

# Abschnitt 11: Korrektheit von imperativen Programmen

## 11. Korrektheit von imperativen Programmen

### 11.1 Einführung

### 11.2 Testen der Korrektheit in Java



# Überblick

## 11. Korrektheit von imperativen Programmen

### 11.1 Einführung

### 11.2 Testen der Korrektheit in Java

**LMU**

# Korrektheit von Algorithmen

- ▶ Algorithmen und Programme sollten das tun, wofür sie entwickelt wurden, d.h. sie sollten *korrekt* sein.
- ▶ Problem: wie kann man nachweisen, dass ein Algorithmus bzw. ein Programm korrekt ist?
- ▶ Schon bei relativ kleinen Beispielen kann man oft nicht mehr „mit dem Auge“ entscheiden, ob ein Algorithmus/Programm korrekt ist.
- ▶ Im Folgenden werden wir uns auf Java-Programme konzentrieren.
- ▶ Wir werden aber allgemeine Konzepte besprechen, die auch für andere Programmiersprachen und auch ganz abstrakt für die Algorithmenentwicklung (beispielsweise mittels Pseudo-Code) gelten.

# Korrektheit von Algorithmen

Beispiel: folgendes Java-Programm soll das Quadrat einer Zahl  $a \in \mathbb{N}$  berechnen.

```
public static int quadrat(int a) {  
    int y;  
    int z;  
  
    // Anfang der Berechnung  
    y = 0;  
    z = 0;  
    while (y != a) {  
        z = z + 2*y + 1;  
        y = y + 1;  
    }  
    // Ende der Berechnung  
  
    return z;  
}
```

# Korrektheit von Algorithmen

- ▶ Bemerkung: Als Java-Programm macht der Algorithmus wenig Sinn, aber als Programm für einen Mikroprozessor, für den die allgemeine Multiplikation viel aufwendiger ist als die Addition und Multiplikation mit 2, sehr wohl!
- ▶ Es ist nicht offensichtlich, dass für alle `int`-Werte  $a$  das Programm wirklich den Wert  $a^2$  berechnet.
- ▶ Tatsächlich stimmt dies nur für alle  $a \geq 0$ .
- ▶ Wie können wir die Korrektheit dieses Java-Programms nachweisen?



# Korrektheit von Algorithmen

- ▶ Zur Erinnerung: in der funktionalen Programmierung sind Algorithmen Funktionen, die, auf die Eingabewerte angewendet, spezielle Werte zurückliefern.
- ▶ Der Beweis, dass eine Funktion eine gewisse Eigenschaft hat, ist meist (relativ) einfach zu führen (z.B. durch vollständige Induktion, wenn die Funktion rekursiv definiert ist).

# Korrektheit von Algorithmen

- ▶ Bei imperativen Programmen ist dies i.d.R. deutlich schwieriger.
  - ▶ Wir haben keinen direkten funktionalen Zusammenhang zwischen Eingabewerten und Ausgabewerten.
  - ▶ Das Resultat kommt vielmehr durch Abarbeiten verschiedener Berechnungsschritte zustande.
  - ▶ Diese Berechnungsschritte können Seiteneffekte haben.
- ▶ Der Beweis der Korrektheit imperativer Programme ist daher nicht ganz so einfach.

# Korrektheit von Algorithmen

- ▶ Zunächst sollte festgelegt werden, wie man die Aussagen, die man für ein Programm machen will, formuliert: wie formuliert man z.B., dass die Methode `quadrat` das Quadrat des Eingabewerts berechnet?
- ▶ Dazu kann man eine eigene formale Sprache definieren (ist in diesem Rahmen aber nicht nötig).
- ▶ Wir verwenden Boolesche Java Ausdrücke, um Aussagen zu machen, z.B. `0 <= a` oder `z == a*a`.
- ▶ Eine Aussage ist dann wahr (erfüllt), wenn der entsprechende Ausdruck zu `true` ausgewertet wird, ansonsten natürlich falsch (wir erinnern uns vage an unser Zustandsmodell ...).



# Korrektheit von Algorithmen

- ▶ Zusätzlich verwenden wir als gängige Abkürzung für  $\neg a \vee b$  die Notation  $a \Rightarrow b$ , wobei  $a$  und  $b$  beliebige Boolesche Ausdrücke sein können.
- ▶  $a \Rightarrow b$  drückt die logische Implikation „wenn  $a$ , dann  $b$ “ aus.
- ▶ Damit hat der Ausdruck  $a \Rightarrow b$  den Wert **true** gdw.  $b$  den Wert **true** oder  $a$  den Wert **false** hat.

# Korrektheit von Algorithmen

- ▶ Welche Art von Aussagen über ein Programm  $p$  will man nun beweisen?
- ▶ Typischerweise Aussagen der Art:  
„Wenn für die Eingabewerte des Programms  $p$  die Bedingung  $PRE$  gilt, dann gilt nach Ausführung von  $p$  für die Ausgabewerte immer die Bedingung  $POST$ “.

- ▶ Dies schreibt man meist:

$$(PRE) \ p \ (POST)$$

- ▶  $PRE$  heißt *Vorbedingung (Precondition)*.
- ▶  $POST$  heißt *Nachbedingung (Postcondition)*.



# Korrektheit von Algorithmen

► Beispiel:

Für das Programm `p = quadrat(int a)` wären entsprechende Vor- und Nachbedingungen:

$$(0 \leq a) \quad p \quad (z == a * a)$$

# Korrektheit von Algorithmen

Wir unterscheiden zwei Arten von Korrektheit von Programmen:

▶ **Partielle Korrektheit:**

Partielle Korrektheit eines Programms  $p$  heißt: falls  $p$  auf Eingabewerte, die der Vorbedingung  $PRE$  genügen, angewendet wird, und falls es terminiert, dann muss anschließend die Nachbedingung  $POST$  gelten.

▶ **Totale Korrektheit:**

Totale Korrektheit eines Programms  $p$  heißt: falls  $p$  auf Eingabewerte, die der Vorbedingung  $PRE$  genügen, angewendet wird, dann terminiert es und danach muss die Nachbedingung  $POST$  gelten.

▶ **Beobachtung:** Totale Korrektheit = Partielle Korrektheit + Terminierung



# Korrektheit von Algorithmen

- ▶ Wie kann man nun die Korrektheit imperativer Programme überprüfen?
- ▶ Testen: Man kann z.B. für alle möglichen Eingabewerte testen, ob das Programm die richtigen Ergebniswerte liefert.
- ▶ Offensichtlich geht dies im Allgemeinen nicht für alle möglichen Eingabewerte (primitive Datentypen haben zwar keinen unendlichen Wertebereich, die vollständige Aufzählung z.B. aller `int`-Werte ist allerdings wenig praktikabel insb. dann, wenn es mehrere Eingabeparameter gibt).
- ▶ Testen kann dennoch sehr wichtig sein: in der Entwicklungsphase verwendet man z.B. Tests zum Debugging.
- ▶ Zum formalen Beweis der Korrektheit imperativer Programme benötigt man stattdessen spezielle Logik-Kalküle (z.B. den Hoare-Kalkül).

# Definitionsbereiche von Programmen

- ▶ Frage: Was passiert eigentlich, wenn Eingabewerte, die der Vorbedingung `PRE` nicht genügen, verarbeitet werden?
- ▶ Zunächst wiederum nur funktionale Algorithmen:
  - ▶ Der Algorithmus stellt eine Funktion dar.
  - ▶ Dies ist eine Abbildung  $f : D \rightarrow B$  vom Definitionsbereich  $D$  auf/in den Bildbereich  $B$ .
  - ▶ Korrektheit bedeutet informell: für Eingaben aus  $D$  erhält man durch Auswertung von  $f$  die entsprechend gewünschten Werte aus  $B$ .
  - ▶ Für Eingabewerte, die nicht Element von  $D$  sind, erhält man durch Auswertung von  $f$  (je nach Definition von  $f$ ) beliebige Werte aus  $B$  oder gar keine Werte oder  $f$  terminiert nicht, etc.



# Definitionsbereiche von Programmen

- ▶ Bei imperativen Algorithmen ist die Situation ähnlich:
  - ▶ Eine Methode ist ebenfalls eine Abbildung  $f : D \rightarrow B$  vom Definitionsbereich  $D$  auf/in den Bildbereich  $B$  mit möglichen Seiteneffekten.
  - ▶ Die „Vorschrift“, wie diese Abbildung berechnet wird, ist nun als Folge von Anweisungen notiert, statt als mathematische Funktion.
  - ▶ Analog bedeutet Korrektheit damit informell: für Eingaben aus  $D$  erhält man durch Abarbeiten der Anweisungen die entsprechend gewünschten Werte aus  $B$  bzw. (z.B. wenn  $B = \emptyset$ ) entsprechend gewünschte Seiteneffekte.
  - ▶ Für Eingabewerte, die nicht Element von  $D$  sind, erhält man durch Abarbeiten der Anweisungen (je nach Anweisungen) wiederum beliebige Werte oder gar keine Werte oder der Algorithmus terminiert nicht, etc.
  - ▶ Zusätzlich können bei imperativen Algorithmen für Eingabewerte, die nicht Element von  $D$  sind, (möglicherweise unerwünschte) Nebeneffekte auftreten.
- ▶ Eingabewerte aus falschen Domänen sind also zu vermeiden.

# Definitionsbereiche von Programmen

▶ Beispiel: Java-Methode `quadrat`

- ▶ Definitionsbereich  $D = \mathbf{int}$
- ▶ Bildbereich  $B = \mathbf{int}$

▶ Frage:

Was passiert, wenn ich die Methode `quadrat` statt mit einem `int`-Wert mit einem `double`-Wert aufrufe?

▶ Antwort:

- ▶ Falls dies bereits zur Kompilierzeit klar ist, wird der Compiler einschreiten.
- ▶ Falls dies erst zur Laufzeit klar ist, wird die JRE eine Fehlermeldung in Form einer Exception produzieren.

▶ Folgerung: Dann ist ja alles O.K., oder?



# Definitionsbereiche von Programmen

- ▶ Nochmal eine Frage:  
Was passiert, wenn ich die Methode `quadrat` mit einem `int`-Wert  $a < 0$  oder einem `char`-Wert aufrufe?
- ▶ Antwort für `int`-Wert  $a < 0$ : Die Methode terminiert nicht.
- ▶ Antwort für `char`-Wert:  
Der `char`-Wert wird in einen positiven `int`-Wert konvertiert und damit ist der Ergebniswert wohldefiniert (Autsch!).
- ▶ Vermutlich ist weder diese Wohldefiniertheit, noch die Nicht-Terminierung beabsichtigt.
- ▶ In der Praxis sind solche Situationen häufiger als man denken mag, daher ist ausführliches Kommentieren umso wichtiger!

# Definitionsbereiche von Programmen

- ▶ Zusätzlich gibt es in Java Möglichkeiten, unerwünschte Ergebnisse aufgrund von unzulässigen Eingabewerten abzufangen.
- ▶ Zusicherungen (Assertions) können benutzt werden, um beliebige Bedingungen (in Form von Booleschen Ausdrücken) an beliebigen Stellen im Programm zu platzieren (siehe nächster Unter-Abschnitt).
- ▶ Ausnahmen sind dazu da, Fehler, die während der Programmausführung auftreten können, abzufangen (die Diskussion, wann eine Ausnahme auch wirklich abgefangen werden sollte, hatten wir ja schon).

