wir: $\dots \rightarrow (\text{false} \rightarrow \text{true}) \checkmark$

wenn $i = n+1$, haben wir: $s=(n+1)n/2 \rightarrow (\text{true} \rightarrow s=(n+1)n/2) \checkmark$

wenn $i > n+1$, haben wir: $\text{false} \rightarrow \dots \checkmark$

$H \rightarrow H_k$? $s=(i-1)i/2 \wedge i \leq n+1 \rightarrow i \leq n \rightarrow s+i=(i+1)i/2 \wedge (i+k=n)$

wenn $i \leq n$ haben wir: $s=(i-1)i/2 \rightarrow s+i=(i+1)i/2 \checkmark$

wenn $i > n$ haben wir: $\dots \rightarrow \text{true} \vee \text{true} \checkmark$

Jetzt fehlt noch: $\text{wp}(s=0; i=1; s=(i-1)i/2 \wedge i \leq n+1)$



Eine Beispielverifikation (Forts.)

Jetzt fehlt noch: $\text{wp}(s=0; i=1; s=(i-1)i/2 \wedge i \leq n+1)$

= $\text{wp}(s=0; \text{wp}(i=1; s=(i-1)i/2 \wedge i \leq n+1))$

= $\text{wp}(s=0; s=0 \wedge 1 \leq n+1)$

= $0 \leq n$

Also muß vor Beginn des Codestückes mindestens $0 \leq n$ gelten, was in der Spezifikation so beschrieben ist.

Bemerkung: wir haben die **totale** Korrektheit gezeigt.

