Writing Correct Programs

(Programme korrekt erstellen)

Vortrag zum Oberseminar
„Softwareentwicklung“

von Jörg Winkler (01IN-PI)

Inhalt des Vortrags

- Motivation
- Wiederholung ‚Binäre Suche‘
- Herausforderung ‚Binäre Suche‘
- Programmskizzierung
- Verifikation
- Prinzipien der Verifikation
- Aufgabe der Verifikation
- Bemerkungen & Probleme
- weitere Beispiele
- Quellen

Motivation

- Wunsch nach Programmen zur Überprüfung der Richtigkeit anderer Programme in den späten 60iger Jahren
- Neuer Ansatz: Vermittlung eines grundlegenden Verständnisses der Computerprogrammierung

  Ziel: Schreiben korrekter Programme
- Betrachtung nachfolgend am Beispiel der Binärsuche

Wiederholung *Binäre Suche*

- Ausgangslage:
  - sortiertes Array x [0 .. n-1]

    mit $n \geq 0$ und x[0] ≤ x[1] ≤ ... ≤ x[n-1]
  - der Datentyp von t und den Elementen von x ist gleich
- Verfahren (sehr grob):
  - Suche ob gesuchte Zahl t enthalten ist
- Ergebnis:
  - Position p von t in x

    mit $p = -1$, falls t nicht in x oder $x = 0$ (leeres Array) enthalten, anderenfalls $0 \leq p \leq$ n-1

Herausforderung *Binäre Suche*

- Jon Bentley:

  „Nur 10% der professionellen Programmierer, konnten nach mehreren Stunden die Binärsuche fehlerfrei in der Sprache ihrer Wahl beschreiben.“

- 1946: erste Veröffentlichung der Binären Suche
- 1962: erste fehlerfreie Binär-Suche

Programmskizzierung (Verfahren)

- Idee: wenn *t* irgendwo in *x* [0 .. *n* -1] enthalten ist, dann innerhalb der Grenzen (range) von *x*

  Kurz: mustbe(range)
- Betrachtung des mittleren Elementes von *x* und Vergleich mit *t*

  (1) *t* ist mittleres Element von *x* ... Ende

  (2) Verkleinerung der Grenzen
- Wiederholung bis *t* gefunden oder *x* leer ist

Programmskizzierung (Pseudocode)

```
1   initialize range to 0..n-1
2   loop
3           {invariant: mustbe(range)}
4           if range is empty,
5                   break and report that t is not in the array
6           compute m, the middle of the range
7           use m as probe to shrink the range
8                   if t is found during the shrinking process,
9                   break and report its position
10
11
12
```

Programmskizzierung (Pseudocode)

```
1   l = 0; u = n-1;
2   loop
3           {invariant: mustbe(l, u)}
4           if l > u
5                   p = -1; break
6           m = (l + u) / 2
7           use m as probe to shrink the range
8                   if t is found during the shrinking process,
9                   break and report its position
10
11
12
```

Programmskizzierung (Pseudocode)

```
1   l = 0; u = n-1;
2   loop
3           {mustbe(l, u)}
4           if l > u
5                   p = -1; break
6           m = (l + u) / 2
7           case
8                   x[m] < t: action a
9                   x[m] == t: action b
10                  x[m] > t: action c
11
12
```

Programmskizzierung (Pseudocode)

```
1   l = 0; u = n-1;
2   loop
3           {mustbe(l, u)}
4           if l > u
5                   p = -1; break
6           m = (l + u) / 2
7           case
8                   x[m] < t: l = m + 1
9                   x[m] == t: p = m; break
10                  x[m] > t: u = m - 1
11
12
```

Verifikation

```
1   {mustbe(0, n-1)}
2   l = 0; u  = n-1
3   {mustbe(l, u)}
4   loop
5      {mustbe(l, u)}
6      if l > u
7         {l > u && mustbe(l, u)}
8         {t is not in the array
9         p = -1; break
10     {mustbe(l, u) && l ≤ u}
11     m = (l + u) / 2
12     {mustbe(l, u) && l ≤ m ≤ u}
13     case
14        x[m] < t:
15           {mustbe(l, u) && cantbe(0, m)}
16           {mustbe(m+1, u)}
17           l = m+1
18           {mustbe(l, u)}
19        x[m] == t:
20           {x[m] == t}
21           p = m; break
22        x[m] > t:
23           {mustbe(l, u) && cantbe(m, n)}
24           {mustbe(l, m-1)}
25           u = m-1
26           {mustbe(l, u)}
27    {mustbe(l, u)}
```

Verifikation

Beweis der Richtigkeit für die Schleife (3 Teile):

- Initialization (Initialisierung):
  - Invariante ist beim ersten Schleifendurchlauf wahr
- Preservation (Erhaltung):
  - gilt die Invariante zu Beginn einer Iteration, dann ist die Invariante immer noch wahr, wenn die Schleife ausgeführt wurde
- Termination (Abschluss):
  - die Schleife terminiert und das gewünschte Ergebnis p gilt

Verifikation

- Teil 1:

  „Invariante ist beim ersten Schleifendurchlauf wahr“

- Teil 2 & 3:

  Argumentation durch nähere Betrachtung der Zeilen 4 bis 27

  durch die `break`-Anweisungen in Zeile 9 und 21 zeigen wir den Abbruch

```
3       {mustbe(l, u)}
4       loop
5       {mustbe(l, u)}

4       loop
5           {mustbe(l, u)}
...
25              u = m-1
26          {mustbe(l, u)}
27      {mustbe(l, u)}

9       p = -1; break
21          p = m; break
```

Prinzipien der Verifikation

Generelle Prinzipien:

- Assertion (Grundannahmen):
  - die Beziehungen zwischen Eingabe, Programmvariablen und Ausgabe beschreiben den Zustand „state“ eines Programms
  - Grundannahmen ermöglichen die präzise Beschreibung dieser Beziehungen
- Sequential Control Structures (Sequenzielle Strukturen zur Ablaufsteuerung):
  - um „do this statement then that statement“-Strukturen zu verstehen, setzen wird Grundannahmen dazwischen und analysieren jeden Schritt

Prinzipien der Verifikation

- Selection Control Structures (Auswahlstrukturen zur Ablaufsteuerung):
  - verschiedene Formen von if- und case-Anweisungen
  - einzelne Betrachtung der Auswahlmöglichkeiten
  - durch die Auswahl kommen nützliche Annahmen hinzu
- Iteration Control Structures (Auswahlstrukturen mit Iteration):
  - dazu müssen zuvor drei weitere Eigenschaften festgelegt werden (siehe Beispiel Binäre Suche):
    - Initalization (Initialisierung)
    - Preservation (Erhaltung)
    - Termination (Beendigung)

Prinzipien der Verifikation

- durch Induktion zeigen wir, dass die Invariante vor und nach jeder Schleifeniteration wahr ist
- weiter ist zu zeigen, dass das Ergebnis am Ende eines jeden Schleifendurchlaufes wahr ist
- zusammen zeigt dies, dass jedes Anhalten der Schleife korrekt ist

- Functions (Funktionen):
  - um eine Funktion zu überprüfen, setzen wir durch zwei Grundannahmen ihren Zweck fest:
    - Precondition (Vorbedingung)
    - Postcondition (Nachbedingung)
  - diese Bedingungen sind eher ein Vertrag als eine Tatsachen-behauptung (Programming by Contract)

Aufgaben der Verifikation

- Alternative zum normalen Überprüfen mit Testfällen
- stellt den Programmierern eine ‚Sprache' zur Verfügung, in der sie ihr Verstehen ausdrücken können
- ermöglicht die Skizzierung z.B. von Invarianten in einer Schleife
- Verletzungen von Anweisungen weisen den Weg zu Fehlern und zeigen wie diese behoben werden können ohne Neue einzuführen
- hilft dabei Grundannahmen treffen zu können und zu entscheiden, welche davon in die Software (in Form von Kommentaren) aufgenommen werden

Bemerkungen & Probleme

- die Techniken der Verifikation stellen nur einen kleinen Schritt auf den Weg zu korrekten Programmen dar
- ein einfacher Code ist meist der richtige Weg zur Korrektheit
- oft wird beim beim Programmieren von ‚schweren' Passagen erfolgreich mit mächtigen Formalismen gearbeitet, jedoch bei den vermeidlichen ‚einfachen' Stellen fällt man meist zum ‚alten' Programmierstil mit seinen Fehlern zurück

Aufgaben zum Grübeln für Unterwegs

- Beweisen Sie, dass das folgende Programm endet, wenn seine Eingabe x eine positive ganze Zahl ist.

```
while x != 1 do
        if even(x) x = x/2
        else x = 3 * x + 1
```

(über 100 Hinweise: www.research.att.com/~jcl/3x+1.html)

- ‚Kaffeedosenproblem' – Dose mit schwarzen und weißen Bohnen und einem Extrahaufen schwarzer Bohnen
  - Wähle 2 Beliebige → bei gleicher Farbe wechseln gegen Schwarze
  - Ansonsten Schwarze entfernen und Weiße zurück in die Dose

Beweis der Termination & Aussage über die Farbe der letzten Bohne in Abhängigkeit der Anzahl von weißen und schwarzen

Quellen

- [1] Jon Bentley: **Programming Pearls**
  [Addision-Wesley, 2. Auflage, 2000]
- [2] Jon Bentley: **Perlen der Programmierkunst**
  [Addision-Wesley, 2. Auflage, 2000]
- [3] Online-Ausgabe: **Progamming Pearls**
  http://www.cs.bell-labs.com/cm/cs/pearls/
- [4] David Gries: **The Science of Programming**
  [Springer Verlag, 1987]

Danke

Vortrag & Ausarbeitung
demnächst
hier:

www.imn.htwk-leipzig.de/~jwinkle3/