Script **generated by TTT**

Title: Seidl: Info2 (21.10.2016)

Date: Fri Oct 21 08:37:31 CEST 2016

Duration: 80:25 min

Pages: 42

Informatik 2

Wintersemester 2016/17

Helmut Seidl

Institut für Informatik
TU München

Vorlesungsdaten

Modul: IN0003 mit SWS (2+2), 5 ECTS

Zeiten: Freitag 8:30 – 10:00 Uhr (Hörsaal MW 0001)

Webseite: `http://www2.in.tum.de/hp/Main?nid=329`

Voraussetzung: IN0001 – Einführung in die Informatik 1
IN0015 – Diskrete Strukturen

Prüfung:

Klausur: Mo. 20.02.2017
Wiederholung: Fr. 07.04.2017

Tweedback

8uc

Webseite: `tum.twbk.de`

Übungen

- 2 SWS Tutorübungen
- 23 Gruppen (geplant)
- Anmeldung zur Vorlesung Ãijber TUMonline
  https://campus.tum.de/
- Anmeldung zu den Übungsgruppen bis **heute, 21.10.2016 18:00 Uhr** über Matchingsystem:
  https://matching.in.tum.de/m/30v5xsj-info2
- Übungsleitung (info2@in.tum.de)
  Julian Kranz, Helmut Seidl
- Webseite:
  https://www.moodle.tum.de/course/view.php?id=28866
- Forum: Piazza

Übung (Forts.)

- Die Bearbeitung der Übungen ist freiwillig, aber empfehlenswert.
- Es gibt sowohl theoretische Aufgaben, wie Programmieraufgaben.
- Für jedes Übungsblatt gibt es Punkte.
- Für 2/3 der Gesamtpunktzahl gibt es einen Notenbonus auf die erfolgreich bestandene Klausur (oder Wiederholungsklausur).
- Die Hausaufgaben sind **selbstständig** anzufertigen! Nach Plagiaten wird automatisiert und manuell gesucht.

0 Allgemeines

Inhalt dieser Vorlesung

- Korrektheit von Programmen;
- Funktionales Programmieren mit OCaml

1 Korrektheit von Programmen

- Programmierer machen Fehler !?
- Programmierfehler können teuer sein, z.B. wenn eine Rakete explodiert, ein firmenwichtiges System für Stunden ausfällt ...
- In einigen Systemen dürfen keine Fehler vorkommen, z.B. Steuerungssoftware für Flugzeuge, Signalanlagen für Züge, Airbags in Autos ...

Problem

Wie können wir sicherstellen, dass ein Programm das richtige tut?

Ansätze

- Sorgfältiges Vorgehen bei der Software-Entwicklung;
- Systematisches Testen

  $\Longrightarrow$ formales Vorgehensmodell (Software Engineering)
- Beweis der Korrektheit

  $\Longrightarrow$ Verifikation

Ansätze

- Sorgfältiges Vorgehen bei der Software-Entwicklung;
- Systematisches Testen

  $\Longrightarrow$ formales Vorgehensmodell (Software Engineering)
- Beweis der Korrektheit

  $\Longrightarrow$ Verifikation

Hilfsmittel: Zusicherungen

Beispiel

```
public class GGT {
   public static void main (String[] args) {
   int x, y, a, b;
   a = read(); b = read();
   x = a; y = b;
   while (x != y)
      if (x > y) x = x - y;
      else       y = y - x;

   assert(x != y);

   write(x);
   }  // Ende der Definition von main();
}     // Ende der Definition der Klasse GGT;
```


Kommentare

- Die statische Methode `assert()` erwartet ein Boolesches Argument.
- Bei normaler Programm-Ausführung wird jeder Aufruf `assert(e);` ignoriert !?
- Starten wir Java mit der Option: –ea (enable assertions), werden die `assert`-Aufrufe ausgewertet:

  $\Rightarrow$ Liefert ein Argument-Ausdruck `true`, fährt die Programm-Ausführung fort.

  $\Rightarrow$ Liefert ein Argument-Ausdruck `false`, wird ein Fehler `AssertionError` geworfen.

Achtung

Der Laufzeit-Test soll eine Eigenschaft des Programm-Zustands bei Erreichen eines Programm-Punkts überprüfen.

Der Test sollte keineswegs den Programm-Zustand verändern !!!

Sonst zeigt das beobachtete System ein anderes Verhalten als das unbeobachtete ???

Achtung

Der Laufzeit-Test soll eine Eigenschaft des Programm-Zustands bei Erreichen eines Programm-Punkts überprüfen.

Der Test sollte keineswegs den Programm-Zustand verändern !!!

Sonst zeigt das beobachtete System ein anderes Verhalten als das unbeobachtete ???

Tipp

Um Eigenschaften komplizierterer Datenstrukturen zu überprüfen, empfiehlt es sich, getrennt Inspector-Klassen anzulegen, deren Objekte eine Datenstruktur störungsfrei besichtigen können !

Problem

- Es gibt i.a. sehr viele Programm-Ausführungen ...
- Einhalten der Zusicherungen kann das Java-Laufzeit-System immer nur für eine Program-Ausführung überprüfen.

$\Longrightarrow$

Wir benötigen eine generelle Methode, um das Einhalten einer Zusicherung zu garantieren ...

1.1 Verifikation von Programmen

Robert W Floyd, Stanford U. (1936 – 2001)

Vereinfachung

Wir betrachten erst mal nur MiniJava:

- nur eine Methode, nämlich main;
- nur int Variablen;
- nur if und while.

Vereinfachung

Wir betrachten erst mal nur MiniJava:

- nur eine Methode, nämlich main;
- nur int Variablen;
- nur if und while.

Idee

- Wir schreiben eine Zusicherung an jeden Programmpunkt !
- Wir argumentieren, dass lokal an jedem Programmpunkt, dass die Zusicherungen eingehalten werden ...

Vereinfachung

Wir betrachten erst mal nur MiniJava:

- nur eine Methode, nämlich main;
- nur int Variablen;
- nur if und while.

Idee

- Wir schreiben eine Zusicherung an jeden Programmpunkt !
- Wir argumentieren, dass lokal an jedem Programmpunkt, dass die Zusicherungen eingehalten werden ...

17

Vereinfachung

Wir betrachten erst mal nur MiniJava:

- nur eine Methode, nämlich main;
- nur int Variablen;
- nur if und while.

Idee

- Wir schreiben eine Formel an jeden Programmpunkt !
- Wir beweisen, dass lokal an jedem Programmpunkt, dass die Zusicherungen eingehalten werden $\Longrightarrow$ Logik

Exkurs: Logik

Aussagen: "Alle Menschen sind sterblich",
"Sokrates ist ein Mensch", "Sokrates ist sterblich"

Exkurs: Logik

Aussagen: "Alle Menschen sind sterblich",
"Sokrates ist ein Mensch", "Sokrates ist sterblich"

$\boxed{\forall x.\,\text{Mensch}(x) \Rightarrow \text{sterblich}(x)}$

$\text{Mensch}(\text{Sokrates}),\ \ \text{sterblich}(\text{Sokrates})$

Exkurs: Logik

Aussagen: "Alle Menschen sind sterblich",
"Sokrates ist ein Mensch", "Sokrates ist sterblich"

$\forall x.\,\text{Mensch}(x) \Rightarrow \text{sterblich}(x)$

$\text{Mensch}(\text{Sokrates}),\ \ \text{sterblich}(\text{Sokrates})$

Schließen: Falls $\forall x.\,P(x)$ gilt, dann auch $P(a)$ für ein konkretes a !
Falls $A \Rightarrow B$ und A gilt, dann muss auch B gelten !

Exkurs: Logik

Aussagen: "Alle Menschen sind sterblich",
"Sokrates ist ein Mensch", "Sokrates ist sterblich"

$\forall x.\,\text{Mensch}(x) \Rightarrow \text{sterblich}(x)$

$\text{Mensch}(\text{Sokrates}),\ \ \text{sterblich}(\text{Sokrates})$

Schließen: Falls $\forall x.\,P(x)$ gilt, dann auch $P(a)$ für ein konkretes a !
Falls $A \Rightarrow B$ und A gilt, dann muss auch B gelten !

Tautologien: $A \vee \neg A$

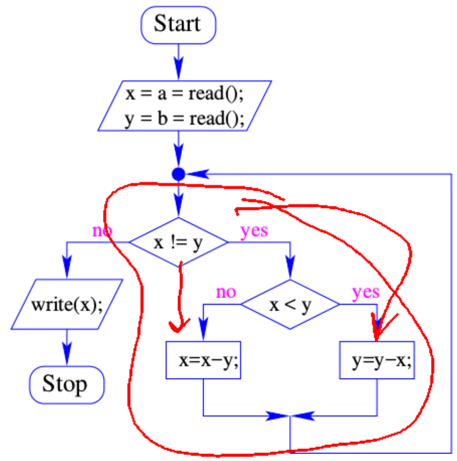
$\forall x \in \mathbb{Z}.\,\boxed{x < 0 \ \vee\ x = 0 \ \vee\ x > 0}$

Exkurs: Logik (Forts.)

Gesetze:

$$
\begin{aligned}
\neg\neg A &\equiv A \\
A \wedge A &\equiv A \\
\neg(A \vee B) &\equiv \neg A \wedge \neg B \\
A \wedge (B \vee C) &\equiv (A \wedge B) \vee (A \wedge C) \\
A \vee (B \wedge C) &\equiv (A \vee B) \wedge (A \vee C) \\
A \vee (B \wedge A) &\equiv A \\
A \wedge (B \vee A) &\equiv A
\end{aligned}
$$

Unser Beispiel



Diskussion

- Die Programmpunkte entsprechen den Kanten im Kontrollfluss-Diagramm !
- Wir benötigen eine Zusicherung pro Kante ...

Hintergrund

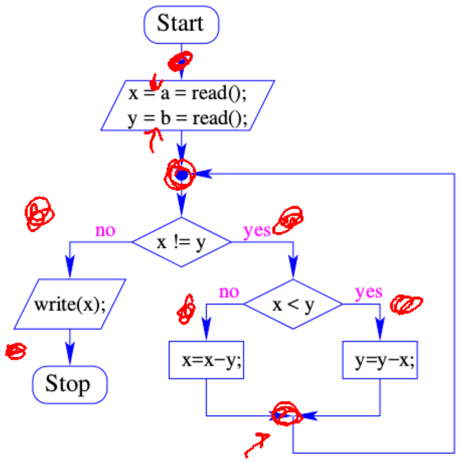
$d \mid x$ gilt genau dann wenn $x = d \cdot z$ für eine ganze Zahl z.

Für ganze Zahlen x, y sei $ggT(x, y) = 0$, falls $x = y = 0$ und andernfalls die größte ganze Zahl d, die x und y teilt.

Dann gelten unter anderem die folgenden Gesetze:

$$
\begin{aligned}
ggT(x,0) &= |x| \\
ggT(x,x) &= |x| \\
ggT(x,y) &= ggT(x, y-x) \\
ggT(x,y) &= ggT(x-y, y)
\end{aligned}
$$

Unser Beispiel



Diskussion

- Die Programmpunkte entsprechen den Kanten im Kontrollfluss-Diagramm !
- Wir benötigen eine Zusicherung pro Kante ...

Hintergrund

$d \mid x$ gilt genau dann wenn $x = d \cdot z$ für eine ganze Zahl z.

Für ganze Zahlen x, y sei $ggT(x, y) = 0$, falls $x = y = 0$ und andernfalls die größte ganze Zahl d, die x und y teilt.

Dann gelten unter anderem die folgenden Gesetze:

$$\begin{aligned}
ggT(x,0) &= |x| \\
ggT(x,x) &= |x| \\
ggT(x,y) &= ggT(x,y-x) \\
ggT(x,y) &= ggT(x-y,y)
\end{aligned}$$

Idee für das Beispiel

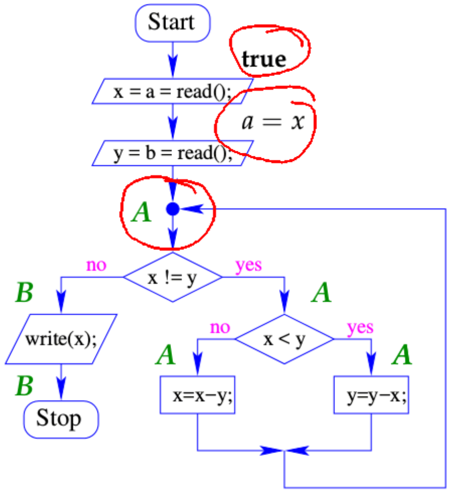
- Am Anfang gilt nix.
- Nach `a=read(); x=a;` gilt $a = x$.
- Vor Betreten und während der Schleife soll gelten:

$$A \quad \equiv \quad ggT(a,b) = ggT(x,y)$$

- Am Programm-Ende soll gelten:

$$B \quad \equiv \quad A \wedge x = y$$

Unser Beispiel



Idee für das Beispiel

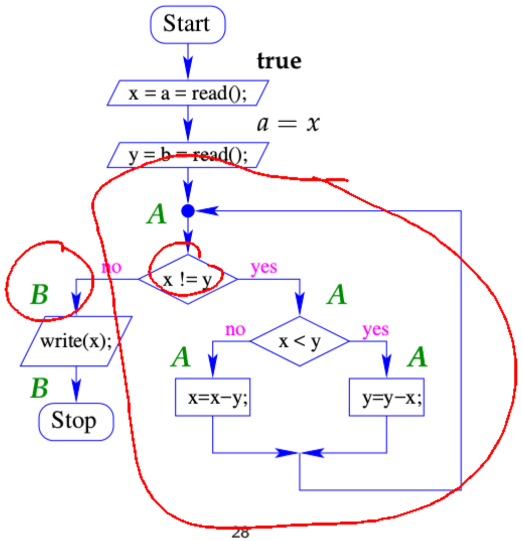
- Am Anfang gilt nix.
- Nach `a=read(); x=a;` gilt $a = x$.
- Vor Betreten und während der Schleife soll gelten:

$$A \quad \equiv \quad ggT(a, b) = ggT(x, y)$$

- Am Programm-Ende soll gelten:

$$B \quad \equiv \quad A \wedge x = y$$

Unser Beispiel



Idee für das Beispiel

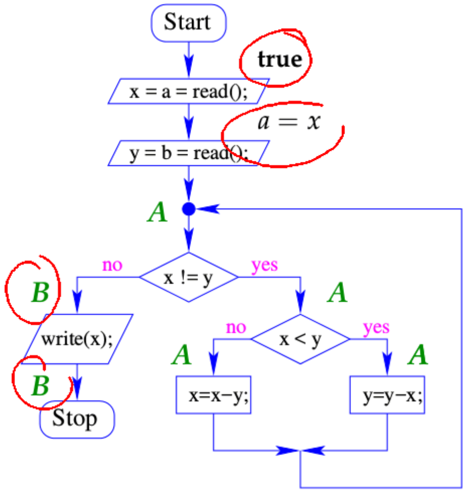
- Am Anfang gilt nix.
- Nach `a=read(); x=a;` gilt $a = x$.
- Vor Betreten und während der Schleife soll gelten:

$$A \quad \equiv \quad ggT(a, b) = ggT(x, y)$$

- Am Programm-Ende soll gelten:

$$B \quad \equiv \quad A \wedge x = y$$

Unser Beispiel



Frage

Wie beweisen wir, dass Zusicherungen lokal zusammen passen?

Teilproblem 1: Zuweisungen

Betrachte z.B. die Zuweisung: `x = y+z;`

Damit nach der Zuweisung gilt: $x > 0$, // Nachbedingung

muss vor der Zuweisung gelten: $y + z > 0$. // Vorbedingung

Allgemeines Prinzip

- Jede Anweisung transformiert eine Nachbedingung B in eine minimale Anforderung, die vor Ausführung erfüllt sein muss, damit B nach der Ausführung gilt.

Allgemeines Prinzip

- Jede Anweisung transformiert eine Nachbedingung B in eine minimale Anforderung, die vor Ausführung erfüllt sein muss, damit B nach der Ausführung gilt.
- Im Falle einer Zuweisung `x = e;` ist diese schwächste Vorbedingung (engl.: weakest precondition) gegeben durch

$$\mathbf{WP}[\![\texttt{x = e;}]\!]\,(B) \quad\equiv\quad B[e/x]$$

Das heißt: wir substituieren einfach in B überall x durch e !!!

Allgemeines Prinzip

- Jede Anweisung transformiert eine Nachbedingung B in eine minimale Anforderung, die vor Ausführung erfüllt sein muss, damit B nach der Ausführung gilt.
- Im Falle einer Zuweisung `x = e;` ist diese schwächste Vorbedingung (engl.: weakest precondition) gegeben durch

$$\mathbf{WP}[\![\texttt{x = e;}]\!]\ (B) \quad \equiv \quad B[e/x]$$

Das heißt: wir substituieren einfach in B überall x durch e !!!

Allgemeines Prinzip

- Jede Anweisung transformiert eine Nachbedingung B in eine minimale Anforderung, die vor Ausführung erfüllt sein muss, damit B nach der Ausführung gilt.
- Im Falle einer Zuweisung `x = e;` ist diese schwächste Vorbedingung (engl.: weakest precondition) gegeben durch

$$\mathbf{WP}[\![\texttt{x = e;}]\!]\ (B) \quad \equiv \quad B[e/x]$$

Das heißt: wir substituieren einfach in B überall x durch e !!!

- Eine beliebige Vorbedingung A für eine Anweisung s ist gültig, sofern

$$A \Rightarrow \mathbf{WP}[\![s]\!]\ (B)$$

// A impliziert die schwächste Vorbedingung für B.

Beispiel

Zuweisung:	`x = x-y;`
Nachbedingung:	$x > 0$
schwächste Vorbedingung:	$x - y > 0$
stärkere Vorbedingung:	$x - y > 2$
noch stärkere Vorbedingung:	$x - y = 3$