



Technische Universität Berlin

Software and Embedded Systems Engineering Group

Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin



Softwaretechnik und Programmierparadigmen WiSe 2023/2024

Prof. Dr. Sabine Glesner
Simon Schwan
Julian Klein

Übungsblatt 11

Beispiellösung

Aufgabe 1: Terminierung

Beweist die *totale* Korrektheit folgender partiell korrekter Programme. Gibt es einen Algorithmus, der für jedes mögliche Programm eine passende Terminierungsfunktion findet?

a) `max(int a, int b):`

```
{true}
if a > b then
  m := a
else
  m := b
fi
{m ≥ a ∧ m ≥ b ∧ (m = a ∨ m = b)}
```



b) `trinumbr(int n):`

```
{n ≥ 0}
s := 0; i := 0;
while i < n do
  {i < n ∧ s = ∑j=0i j ∧ i ≤ n}
  i := i + 1;
  s := s + i
  {s = ∑j=0i j ∧ i ≤ n}
od
{s = ∑j=0n j}
```



Regel (5) $\{B \wedge I\}$

Regel (5) $\{I\}$

c) `rest(int x, int y):`

```

{ $x \geq 0$ }
q := 0;
r := x;
while r >= y do
    { $r \geq y \wedge x = q * y + r \wedge r \geq 0$ }
    r := r - y;
    q := q + 1
    { $x = q * y + r \wedge r \geq 0$ }
od;
{ $x = q * y + r \wedge r \geq 0 \wedge r < y$ }

```

Regel (5) $\{B \wedge I\}$

Regel (5) $\{I\}$

Referenz: Hoare Kalkül

- (1) Skip-Axiom: $\{P\} \text{ skip } \{P\}$
- (2) Zuweisungsaxiom: $\{P[x \leftarrow E]\} x := E \{P\}$
- (3) Sequenzregel:

$$\frac{\{P\} S_1 \{R\} \quad \{R\} S_2 \{Q\}}{\{P\} S_1; S_2 \{Q\}}$$

- (4) if-then-else-Regel:

$$\frac{\{B \wedge P\} S_1 \{Q\} \quad \{\neg B \wedge P\} S_2 \{Q\}}{\{P\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{Q\}}$$

- (5) while-Regel:

$$\frac{\{B \wedge I\} S \{I\}}{\{I\} \text{ while } B \text{ do } S \text{ od } \{\neg B \wedge I\}}$$

- (6) Konsequenzregel:

$$\frac{\{P \Rightarrow P'\} \quad \{P'\} S \{Q'\} \quad \{Q' \Rightarrow Q\}}{\{P\} S \{Q\}}$$

- (7) Terminierung:

$$\frac{\{B \wedge I \wedge (t = m)\} S \{I \wedge (t < m)\}, \quad B \wedge I \Rightarrow t \geq 0}{\{I\} \text{ while } B \text{ do } S \text{ od } \{\neg B \wedge I\}}$$

Vorgehen: finde Terminierungsfunktion $t \mapsto \mathbb{N}$, sodass

1. $B \wedge I \Rightarrow t \geq 0$ und
2. $\{B \wedge I \wedge (t = m)\} S \{I \wedge (t < m)\}$ gilt.

Lösung:

a) Terminiert natürlich, weil keine Schleife.

b) Terminierung: $t = n - i$

1. $\{i < n \wedge s = \sum_{j=0}^i j \wedge i \leq n\} \Rightarrow n - i \geq 0$ Regel (7) $B \wedge I \Rightarrow n - i \geq 0$
2. while $i < n$ do
 $\{\dots \wedge (n - i = m)\}$ Regel (7) $\{B \wedge I \wedge (t = m)\}$
 $\Rightarrow \{\dots \wedge (n - i - 1 < m)\}$ Regel (6)
 $\Rightarrow \{\dots \wedge (n - (i + 1) < m)\}$ Regel (6) (2) $\{P[i \leftarrow i + 1]\}$
 $i := i + 1;$
 $\{\dots \wedge (n - i < m)\}$ Regel (3) (2) $\{P[s \leftarrow s + i]\}$
 $s := s + i;$
 $\{\dots \wedge (n - i < m)\}$ Regel (7) $\{I \wedge (t < m)\}$
od

c) Terminierung: $t = r - y$

1. $\{r \geq y \wedge x = q * y + r \wedge r \geq 0\} \Rightarrow r - y \geq 0$ Regel (7) $B \wedge I \Rightarrow r - y \geq 0$
2. while $r \geq y$ do
 $\{\dots \wedge (r - y = m)\}$ Regel (7) $\{B \wedge I \wedge (t = m)\}$
 $\nRightarrow \{\dots \wedge (r - 2y < m)\}$ Regel (6), terminiert nur wenn $y > 0$
 $\Rightarrow \{\dots \wedge (r - y - y < m)\}$ Regel (6) (2) $\{P[r \leftarrow r - y]\}$
 $r := r - y;$
 $\{\dots \wedge (r - y < m)\}$ Regel (3) (2) $\{P[q \leftarrow q + 1]\}$
 $q := q + 1$
 $\{\dots \wedge (r - y < m)\}$ Regel (7) $\{I \wedge (t < m)\}$
od;

Wenn es einen generellen Algorithmus gäbe, könnte dieser generell die Terminierung von Programmen entscheiden, was das Halteproblem lösen würde. Daher kann es keinen solchen Algorithmus geben. Die Semi-Entscheidbarkeit des Halteproblems zeigt sich auch hier: wenn *eine* Terminierungsfunktion gefunden wird, dann terminiert das Programm sicher. Andernfalls müsste *für alle* möglichen Kandidaten gezeigt werden, dass diese nicht die Terminierung zeigen können. Da es unendlich viele mögliche Kandidaten gibt, kann dieser Teil nicht entschieden werden (der Algorithmus terminiert nicht).

Aufgabe 2: Nicht-funktionale Anforderungen

Die folgende Anforderungsbeschreibung wurde euch für die Entwicklung eines Programms 

geliefert.

Eine Autowerkstatt möchte die Abfertigung ihrer Aufträge komfortabel mit einer Software verwalten. Dazu können Mitarbeitende im System Kunden und Kundinnen anlegen und ihnen Fahrzeuge zuordnen. Für neue Kunden und Kundinnen werden Name, Telefonnummer und Rechnungsadresse gespeichert und die Fahrzeuge werden mit Kennzeichen und Typ registriert.

Ein Auftrag kann entweder eine Inspektion, ein Reifenwechsel oder eine Reparatur sein. Einem neuen Auftrag wird ein Preis, ein Fahrzeug und automatisch ein Datumsstempel zugewiesen. Eine Reparatur erhält außerdem eine genaue Tätigkeitsbeschreibung. Ein Auftrag kann von Mitarbeitenden als beendet markiert werden. In diesem Fall wird der Kunde bzw. die Kundin automatisch vom System benachrichtigt. Außerdem wird für den Auftrag vermerkt, welche/r Mitarbeiter:in ihn beendet hat.

Um Missbrauch vorzubeugen, müssen sich Mitarbeitende am Browser mit ID und Passwort sicher anmelden. Ein/e Administrator:in kann Mitarbeitende anlegen und entfernen.

- a) Was ist der Unterschied zwischen funktionalen und nicht-funktionalen Eigenschaften?
- b) Überlegt, welche nicht-funktionalen Anforderungen aus dem Text oben hervorgehen.
- c) Welche Eigenschaften sind für dieses System wahrscheinlich auch noch wichtig?

Lösung:

- a) Funktionale Anforderungen beschreiben, was das System bei bestimmten Eingaben (ggf. nicht) tun soll.
Nicht-funktionale Anforderungen beschreiben, wie (im Sinne von Eigenschaften) das System bestimmte Dinge machen sollte. Oft treffen nicht-funktionale Eigenschaften nicht nur einzelne Module, sondern betreffen das gesamte Systeme.
- b) Folgende kann man finden:
 - Produktanforderung - Usability: Abfertigung soll komfortabel sein.
 - Produktanforderung - Security: Anmeldung soll sicher sein.

Aufgabe 3: Requirements Engineering

- a) Ermittelt die im Text von Aufgabe 2 enthaltenen Use-Cases.
- b) Diskutiert, welche Anforderungen Kunden und Kundinnen wahrscheinlich zusätzlich an das System haben werden bzw. welche der Beschreibungen unklar sind.
- c) Erstellt aus zwei Use-Cases User Stories. Sind Vorteile der User Stories ersichtlich?



Lösung: Folgende Use-Cases gibt es:

- Kunde/Kundin anlegen
- Fahrzeug erstellen (und zuordnen)
- Auftrag anlegen (implizit)
- Auftrag beenden
- Als Mitarbeiter:in anmelden
- Mitarbeitende registrieren
- Mitarbeitende entfernen

Und die hier fehlen wahrscheinlich:

- Kunde/Kundin aktualisieren/deaktivieren
- Fahrzeug löschen (bzw. deaktivieren)
- Auftrag stornieren
- Passwort ändern
- Offene Aufträge auflisten

Etwas weiter gehend:


- Rechnung erstellen
- Bestellung ändern

Mögliche User Stories:

- Als Mitarbeiter:in möchte ich neue Kunden und Kundinnen im System anlegen können und ihnen Fahrzeuge zuordnen.
- Als Administrator:in möchte ich Mitarbeitende anlegen, verwalten und entfernen.

Vorteile User Stories: Aus Sicht der Anwender:innen (was brauchen Benutzer:innen vs. was sollte das System liefern), mehr Informationen im Text möglich im Gegensatz zu Schlüsselwort, Anwender:in bekannt; Informelle Struktur ermöglicht freie Darstellung relevanter Informationen.

Aufgabe 4: Strukturierte Anforderungsspezifikation

Modelliert einige der in Aufgabe 3 beschriebenen Anwendungsfälle (Use-Cases) in Form von strukturierten Spezifikationen. Überlegt euch sinnvolle Attribute zur Strukturierung. 

Lösung:

Beispielanwendungsfall: Auftrag anlegen.

Funktion Auftrag anlegen

Beschreibung Ein neuer Auftrag vom richtigen Typ wird im System angelegt. Diesem Auftrag ist ein Fahrzeug zugeordnet.

Input Auftrag (Reparatur, Inspektion, Reifenwechsel), Kunde/Kundin, Fahrzeug/Kennzeichen, Preis, Beschreibung

Output Erfolgs-/Misserfolgs-Nachricht

Aktion Auftrag im System anlegen.

Vorbedingung Kunde/Kundin existiert, Fahrzeug mit Kennzeichen existiert und gehört diesem Kunde/dieser Kundin.

Nachbedingung Auftrag existiert und ist dem Fahrzeug zugewiesen.

Beispielanwendungsfall: Mitarbeiter:in entfernen

Funktion Mitarbeiter:in entfernen

Beschreibung Ein/e Mitarbeiter:in wird aus dem System entfernt

Input Mitarbeiter:in-ID

Output Erfolgs-/Misserfolgs-Nachricht

Aktion Löschen von Mitarbeiter:in im System

Vorbedingung Mitarbeiter:in mit der ID existiert

Nachbedingung Mitarbeiter:in mit der ID existiert nicht

Für die anderen Anwendungsfälle analog.