

wp-Kalkül

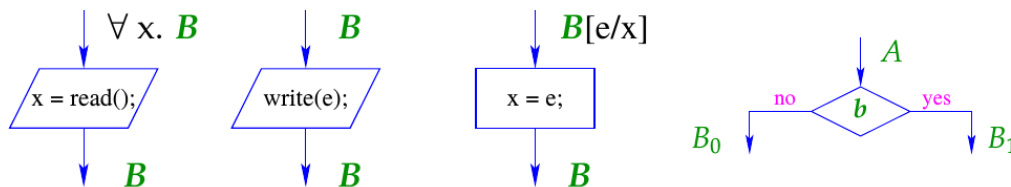
1 Theorie

Das wp-Kalkül dient zur Verifikation der Semantik von Programmen und fällt in den Bereich der axiomatischen Semantik. Es werden Schlussregeln (Axiome) festgelegt, mit welchen sich Aussagen zu Programmpunkten in Abhängigkeit von nachfolgenden Operationen berechnen lassen. Die Verifikation von Programmen ist ein sehr wichtiges Themengebiet der Informatik. Die größte Fehlerquelle der Welt ist der Mensch und somit sollten wir auch Programme, die von Menschen geschrieben sind, mit Skepsis betrachten und sie verifizieren, anstatt ihnen blind zu vertrauen, denn das hat schon zu oft in der Geschichte zu unschönen Ereignissen geführt¹.

Die Idee des wp-Kalküls ist folgende:

1. Annotiere das gewünschte Verhalten des Programmes im Kontrollflussgraphens des Programmes an den geeigneten Stellen (jeweils an den Kanten). Immer, wenn das Programm an den jeweiligen Punkt ankommt, muss die Annotation/Zusicherung erfüllt sein.
2. Beginne vom Stopknoten und berechne durch die vom wp-Kalkül gegebenen Schlussregeln die Vorbedingung zur aktuellen Annotation.
3. Verifiziere, ob die annotierte und die berechnete Annotation zueinander konsistent sind. Was “konsistent” hier bedeutet, erläutern wir noch im Laufe dieses Blattes im Fallbeispiel.
4. Wiederhole diesen Schritt, bis zum Startknoten und überprüfe somit das ganze Programm.

Diese Vorgehensweise ist einfach, solange keine Schleifen im Spiel sind. Diese erschweren, wie wir später sehen werden, das Leben. Wir betrachten nun die Schlussregeln des wp-Kalküls aus der Vorlesung.



$$\begin{aligned}
 \text{WP}[\text{;}] (B) &\equiv B \\
 \text{WP}[\text{x} = \text{e};] (B) &\equiv B[e/x] \\
 \text{WP}[\text{x} = \text{read}();] (B) &\equiv \forall x. B \\
 \text{WP}[\text{write}(\text{e});] (B) &\equiv B
 \end{aligned}
 \qquad
 \begin{aligned}
 \text{WP}[b] (B_0, B_1) &\equiv ((\neg b) \Rightarrow B_0) \wedge (b \Rightarrow B_1) \\
 &\equiv (\neg b \wedge B_0) \vee (b \wedge B_1)
 \end{aligned}$$

Quelle: VL-Folien von Prof. Seidl

B stellt dabei stets einen logischen Ausdruck dar. $B[e/x]$ bedeutet “ersetze jedes Vorkommen von x in B mit e”. Die Regeln sollten einen (hoffentlich) recht intuitiv vorkommen.

¹https://en.wikipedia.org/wiki/List_of_software_bugs

Beispiel 1.1.

- $(i > 41 \wedge j = -1)[k/i] \equiv (k > 41 \wedge j = -1)$.
- $(i > 41 \wedge j = -1)[j/i] \equiv (j > 41 \wedge j = -1) \equiv \text{false}$.
- $0[i/j] = 0$
- $WP[(x+1)^2 = x^2 + 2x + 1](x = \text{read}();) \equiv \forall x. (x+1)^2 = x^2 + 2x + 1$

Wie bereits gesagt, stellen Schleifen ein Problem dar, da diese uns hindern sequentiell vom Stop- zum Startknoten unsere Annotation zu verifizieren. Wir könnten ja theoretische öfter die Schleife durchlaufen und müssen dies bei unserer Verifikation berücksichtigen. Um dies zu tun, müssen wir eine Zusicherung I am Schleifenkopf annotieren, die vor jedem Eintritt der Schleife, als auch vor dem Verlassen des Schleifenabschnitts gelten muss. Die Zusicherung I nennt man dann **Schleifeninvariante**.

Damit nicht genug, mit Schleifen kommt noch ein zweites Problem auf uns zu: Terminierung. Vielleicht kommen wir am Stopknoten unseres Programmes an, sondern verlieren uns in einer sich unaufhaltsam wihrenden Schleife? Wie können wir sicher gehen, dass dies nicht passiert? Wir geben wieder ein Vorgehensweise an:

- Für jede Schleife, füge eine Messgröße r in das Programm ein, die vor dem ersten Antreffen der Schleife initialisiert wird.
- Bei jedem Betreten der Schleife, stelle sicher, dass r größer als eine von dir festgelegte untere Schranke bleibt (meist $r \geq 0$).
- Vor dem Verlassen der Schleife, stelle sicher, dass r strikt kleiner gesetzt wird.

2 Fallbeispiel

2.1 Ente, Ente, Ente, Gauß!

Wir betrachten folgendes Programm

```
1:  $x \leftarrow \text{read}()$ 
2:  $i \leftarrow 0$ 
3:  $s \leftarrow i$ 
4: while  $i \leq x$  do
5:    $s \leftarrow s + i$ 
6:    $i \leftarrow i + 1$ 
7:  $\text{write}(s)$ 
```

und zeichnen den Kontrollflussgraphen (siehe Abbildung 1).

2.1.1 Semantische Analyse: Was berechnet das Programm?

Diejenigen, die sich das Grundlagenblatt über Induktion gut durchgelesen haben bzw. ein gutes Gespür haben, bemerken sofort, was dieses Programm berechnet. Wir vermuten, dass am Ende des Programmes $s = \frac{x*(x+1)}{2}$, falls $x \geq 0$, gilt. Wem diese Formel jetzt merkwürdig vorkommt, der sollte sich kurz genanntes Blatt durchlesen.

Beweis. Wir setzen also zunächst $G := s = \frac{x*(x+1)}{2}$ und berechnen dann F:

$$WP[\text{write}(s);](G) \equiv G =: F$$

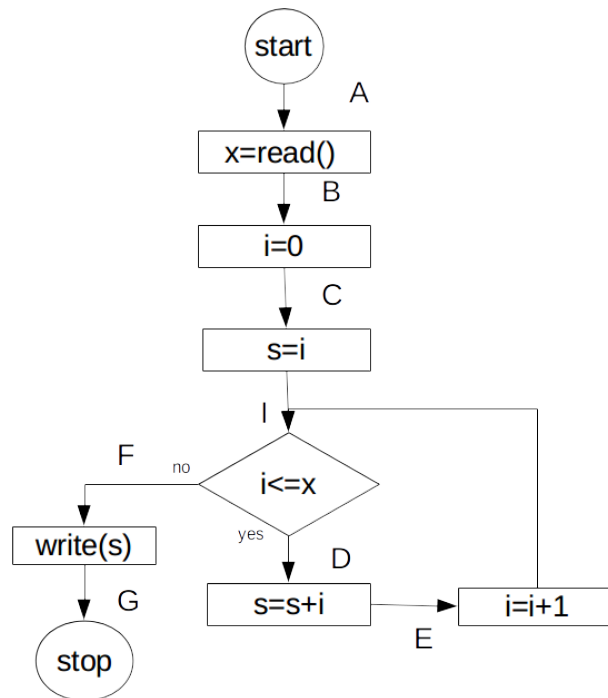


Abbildung 1: Kontrollflussgraph des Algorithmus

Jetzt kommt der schwierige Teil: Wir sind bei der Schleife angelangt und können nun nicht einfach mehr sequentiell unsere Zusicherungen berechnen, sondern müssen eine geeignete Schleifeninvariante I finden. Um dies zu tun, gibt es ein paar Tipps:

- Die Schleifeninvariante hängt im Normalfall irgendwie von der Laufvariable ab.
- Oft hängt die Invariante stark mit der Aussage am “Ausstiegspunkt” (hier E) zusammen und beschreibt diesen in Abhängigkeit der Laufvariable (hier i).
- Es ist besser, eine gültige Aussage zu viel in die Invariante zu packen, als zu wenig (z.B. scheint hier $i \geq 0$ vielleicht zunächst überflüssig; manchmal ist dies aber eine entscheidende Notwendigkeit in der Verifikation).
- Wenn man keine gute Vermutung hat, hilft es oft sich den Programmablauf (d.h. den Verlauf der Variablenwerte) für eine Beispieleingabe aufzuschreiben.

Wir tun so, als hätten wir keine gute Vermutung und schreiben uns den Programmablauf für $x=3$ auf (links strikt nach VL, rechts verkürzt tabellarisch nur am Programmpunkt I):

$\pi = ((A, \{x \mapsto \top, i \mapsto \top, s \mapsto \top\}),$
 $(B, \{x \mapsto \top, i \mapsto \top, s \mapsto \top\}),$
 $(C, \{x \mapsto 3, i \mapsto 0, s \mapsto \top\}),$
 $(I, \{x \mapsto 3, i \mapsto 0, s \mapsto 0\}),$
 $(D, \{x \mapsto 3, i \mapsto 0, s \mapsto 0\}),$
 $(E, \{x \mapsto 3, i \mapsto 0, s \mapsto 0\}),$
 $(I, \{x \mapsto 3, i \mapsto 1, s \mapsto 0\}),$
 $(D, \{x \mapsto 3, i \mapsto 1, s \mapsto 0\}),$
 $(E, \{x \mapsto 3, i \mapsto 1, s \mapsto 1\}),$
 $(I, \{x \mapsto 3, i \mapsto 2, s \mapsto 1\}),$
 $(D, \{x \mapsto 3, i \mapsto 2, s \mapsto 1\}),$
 $(E, \{x \mapsto 3, i \mapsto 2, s \mapsto 3\}),$
 $(I, \{x \mapsto 3, i \mapsto 3, s \mapsto 3\}),$
 $(D, \{x \mapsto 3, i \mapsto 3, s \mapsto 3\}),$
 $(E, \{x \mapsto 3, i \mapsto 3, s \mapsto 6\}),$
 $(I, \{x \mapsto 3, i \mapsto 4, s \mapsto 6\}),$
 $(F, \{x \mapsto 3, i \mapsto 4, s \mapsto 6\}),$
 $(G, \{x \mapsto 3, i \mapsto 4, s \mapsto 6\}))$

#Iteration	x	i	s
0	3	0	0
1	3	1	0
2	3	2	1
3	3	3	3
4	3	4	6

Vor allem die Zustände am Punkt I sind für uns interessant. Wir erkennen, dass zum Punkt I stets $s = \sum_{k=0}^{i-1} k$, also auch $s = 0.5 * (i - 1) * (i)$ gilt (Bemerkung: $\sum_{k=0}^{-1} k = 0$, da 0 das neutrale Element der Addition ist).

Mit dieser Erkenntnis und der Berücksichtigung der weiteren Tipps, setzen wir

$$I := s = \frac{(i-1) * i}{2} \wedge 0 \leq i$$

Es gilt jedoch nicht $0 \leq x$, denn der Benutzer könnte eine negative Zahl eingeben. Nun gilt es, unsere Invariante zu verifizieren. Hierfür beginnen wir von I und laufen rückwärts einmal die ganze Schleife entlang.

$$\begin{aligned}
WP[i = i + 1; \mathbb{I}(I)] &\equiv s = \frac{i * (i + 1)}{2} \wedge 0 \leq i + 1 \\
&\equiv s = \frac{i * (i + 1)}{2} \wedge -1 \leq i \\
&\Leftarrow s = \frac{i * (i + 1)}{2} \wedge 0 \leq i =: E
\end{aligned}$$

Hier haben wir im letzten Schritt unsere Zusicherung verstärkt, d.h. strikter gemacht. Dies kann manchmal hilfreich sein, um Terme zu vereinfachen. Wir haben es hier gemacht, da wir uns sicher sind, dass der Fall $i = -1$ nie auftritt und werfen daher diese Information weg. Wir können unsere Annahmen immer strikter machen als die berechnete Vorbedingung. Wir müssen dabei immer sicher gehen, dass die Implikation gültig ist, d.h. in unserem Fall muss immer, wenn E erfüllt ist, auch die ursprünglich berechnete Vorbedingung erfüllt sein (vgl. Blatt "Grundlagen der Logik"). Wenn unsere Annotation die berechnete Zusicherung impliziert, nennen wir die Annotation **konsistent**.

Wir rechnen weiter

$$\begin{aligned}
WP[s = s + i; \mathbb{I}(E) &\equiv s + i = \frac{i * (i + 1)}{2} \wedge 0 \leq i \\
(\text{Gaußsche Summenformel, } 0 \leq i) &\equiv s = \frac{(i - 1) * i}{2} \wedge 0 \leq i =: D \\
WP[i \leq x; \mathbb{I}(F, D) &\equiv (i > x \wedge s = \frac{x * (x + 1)}{2}) \vee (s = \frac{(i - 1) * i}{2} \wedge 0 \leq i) \\
&\Leftarrow (i = x + 1 \wedge s = \frac{x * (x + 1)}{2}) \vee (s = \frac{(i - 1) * i}{2} \wedge 0 \leq i) \\
&\equiv (i = x + 1 \wedge s = \frac{(i - 1) * i}{2}) \vee (s = \frac{(i - 1) * i}{2} \wedge 0 \leq i) \\
&\equiv s = \frac{(i - 1) * i}{2} \wedge (i = x + 1 \vee 0 \leq i) \\
&\Leftarrow s = \frac{(i - 1) * i}{2} \wedge 0 \leq i \equiv I
\end{aligned}$$

Unsere Schleife ist also lokal konsistent annotiert! Bemerke, wie wir schrittweise die beiden Oder-Bedingungen auf eine gemeinsame Form gebracht haben, um dann mit Distributivität die Gemeinsamkeit auszuklammern und so schrittweise sich der Invariante I zu nähern. Im vorletzten Schritt haben wir unsere Aussage verstärkt, indem wir eine Und-Klausel hinzugefügt haben und im letzten Schritt haben wir sie verstärkt, indem wir eine Oder-Klausel entfernt haben. Jetzt sind wir fast fertig, wir müssen nur noch sequentiell zum Start zurückrechnen.

$$\begin{aligned}
WP[s = i; \mathbb{I}(I) &\equiv i = \frac{(i - 1) * i}{2} \wedge 0 \leq i =: C \\
WP[i = 0; \mathbb{I}(C) &\equiv 0 = \frac{(0 - 1) * 0}{2} \wedge 0 \leq 0 \equiv \text{true} =: B \\
WP[x = \text{read}(); \mathbb{I}(B) &\equiv \forall x. \text{true} \equiv \text{true} =: A
\end{aligned}$$

□

Wir haben somit bewiesen, dass alle Annotationen lokal konsistent sind. Insbesondere gilt $A \equiv \text{true}$, d.h. wir müssen keine Bedingung zum Start des Programmes stellen. Aber halt: Für $x < 0$ sollten die Zusicherungen am Endknoten doch nicht erfüllt sein?! Nicht ganz, die Zusicherungen gelten nur, wenn der Programmpunkt auch wirklich erreicht wird. Im Fall von $x < 0$ erreichen wir aber den Stopknoten nie, da das Programm offensichtlich nicht terminiert. Alle erreichbaren Annotation sind allerdings auch für $x < 0$ konsistent (das haben wir schließlich bewiesen).

2.1.2 Lieber ein Ende mit Schrecken als ein Schrecken ohne Ende

Wir sind also nicht ganz zufrieden mit unserem ersten Beweis und wollen uns zusätzlich vergewissern, in welchen Fällen das Programm auch wirklich terminiert und dann die Zusicherung am Ende des Programmes erfüllt. Wir vermuten, dass dies für alle $x \geq 0$ der Fall ist.

Beweis. Zunächst fügen wir die eine Variable r in unser Programm ein, die wir für den Terminierungsbeweis benötigen. Was das Programm berechnet, ist uns für den Terminierungsbeweis egal, das haben wir schließlich schon bewiesen. Uns ist nur wichtig, dass die Programmpunkte F und G erreichbar sind und setzen deshalb möglichst einfach $F := \text{true} =: G$. Achtung: $F := \text{false} =: G$ wäre falsch. Wird an einem Punkt false annotiert und dessen Konsistenz bewiesen, muss der Punkt unerreichbar sein.

Als Schleifeninvariante setzen wir

$$I := r = x - i \wedge 0 \leq x \wedge i \leq x + 1$$

Die Größe von x spielt nun also eine Rolle, die Größe von s ist für die Terminierung hingegen irrelevant.

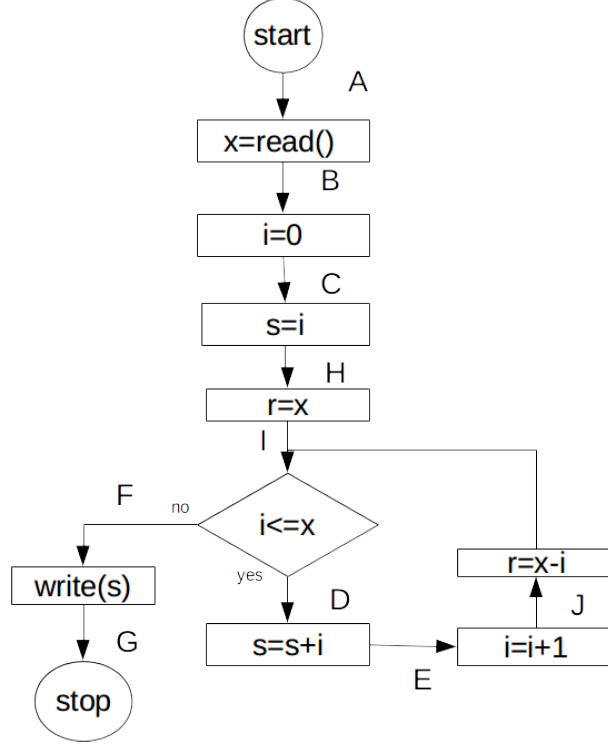


Abbildung 2: Modifizierter Kontrollflussgraph für Terminierungsbeweis

Wir überprüfen die Konsistenz

$$\begin{aligned}
 WP[r = x - i; \llbracket I \rrbracket &\equiv x - i = x - i \wedge 0 \leq x \wedge i \leq x + 1 \\
 &\equiv 0 \leq x \wedge 0 \leq i \leq x + 1 \\
 &\Leftarrow r = x - i + 1 \wedge 0 \leq x \wedge 0 \leq i \leq x + 1 =: J
 \end{aligned}$$

Die letzte Implikation war notwendig, damit wir zusichern, dass r strikt kleiner wird (d.h. $J \Rightarrow r > x - i$).

$$\begin{aligned}
 WP[i = i + 1; \llbracket J \rrbracket &\equiv r = x - (i + 1) + 1 \wedge 0 \leq x \wedge 0 \leq i + 1 \leq x + 1 \\
 &\equiv r = x - i \wedge 0 \leq x \wedge -1 \leq i \leq x \\
 &\Leftarrow r = x - i \wedge 0 \leq x \wedge 0 \leq i \leq x =: E
 \end{aligned}$$

$$WP[s = s + i; \llbracket E \rrbracket \equiv E =: D$$

Insbesondere gilt $D \Rightarrow r \geq 0$, somit haben wir eine untere Grenze für r bei Betreten der Schleife. Zusammen mit J folgt, dass die Schleife nur endlich oft durchlaufen werden kann, vorausgesetzt, unsere Schleifeninvariante ist lokal konsistent. Wir überprüfen den letzten Schritt der Schleife

$$\begin{aligned}
 WP[i \leq x; \llbracket F, E \rrbracket &\equiv (i > x) \vee (r = x - i \wedge 0 \leq x \wedge 0 \leq i \leq x) \\
 &\Leftarrow (i = x + 1 \wedge 0 \leq x \wedge r = x - i) \vee (r = x - i \wedge 0 \leq x \wedge 0 \leq i \leq x) \\
 &\equiv 0 \leq x \wedge r = x - i \wedge (i = x + 1 \vee 0 \leq i \leq x) \\
 &\equiv r = x - i \wedge 0 \leq x \wedge 0 \leq i \leq x + 1 \equiv I
 \end{aligned}$$

Die Schleife ist also lokal konsistent annotiert, wir bewegen uns Richtung Startknoten

$$\begin{aligned}
 WP[r = x;](I) &\equiv x - i = x - i \wedge 0 \leq x \wedge 0 \leq i \leq x + 1 \\
 &\equiv 0 \leq x \wedge 0 \leq i \leq i + 1 =: H \\
 WP[s = i;](H) &\equiv H =: C \\
 WP[i = 0;](C) &\equiv 0 \leq x =: B \\
 WP[x = read();](B) &\equiv \forall x. 0 \leq x =: A
 \end{aligned}$$

Das Programm terminiert also, falls $0 \leq x$ gilt. □

3 Übungen

Semester 2016/17

- Blatt 2: Aufgabe 2, 3, 6, 7
- Blatt 3: Aufgabe 1, 2, 3, (4)
- Blatt 4: Aufgabe 1, 2, 4, (5)

Semester 2015/16

- Blatt 4: Aufgabe 1, 2

Semester 2008/09

- Blatt 2: Aufgabe 4

Von Kevin (Lösungen am Ende des Blatts)

1. Ordnen Sie die folgenden Zusicherungen mittels Implikationen und Äquivalenzen.

- $i = a$
- $i = a \wedge b \neq c \wedge i - 2 * a = -9$
- $i = 9 \wedge i = a \wedge c \neq b$
- $b = 42 \wedge c = 33 \wedge a = b - c \wedge i = 9$
- $i = a \vee (a = 9 \wedge c > b)$
- $i = a \vee a > a$

2. Gegeben folgendes Programm

```

1: while  $x \leq 0$  do
2: |    $x \leftarrow x * x + 2 * x$ 

```

Welche Startzusicherungen sind hinreichend für die Zusicherung $x > 1$ am Ende des Programms?

- (a) $x = 0$
- (b) $x > 0$
- (c) *false*
- (d) $x < -2$
- (e) $x - 1 \geq 1$
- (f) *true*
- (g) $x > x * x \wedge x \in [2, 100]$
- (h) $x \leq 0$

Lösungen:

1. $(b = 42 \wedge c = 33 \wedge a = b - c \wedge i = 9) \Rightarrow (i = 9 \wedge i = a \wedge c \neq b) \equiv (i = a \wedge b \neq c \wedge i - 2 * a = -9)$
 $\Rightarrow (i = a) \equiv (i = a \vee a > a) \Rightarrow (i = a \vee (a = 9 \wedge c > b))$
2. $(a), (c), (d), (e), (g), (h)$