Abschnitt 4.4

Verifikation prozeduraler Programme

Verifikation prozeduraler Programme

- Einfache Verifikation schon bei funktionalen Programmen durchgeführt.
- `assert` schon an einigen Stellen verwendet
- Wiederholung und Erweiterung auf prozedurale Programme

Verifikation prozeduraler Programme (2)

Neu...

- ungewollte Seiteneffekte ausschließen
- gewollte Seiteneffekte spezifizieren
- Schleifen
- Verifikation mit Assertions
- Spezifikationssprachen

(Vertiefung in Modul "Formale Grundlagen der Programmierung.")

Spezifikation und Verifikation

- Brückenbau: wie wissen wir, dass sie nicht zusammenbricht?
- Erzeuge ein mathematisches Modell (Mechanik, Statik usw.)
- Wähle Randbedingungen (max. Kapazität, Material usw.)
- Verifiziere, dass die Brücke unter den Bedingungen hält

Spezifikation und Verifikation (2)

Formale Methoden: versuche, dasselbe für Software zu tun.

- erstelle ein formales Modell
- spezifiziere die Eigenschaften
- verifiziere Modell und Eigenschaften

Spezifikation und Verifikation (3)

Sprachen für Spezifikation

- mathematische Notation separat vom Programm
- informelle Beschreibung in Kommentaren
- maschinenelesbare Beschreibungen

Spezifikation und Verifikation (4)

Verifikation

- separate mathematische Beweise
- Laufzeitprüfung der Eigenschaften nach Spezifikation
- formale Beweisführung, mit automatischer Unterstützung

Spezifikation und Verifikation (5)

Hier:

- einfache Spezifikationen mit `assert`
- einige zusätzliche Eigenschaften werden in Kommentaren spezifiziert

Spezifikation und Verifikation (6)

Java Konstrukt (eingebaut):

```
assert x>=0;
float y = Math.sqrt(x);
```

Spezifikation im Kommentar (zusätzliche Werkzeuge):

```
//@ assert x>=0;
float y = Math.sqrt(x);
```


Spezifikation und Verifikation (7)

Wo kommen Spezifikationen her/hin?

- Vorbedingungen (preconditions)
- Nachbedingungen (postconditions)
- Schleifeninvarianten (loop invariants)
- Schleifenvarianten (loop variants)

Spezifikation und Verifikation (8)

Wie verifizieren wir?

- Beweis von Hand
- Testen während Laufzeit
- automatische oder unterstützte Beweise

Unterabschnitt 4.4.1

Spezifikation von Prozedureigenschaften

Begriffsklärung: (Vorzustand, Nachzustand)

Den Zustand vor der Ausführung einer Prozedur nennen wir den ***Vorzustand***, den Zustand nach Ausführung den ***Nachzustand***.

Der Vorzustand beschreibt die aktuellen Parameter und den Inhalt der globalen Variablen vor Ausführung.

Der Nachzustand beschreibt den Inhalt der globalen Variablen nach Ausführung und das Ergebnis (sofern existent).

Der Einfachheit halber betrachten wir hier nur Prozeduren, die

- keine Zuweisungen an ihre Parameter enthalten, d.h. Parameter bezeichnen im Vor- und Nachzustand den gleichen Wert;
- deren Ein- und Ausgabeverhalten mit der Systemumgebung irrelevant ist.

Begriffsklärung: (Vor-, Nachbedingung)

Prozedureigenschaften lassen sich durch Vor- und Nachbedingungen beschreiben:

- Die ***Vorbedingung*** formuliert Anforderungen an den Vorzustand; wenn die Vorbedingung gilt, muss die Prozedur terminieren.
- Die Nachbedingung formuliert die Eigenschaften des Nachzustands
  - → in Abhängigkeit von den Parameterwerten;
  - → unter der Voraussetzung, dass beim Aufruf die Vorbedingung gilt.

Vor- und Nachbedingung sind seiteneffektfreie boolesche Ausdrücke (einfachste Variante). Wir bezeichnen das Ergebnis in der Nachbedingung mit `\result`.

Beispiel: (Prozedurspezifikation)

```
/*@ requires   x >= 0;
  @ modifies   \nothing
  @ ensures    \result * \result <= x  &&
  @            x < (\result+1) * (\result+1);
  @*/
int f( int x )
{
   int count = 0, sum = 1;
   while (sum <= x) {
      count++;
      sum += 2 * count + 1;
   }
   return count;
}
```


Begriffsklärung: (Prozedurspezifikation)

Eine ***Prozedurspezifikation*** besteht aus:

- einer Vorbedingung: requires <Ausdruck>
- einer Variablenliste: modifies <Liste von Variablen>
- einer Nachbedingung: ensures <Ausdruck>

Für eine leere Variablenliste schreibt man "`\nothing`".

Eine Prozedur darf nur die globalen Variablen und Verbundkomponenten/Instanzvariablen verändern, die in der Variablenliste aufgeführt sind.

Bemerkung:

Häufig reichen die Mittel der Programmiersprache nicht aus, um die Vor- und Nachbedingungen zu beschreiben. Drei Erweiterungsmöglichkeiten:

- informelle Dokumentation
- Verwenden von Funktionen, die nicht mit den Mitteln der Programmiersprache definiert sind.
- Verwenden logischer Formeln

Beispiel: (Mathematische Funktionen in Prozedurspezifikation)

```
/*@ requires   0 <= x  && x <= 12 ;
  @ modifies   \nothing
  @ ensures    \result == fakultaet(x)
  @ // wobei fakultaet wie ueblich definiert ist
  @*/
int facult( int x )
{
 int[] facres =
     {1,1,2,6,24,120,720,5040,40320,
      362880,3628800,39916800,479001600};
  return facres[x];
}
```


Beispiel: (Informelle Prozedurspezifikation)

```
/* verlangt eine Eingabe s, wobei s die Zeichenreihen-
   darstellung einer ganzen Zahl ist; modifiziert den
   Eingabe- und Ausgabestrom; druckt toInt(s)!, falls
   0 ≤ toInt(s) ≤ 12
*/
void main(String[] args ) {
  println("Parametereingabe:");
  int n = IO.readInt();
  if( n< 0 || n>12 ) {
    println("facult fuer "+ n + " nicht definiert");
  } else {
    println("facult("+n+") = "+ facult(n) );
} }
```

Bemerkung:
Prozedur main ist unterspezifiziert. Insbesondere bleibt offen, was sie im Fall `toInt(s)` > 12 tut.

Beispiel: (Spezifikation mit Quantifizierung)

```
/*@ requires   laenge == f.length;
  @ ensures    \result ==
  @   \forall i in [0,laenge-2]: f[i] ≤ f[i+1];
  @*/

boolean sortiert ( int[] f, int laenge ) {
  int i;
  for( i=0; i<laenge-1; i++ ) {
    if( f[i] > f[i+1] ) {
      return false;
    }
  }
  return true;
}
```

Bemerkung:

- Spezifikationen sind wichtig:
  - zur Dokumentation,
  - zum Testen durch dynamisches Prüfen,
  - als Grundlage für die Verifikation mit Beweis.
- Spezifikationen müssen das Verhalten nicht in allen Details festlegen (→ Unterspezifikation)
- Spezifizieren ist oft anspruchsvoller als Programmieren.