Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Dr. Tobias Lasser

Computer Aided Medical Procedures
Technische Universität München

Programm heute

Beispiele für Fehler I

- 1996: Explosion der Ariane 5 Rakete mit 4 Satelliten an Bord
  - Schaden: ca. 500 Millionen Dollar
  - Ursache: Wert der horizontalen Geschwindigkeit von 64 Bit Floating Point Zahl (`double`) konvertiert auf 16 Bit Ganzzahl (`signed short`)

    $\rightarrow$ Zahl grösser als $2^{15} = 32768$ $\rightarrow$ overflow

    $\rightarrow$ Zusammenbruch des Lenksystems

    $\rightarrow$ Backup-Rechner verwendet gleiches Programm, selber Fehler

    $\rightarrow$ Selbstzerstörung nach 36.7 Sekunden Flugzeit

    Annahme dass Rakete nie so schnell fliegen wird...

Beispiele für Fehler II

- 1995: Ausfall Stellwerk Hamburg-Altona
  - Schaden: 2 Tage kein Schienenverkehr in Hamburg-Altona, 2 Monate Notfahrplan
  - Ursache: Stack-Overflow

    $\rightarrow$ neues zentrales, computerisiertes Stellwerk für 62.6 Millionen D-Mark von Siemens

    $\rightarrow$ 3.5kB Stack für Stellbefehle (als sequentielle Liste)

    $\rightarrow$ zu klein schon für normalen Verkehr $\rightarrow$ overflow

    $\rightarrow$ Fehlerbehandlungsroutine fehlerhaft $\rightarrow$ Endlosschleife

Beispiele für Fehler III

- 1985 – 1987: Therac-25 Maschine zur Strahlentherapie
  - Schaden: 3 Todesfälle durch Strahlenüberdosis
  - Ursache: "race condition" in Software-Sperre für Strahl

    (race condition: Ergebnis einer Operation hängt vom zeitlichen Verhalten der Einzeloperationen ab, siehe Multitasking)

    → Overflow in 8 Bit Test-Zähler zusammen mit Benutzereingabe deaktivieren Software-Sperre für Strahl

    → unkontrollierte Bestrahlung mit bis zu 100-facher Dosis

    → 40 - 200 Gray Dosis statt 2 Gray, ab 10 Gray tödlich

    → mind. 3 Personen sterben an Strahlenüberdosis

Relative Korrektheit

Die Korrektheit eines Algorithmus bezieht sich immer auf die Spezifikation dessen, was er tun soll.

- Korrektheit ist relativ
- Algorithmus kann nicht an sich als korrekt bewiesen werden, sondern nur in Bezug auf Spezifikation

Spezifikation: eindeutige Festlegung der berechneten Funktion bzw. des Terminierungsverhaltens einer Softwarekomponente

Nachweis der Korrektheit

Nachweis der Korrektheit durch

- Verifikation: formaler, mathematischer Beweis, dass Algorithmus korrekt bezüglich formaler Spezifikation
  - erfordert Formalisierung der Algorithmensprache und Spezifikation
- Validation: nicht-formaler Nachweis der Korrektheit etwa durch systematisches Testen
  - in der Regel kein 100%ig sicherer Nachweis der Korrektheit!

Verifikation vs. Validation

E.W. Dijkstra (1930 - 2002):

> "Testing shows the presence, not the absence of bugs"

Analogie zur Mathematik:

- Spezifikation: entspricht Satz
- Verifikation: Beweis des Satzes
- Validation: Ausprobieren an Beispielwerten

Programm heute

Vor- und Nachbedingungen

Vor- und Nachbedingungen: Methode zur Spezifikation von gewünschten Eigenschaften von Algorithmen

$$\{\text{VOR}\}\ \text{ANW}\ \{\text{NACH}\}$$

- {VOR}: Aussage über Zustand vor Ausführung von ANW
- {NACH}: Aussage über Zustand nach Ausführung von ANW

Bedeutung:

- Gilt die Vorbedingung {VOR} unmittelbar vor Ausführung von ANW
- und terminiert ANW
- so gilt die Nachbedingung {NACH} unmittelbar nach Ausführung von ANW

Vor- und Nachbedingung

$$\{\text{VOR}\}\ \text{ANW}\ \{\text{NACH}\}$$

Sonderfälle:

- terminiert ANW nicht, so ist Aussage immer wahr
  - egal wie {VOR}, {NACH} aussehen
- gilt {VOR} nicht, so ist Aussage auch immer wahr
  - egal ob ANW terminiert oder ob {NACH} gilt

Vor- und Nachbedingung: Beispiele I

- Aussage
  $$\{\text{X==0}\}\ \text{X} = \text{X+1}\ \{\text{X==1}\}$$
  ist wahr
- Aussage
  $$\{\texttt{true}\}\ \text{X} = \text{Y}\ \{\text{X==Y}\}$$
  ist wahr
- Aussage
  $$\{\text{Y==a}\}\ \text{X} = \text{Y}\ \{\text{X==a} \wedge \text{Y==a}\}$$
  ist wahr für alle $a \in \mathbb{Z}$

Vor- und Nachbedingung: Beispiele II

- Aussage

  {X==a ∧ Y==b ∧ a≠b} X = Y; Y = X {X==b ∧ Y==a}

  ist falsch für alle $a, b \in \mathbb{Z}$!
  (nach erstem Schritt haben X, Y beide Wert b - typischer Fehler bei Wertetausch!)
- Aussage

  {X==a ∧ Y==b} Z = X; X = Y; Y = Z {X==b ∧ Y==a}

  ist wahr für alle $a, b \in \mathbb{Z}$ (korrekter Wertetausch mit Hilfsvariable Z)

Vor- und Nachbedingung: Beispiele III

- Aussage

  {false} ANW {NACH}

  ist wahr für alle ANW und NACH, da Vorbedingung falsch
- Aussage

  {true} ANW {false}

  ist genau dann wahr, wenn ANW nicht terminiert
- Aussage

  {X==a} **while** X ≠ 0 { X = X-1 } {X==0}

  ist wahr für alle $a \in \mathbb{Z}$
  (auch für $a < 0$, da dann while-Schleife nicht terminiert!)

Partielle Korrektheit

Partielle Korrektheit

Ein Programm mit Anweisungen ANW sowie Vorbedingung {VOR} und Nachbedingung {NACH} heißt partiell korrekt bezüglich VOR, NACH genau dann, wenn

{VOR} ANW {NACH}

wahr ist.

Das Programm heißt total korrekt bezüglich VOR, NACH genau dann, wenn

- es partiell korrekt ist bezüglich VOR, NACH ist und
- ANW immer dann terminiert, wenn VOR gilt.

Bausteine von Algorithmen

Wiederholung: Bausteine von Algorithmen

- Elementarer Verarbeitungsschritt (z.B. Zuweisung an Variable)
- Sequenz (elementare Schritte nacheinander)
- Bedingter Verarbeitungsschritt (z.B. if/else)
- Wiederholung (z.B. while-Schleife)

Korrektheit von Anweisungstypen I

- Elementarer Verarbeitungsschritt α

  $$\{\text{VOR}\}\ \alpha\ \{\text{NACH}\}$$

- Sequenz $\alpha; \beta$

  $$\{\text{VOR}\}\ \alpha; \beta\ \{\text{NACH}\}$$

  wird gezeigt mittels Zwischenbedingung MITTE, also

  $$\{\text{VOR}\}\ \alpha\ \{\text{MITTE}\}$$

  und

  $$\{\text{MITTE}\}\ \beta\ \{\text{NACH}\}$$

Korrektheit von Anweisungstypen II

- Bedingter Verarbeitungsschritt

  $$\{\text{VOR}\}\ \textbf{if}\ (\text{B})\ \{\ \alpha\ \}\ \text{else}\ \{\ \beta\ \}\ \{\text{NACH}\}$$

  wird gezeigt mittels

  $$\{\text{VOR} \wedge \text{B}\}\ \alpha\ \{\text{NACH}\}$$

  und

  $$\{\text{VOR} \wedge \neg\text{B}\}\ \beta\ \{\text{NACH}\}$$

  $\rightarrow$ zeige also dass NACH gilt, egal welcher Zweig ausgeführt wird!

Korrektheit von Anweisungstypen III

- Wiederholung

  $\{\text{VOR}\}$
  while (B) { β }
  $\{\text{NACH}\}$

  wird gezeigt mittels Schleifeninvariante P:

  1. prüfe, daß VOR $\Rightarrow$ P
     (Vorbedingung garantiert Schleifeninvariante bei Eintritt in Schleife)
  2. prüfe, daß

     $$\{\text{P} \wedge \text{B}\}\ \beta\ \{\text{P}\}$$

     (wenn Schleife durchlaufen, bleibt Schleifeninvariante wahr)
  3. prüfe, daß $\{\text{P} \wedge \neg\text{B}\} \Rightarrow$ NACH
     (Nachbedingung muß nach Verlassen der Schleife gelten)

Verifikation: Beispiel I

Gauss'sche Summenformel

Sei $n \in \mathbb{N}$. Dann gilt

$$1 + 2 + \ldots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Algorithmus:

```
{ n > 0 }
   sum = 0;
   i = 0;
   while (i < n) {
      i = i + 1;
      sum = sum + i;
   }
{ sum == n*(n+1)/2 }
```

Verifikation: Beispiel II

Sequenz mit Zwischenbedingung:

```
{ n > 0 }
   sum = 0;
{ n > 0 ∧ sum == 0 }
   i = 0;
{ n > 0 ∧ sum == 0 ∧ i == 0 }
```

Algorithmus:

```
{ n > 0 }
   sum = 0;
   i = 0;
   while (i < n) {
      i = i + 1;
      sum = sum + i;
   }
{ sum == n*(n+1)/2 }
```


Verifikation: Beispiel III

Wiederholung mit Invariante:

```
{ n > 0 ∧ sum == 0 ∧ i == 0 }
   while (i < n) {
      i = i + 1;
      sum = sum + i;
   }
{ sum == n*(n+1)/2 }
```

Passende Invariante P?

P = (sum == i*(i+1)/2 ∧ i <= n)

Algorithmus:

```
{ n > 0 }
   sum = 0;
   i = 0;
   while (i < n) {
      i = i + 1;
      sum = sum + i;
   }
{ sum == n*(n+1)/2 }
```


Verifikation: Beispiel IV

Prüfe:

1. {VOR} ⇒ {P}

   es ist i==0, also i<=n, und sum==0*1/2==0 ✓

2. {P ∧ B} β {P}

   es ist i<n, also nach i=i+1 immer noch i<=n , und für sum gilt

   $\text{sum}=\frac{i(i+1)}{2}+(i+1)==\frac{(i+1)*(i+2)}{2}$ ✓

3. {P ∧ ¬B} ⇒ {NACH}

   es ist P ∧ ¬B ⇔ P ∧ !(i<n)

   ⇔ $\text{sum}==\frac{i(i+1)}{2}$ ∧ i<=n ∧ !(i<n)

   ⇔ $\text{sum}==\frac{i(i+1)}{2}$ ∧ i==n

   ⇒ $\text{sum}==\frac{n(n+1)}{2}$ ✓

Wiederholung mit Invariante:

```
{ n > 0 ∧ sum == 0 ∧ i == 0 }
   while (i < n) {
      i = i + 1;
      sum = sum + i;
   }
{ sum == n*(n+1)/2 }
```

Invariante P:

P = (sum == i*(i+1)/2 ∧ i <= n)

Verifikation

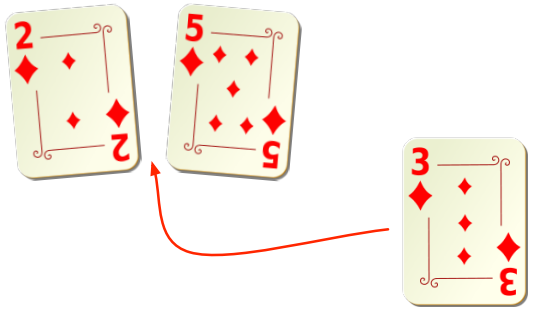
- Verifikation ist manuell sehr aufwendig
  - in Praxis daher leider sehr selten eingesetzt
- Vorgang lässt sich weitgehend automatisieren

  → Programm-Verifizierer
  - Programm mit Vor- und Nachbedingungen
  - Schleifen müssen Invarianten haben
  - nur möglich mit eingeschränkten Programmiersprachen z.B. Lightweight Java
  - an der TUM: Programmsystem Isabelle (Informatik) `http://isabelle.in.tum.de`

Programm heute

Insertion Sort

Insertion Sort: Sortieren durch direktes Einfügen



Beschreibung von Insertion Sort

Beschreibung in Vorlesung durch C-ähnlichen Pseudocode:

Input: Feld A[0..n-1] von n natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):

```
for j=1 to länge(A)-1 {
  key = A[j];
  // füge A[j] in sortierte Liste A[0..j-1] ein
  i = j-1;
  while (i >= 0 && A[i] > key) {
    A[i+1] = A[i];
    i = i-1;
  }
  A[i+1] = key;
}
```


Pseudocode Konventionen I

- Einrücken und geschweifte Klammern {, } kennzeichnen Blockstruktur
- Sequenz durch Semikolon ;
- // bedeutet Rest der Zeile ist Kommentar (wird nicht ausgeführt)
- Wert-Zuweisung, z.B. x=5 weist der Variable x den Wert 5 zu
- Feld A[0..n-1], Zugriff auf i-tes Element A[i] mit Index i (meist 0-basiert!)

  A[0..n-1] bezeichnet die Elemente A[0], A[1], . . . , A[n-1]

Beispiel:

```
for j=1 to länge(A)-1 {
  key = A[j];
  // füge A[j] in sortierte
  // Liste A[0..j-1] ein
  i = j-1;
  while (i >= 0 && A[i] > key) {
    A[i+1] = A[i];
    i = i-1;
  }
  A[i+1] = key;
}
```

Pseudocode Konventionen II

- for-Schleife:

  for j=Wert1 **to** Wert2 { ... }

  Variable j nimmt schrittweise die Werte von Wert1 bis Wert2 an, jedes Mal wird Anweisung des Schleifenkörpers (...) ausgeführt

  Am Ende der Schleife hat j den Wert Wert2+1

- while-Schleife:

  while (Bedingung) { ... }

  Führt Schleifenkörper (...) aus, solange Bedingung erfüllt ist

Beispiel:

```
for j=1 to länge(A)-1 {
    key = A[j];
    // füge A[j] in sortierte
    // Liste A[0..j-1] ein
    i = j-1;
    while (i >= 0 && A[i] > key) {
       A[i+1] = A[i];
       i = i-1;
    }
    A[i+1] = key;
}
```


Pseudocode Konventionen III

- if-Abfrage:

  if (Bedingung) { ... }
  else { ... }

  Verzweigung im Code anhand Bedingung. Falls Bedingung nicht erfüllt, kennzeichnet **else** den Zweig, der gewählt wird

- **print(x)** Anweisung gibt Wert von Variable x aus

Beispiel:

```
i=3;
if (i==0) {
    x=0;
} else {
    x=2;
}
print(x);
```


Insertion Sort: Ablauf

Beispielablauf:

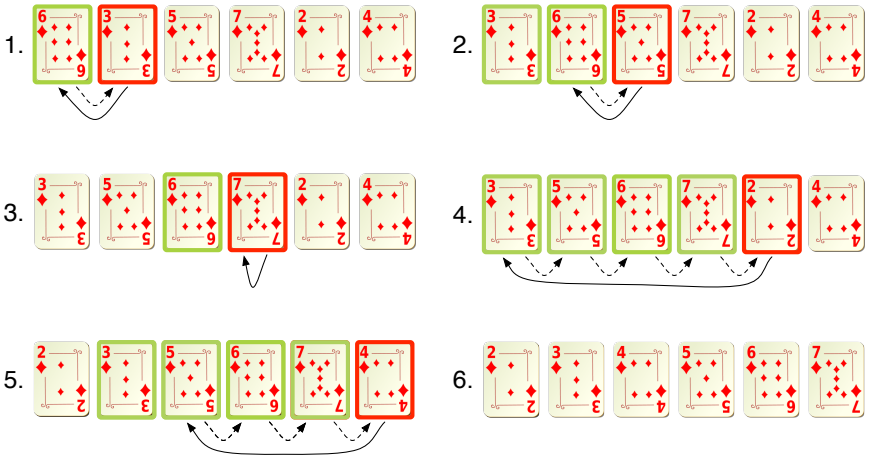
Eingabe: A = 6, 3, 5, 7, 2, 4

j			
key			
A[0..j-1]			
i			

InsertionSort(A):

```
for j=1 to länge(A)-1 {
    key = A[j];
    // füge A[j] in sortierte
    // Liste A[0..j-1] ein
    i = j-1;
    while (i >= 0 && A[i] > key) {
       A[i+1] = A[i];
       i = i-1;
    }
    A[i+1] = key;
}
```


Insertion Sort: Ablauf



Insertion Sort: Verifikation

Entscheidendes Kriterium: Schleifen-Invariante P der **for** Schleife

P = Teilfeld A[0..j-1] besteht aus den ursprünglich in A[0..j-1] enthaltenen Werten und A[0..j-1] ist geordnet

Prüfe:

1. Anfang j=1: Feld A[0..j-1] enthält nur ein Element A[0], ist sortiert ✓
2. aktuelles Element mit Wert key, die Werte A[j-1], A[j-2], ... werden jeweils eine Position nach rechts verschoben, bis korrekte Position A[i] gefunden für aktuelles Element, Element wird dort eingefügt → Feld A[0..j-1] sortiert ✓
3. Algorithmus terminiert wenn j==n, also gilt mit voriger Invariante P: A[0..n-1] ist sortiert ✓

Zusammenfassung