Script generated by TTT

Title: Seidl: Info2 (16.10.2015)

Date: Fri Oct 16 08:32:22 CEST 2015

Duration: 76:53 min

Pages: 34

Informatik 2

Wintersemester 2015/16

Helmut Seidl

Institut für Informatik
TU München

Informatik 2

Wintersemester 2015/16

Helmut Seidl

Institut für Informatik
TU München

1

0 Allgemeines

Inhalt dieser Vorlesung:

- Korrektheit von Programmen;
- Funktionales Programmieren mit OCaml

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

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?

3

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:

  ⇒ Liefert ein Argument-Ausdruck `true`, fährt die Programm-Ausführung fort.

  ⇒ 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 ...

Vereinfachung:

Wir betrachten erst mal nur MiniJava:

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

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 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"

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

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"

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

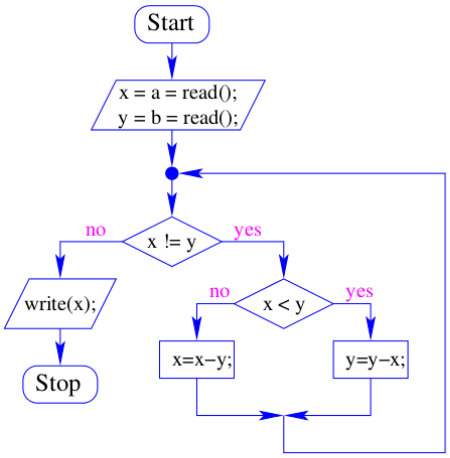
Schließen: Falls $\forall x.\, P(x)$ gilt, dann auch $P(a)$ für ein konkretes a !
Falls $A \Rightarrow B$ und A gilt, dann muss auch B gelten !

Tautologien: $A \vee \neg A$
$\forall x \in \mathbb{Z}.\, x < 0 \ \vee \ x = 0 \ \vee \ x > 0$

Exkurs: Logik (Forts.)

Gesetze:
$\neg\neg A \equiv A$
$A \wedge A \equiv A$
$\neg(A \vee B) \equiv \neg A \wedge \neg B$
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
$A \vee (B \wedge A) \equiv A$
$A \wedge (B \vee A) \equiv A$

Unser Beispiel:



Diskussion:

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

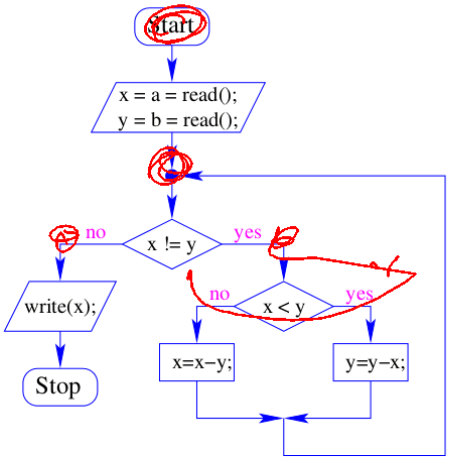
Hintergrund:

$d \mid x$ 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:

Unser Beispiel:



20

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.

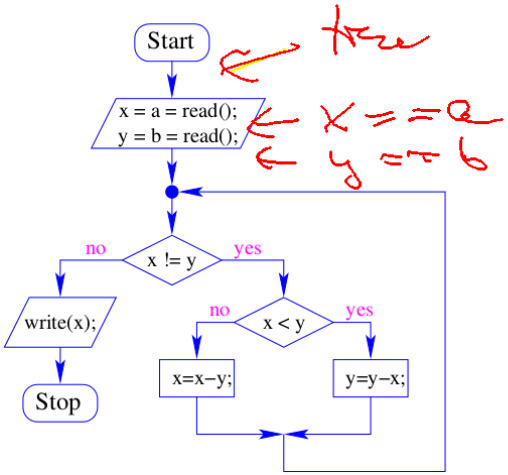
Dann gelten unter anderem die folgenden Gesetze:

21

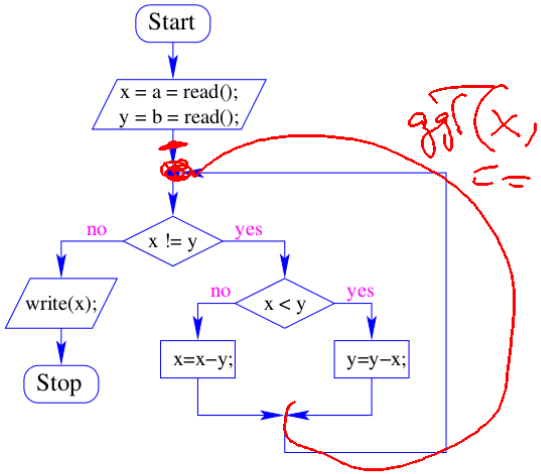
$$
\begin{aligned}
ggT(x,0) &= |x| \\
ggT(x,x) &= |x| \\
ggT(x,y) &= ggT(x,y-x) \\
ggT(x,y) &= ggT(x-y,y)
\end{aligned}
$$

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

Unser Beispiel:



Unser Beispiel:



Idee für das Beispiel:

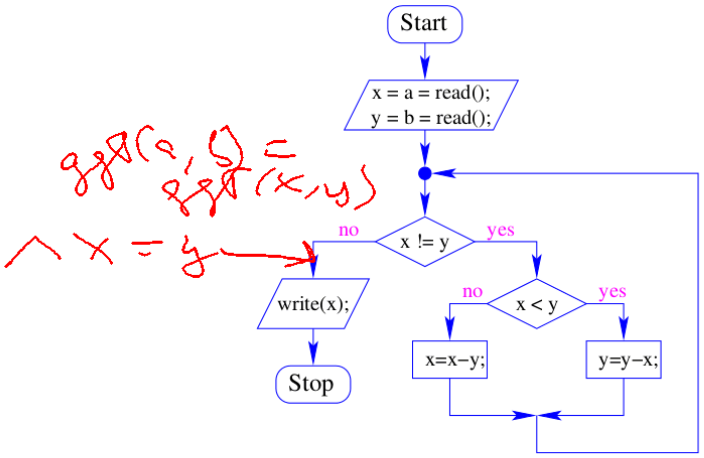
- Am Anfang gilt nix.
- Nach `a=read(); x=a;` gilt $a = x$.
- Vor Betreten und während der Schleife soll gelten:

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

- 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:

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

23