Verifizierende Testverfahren

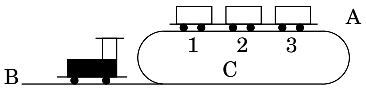
Spezifikation

Um einen Algorithmus zu schreiben, muss das zu lösende Problem genau beschrieben sein. Eine Spezifikation ist

- **vollständig**, wenn alle Anforderungen/alle relevanten Rahmenbedingungen angegeben wurden.
- **detailliert**, wenn klar definiert ist, welche Hilfsmittel/ Operationen zur Lösung zugelassen sind.
- **unzweideutig**, wenn klar angegeben ist, wann eine vorgeschlagene Lösung akzeptabel ist.

Spezifikation (2)

Beispiel: Eine Lok soll die in Abschnitt A stehenden Wagen 1, 2, 3 in der Reihenfolge 3, 1, 2 auf Gleis C abstellen.



Vollständigkeit:
- Wieviele Wagen kann die Lok auf einmal ziehen?
- Wieviele Wagen passen auf Gleisstück B?

Detailliertheit:
- Was kann die Lok? (fahren, koppeln, entkoppeln)

Unzweideutigkeit:
- Darf die Lok am Ende zwischen den Wagen stehen?

Spezifikation (3)

Warum Spezifikation?

- Spezifikation ist Teil des Feinentwurfs
- für jede Funktion beschreiben, was sie tut
- letzte Phase vor Implementierung
- Problemstellung präzisieren
- Entwicklung unterstützen
- Überprüfbarkeit verbessern
- Wiederverwendbarkeit erhöhen

Ziel: Erst denken, dann den Algorithmus entwerfen.

informale Spezifikation

- beschreibe kurz für jede Funktion, was sie tut
- enthält mindestens die Rolle der Parameter/Rückgabewerte und ggf. die Seiteneffekte
- weit verbreitet, gut für die Dokumentation geeignet
- nicht exakt genug, um die Einhaltung der Spezifikation nachzuweisen

Beispiel: `reactor.isCooking()` liefert `true`, wenn Temperatur des Reaktors 100 Grad Celsius erreicht.

Probleme:

- was bedeutet erreicht?
- was passiert bei Temperaturen unter 100 Grad?

exemplarische Spezifikation

- Testfälle beschreiben Beispiele für das Zusammenspiel der Funktionen samt erwarteter Ergebnisse
- formales Verfahren, trotzdem leicht verständlich
- nach der Implementierung dienen die Testfälle zur Validierung
- durch *extreme programming* populär geworden

Beispiel:

```
reactor.setTemperature(99);
assert(!reactor.isCooking());
reactor.setTemperature(100);
assert(reactor.isCooking());
```


formale Spezifikation

- mittels formaler Beschreibungssprache die Semantik der Funktionen exakt festlegen
- eine ausführbare Spezifikationssprache kann als Prototyp dienen
- Möglichkeit des Programmbeweises
- aber: aufwändig, erhöhte Anforderungen an Verwender

eigenschaftsorientiert:

- axiomatisch: Objekte werden aus Typen konstruiert und durch logische Zusicherungen spezifiziert
- algebraisch: definiert Algebren über Objekte und deren Funktionalität

formale Spezifikation (2)

modellorientiert:

- Modellbeschreibung durch reiche, formale Sprache
- Beispiele: VDM, Z

automatenorientiert:

- beschreibt Zustände und Übergänge des Systems
- Beispiele: Petri-Netze, State Charts

Bestandteile der formalen Spezifikation

1. Ein Algorithmus berechnet eine Funktion, daher:
   (a) festlegen der gültigen Eingaben (Definitionsbereich)
   (b) festlegen der gültigen Ausgaben (Wertebereich)
2. Funktionaler Zusammenhang zwischen Ein-/Ausgabe:
   (a) Welche Eigenschaften hat die Ausgabe?
   (b) Wie sehen die Beziehungen der Ausgabe zur Eingabe aus?
3. Festlegen der erlaubten Operationen.

formale Spezifikation: Beispiele

Euklidischer Algorithmus

gegeben: $n, m \in \mathbb{N}$
gesucht: $g \in \mathbb{N}$
funktionaler Zusammenhang: $g = ggT(n, m)$

Jüngste Person

gegeben: $(a_1, \ldots, a_n) \in \mathbb{R}^n, n > 0$
gesucht: $p \in \{1, \ldots, n\}$
funktionaler Zusammenhang:

- $\forall\ i \in \{1, \ldots, n\}$ gilt $a_p \leq a_i$ oder alternativ:
- $\forall\ j \in \{1, \ldots, n\}$ gilt: $a_j \neq a_p \Rightarrow a_j > a_p$.

Verifikation

Ziel: Beweise, dass der Algorithmus korrekt ist.

Wechselspiel zwischen:

- statische Aussagen über den Algorithmus ohne ihn auszuführen $\rightarrow$ nicht vollständig möglich, zu komplex und umfangreich
- dynamisches Testen des Algorithmus $\rightarrow$ zeigt nur die Anwesenheit von Fehlern, nicht deren Abwesenheit

Die Programmverifikation versucht zu zeigen, dass der Algorithmus die funktionale Spezifikation erfüllt:

- Der Algorithmus liefert zu jeder Eingabe eine Ausgabe.
- Die Ausgabe ist die gewünschte Ausgabe.

Prädikatenkalkül Floyd/Hoare

die Wirkung eines Programms ist spezifiziert durch zwei Zusicherungen:

- **Anfangsbedingung** (Vorbedingung, precondition)
  legt vor dem Ablauf des Programms zulässige Werte der Variablen fest
- **Endebedingung** (Nachbedingung, postcondition)
  legt gewünschte Werte der Variablen und Beziehungen zwischen Variablen nach Programmlauf fest
- **Notation:** $\{P\}$ spezifiziertes Programm $\{Q\}$
- kann eine Spezifikation nicht durch ein Programm erfüllt werden, so nennt man sie **widersprüchlich**

Prädikatenkalkül Floyd/Hoare (2)

Verifikationsregeln

- Programme bestehen aus linearen Kontrollstrukturen $\rightarrow$ die Korrektheit des gesamten Programms ergibt sich aus der Korrektheit der Teilstrukturen
- komplexes Programm durch schrittweises Zusammensetzen einfacher Strukturen verifizieren

Notation: $\{P\}$ Schritt $\{Q\}$

- $P =$ Vorbedingung
- $Q =$ Nachbedingung
- Semantik: falls vor Ausführung des Schrittes P gilt, dann gilt nachher Q

Prädikatenkalkül Floyd/Hoare (3)

Sequenz-Regel

$$\begin{array}{l}\{P\}\ S_1\ \{Q\}\\ \{Q\}\ S_2\ \{R\}\end{array} \quad \Rightarrow \quad \{P\}\ S_1; S_2\ \{R\}$$

- Semantik: zwei Programmteile S_1 und S_2 können genau dann zusammengesetzt werden, wenn die Nachbedingung von S_1 gleich der Vorbedingung von S_2 ist

abgekürzte Schreibweise

$\{P_0\}$ Schritt 1 $\{P_1\}$ Schritt 2 $\{P_2\}$ ... $\{P_{n-1}\}$ Schritt n $\{P_n\}$
oder alternativ: $\{P_0\}$ Algorithmus $\{P_n\}$

- Semantik: falls vor Ausführung des Algorithmus P_0 gilt, dann gilt nachher P_n

Prädikatenkalkül Floyd/Hoare (4)

Zuweisungs-Regel $\{P(v)\} \quad v := t \quad \{P(t)\}$

- Semantik: was vorher für t gilt, gilt nachher für v
- Beispiele:
  * $\{t > 0\} \quad v := t \quad \{t > 0 \wedge v > 0\}$
  * $\{x = 2\} \quad x := x + 1 \quad \{x = 3\}$
  * $\{x = 3\} \quad v := 2x + 1 \quad \{v = 7 \wedge x = 3\}$

symbolische Ausführung: Der in der Vorbedingung gegebene, anfängliche Wert ist in den zugewiesenen Ausdruck einzusetzen. im Beispiel:

- Vorbed. $x = 2 \quad \Rightarrow \quad$ Nachbed. $x + 1 = 3$
- Vorbed. $x = 3 \quad \Rightarrow \quad$ Nachbed. $2x + 1 = 7$

Prädikatenkalkül Floyd/Hoare (5)

Anmerkungen:

- auch andere Richtung möglich:

  Nachbedingung $\Rightarrow$ zugehörige Vorbedingung

  in Nachbedingung alle Vorkommen der zugewiesenen Variablen durch zuweisenden Ausdruck ersetzen:
  * Nachbed. $x + 1 = 3 \quad \Rightarrow \quad$ Vorbed. $x = 2$
  * Nachbed. $2x + 1 = 7 \quad \Rightarrow \quad$ Vorbed. $x = 3$
- welche Richtung man wählt, hängt davon ab, ob man die Vor- oder die Nachbedingung kennt

Prädikatenkalkül Floyd/Hoare (6)

Konsequenz-Regel $\{P'\}\ S\ \{Q'\} \quad \Rightarrow \quad \{P\}\ S\ \{Q\}$

- Vorbedingung P ist stärker/schärfer als P'
- Nachbedingung Q ist schwächer als Q'

Beispiel:

$$\begin{array}{lcl} & & \{x > 45\} \\ & & \quad\Downarrow \\ \{x > 40\} & & \{x > 40\} \\ \quad x := x * 3 & & \quad x := x * 3 \\ \quad x := x - 30 & \Rightarrow & \quad x := x - 30 \\ \{x > 90\} & & \{x > 90\} \\ & & \quad\Downarrow \\ & & \{x > 80\} \end{array}$$

Prädikatenkalkül Floyd/Hoare (7)

Konsequenz-Regel (Fortsetzung)

- Bedingungen abschwächen bei Vorwärts-Durchlauf:
  * hinzufügen eines Terms mit ODER-Verknüpfung
  * weglassen eines UND-verknüpften Terms
- Bedingungen verschärfen bei Rückwärts-Durchlauf:
  * hinzufügen eines Terms mit UND-Verknüpfung
  * weglassen eines ODER-verknüpften Terms

Beispiel:

$$\begin{array}{lcl} \{x < y \ \vee \ x = y\} & & \{x < y\} \Rightarrow \{x < y \ \vee \ x = y\} \\ \quad S & \Rightarrow & \quad S \\ \{x = y + 2\} & & \{x = y + 2\} \Rightarrow \{y \leq x\} \end{array}$$

Prädikatenkalkül Floyd/Hoare (8)

die Sequenz-Regel kann mit Hilfe der Konsequenz-Regel verallgemeinert werden:

zwei Programmteile S_1 und S_2 können zusammengesetzt werden, wenn die Nachbedingung von S_1 schärfer als die Vorbedingung von S_2 ist

$$\begin{array}{lcl} \{Q\}\ S_1\ \{P'\} & & \\ \{P'\} \Rightarrow \{P\} & \Rightarrow & \{Q\}\ S_1; S_2\ \{R\} \\ \{P\}\ S_2\ \{R\} & & \end{array}$$

Prädikatenkalkül Floyd/Hoare (9)

Wiederholung $\{P\}$ solange B wiederhole Schritt $\{P \wedge \neg B\}$

- Semantik: P gilt vor und nach jeder Ausführung von Schritt $\rightarrow$ Schleifeninvariante
- Beispiel:

$x := y$
$k := 0$
$\{y = k \cdot a + x\}$
solange $x \geq 0$ wiederhole
    $x := x - a$
    $k := k + 1$
$\{y = k \cdot a + x \wedge x < 0\}$

Prädikatenkalkül Floyd/Hoare (10)

Anmerkungen:

- die Invariante gilt nicht an allen Stellen des Schleifenrumpfs, nur am Anfang und am Ende
- ähnliche Regeln existieren für Zählschleifen und fußgesteuerte Schleifen wie `do ... while ();`
  (werden hier nicht vorgestellt)

if-Regel

$$\begin{array}{l}\{Q \land \ B\}\ S_1\ \{P\} \\ \{Q \land \neg B\}\ S_2\ \{P\}\end{array} \quad \Rightarrow \quad \{Q\} \text{ if } B \text{ then } S_1 \text{ else } S_2\ \{P\}$$

Verifikation: Beispiel

Algorithmus:

$s := 0$
$j := n$
while $j > 0$ do
  $s := s + j$
  $j := j - 1$

Werte für $n = 4$:

Runde	j	s
0	4	0
1	3	4
2	2	$4+3=7$
3	1	$4+3+2=9$
4	0	$4+3+2+1=10$

Invariante?

$$s = \sum_{i=j+1}^{n} i$$

Verifikation: Beispiel (2)

$\{n \in \mathbb{N}_0\}$
$s := 0$
$j := n$
$\{n \in \mathbb{N}_0 \land s = 0 \land j = n \land j \in \mathbb{N}_0 \land s = \sum_{i=j+1}^{n} i\}$
while $j > 0$ do
  $\{n \in \mathbb{N}_0 \land j > 0 \land j \in \mathbb{N}_0 \land s = \sum_{i=j+1}^{n} i = \sum_{i=j}^{n} i - j\}$
  alternativ: $\{n \in \mathbb{N}_0 \land j > 0 \land j \in \mathbb{N}_0 \land s + j = \sum_{i=j}^{n} i\}$
  $s := s + j$
  $\{n \in \mathbb{N}_0 \land j > 0 \land j \in \mathbb{N}_0 \land s = \sum_{i=j}^{n} i = \sum_{i=j-1+1}^{n} i\}$
  $j := j - 1$
  $\{n \in \mathbb{N}_0 \land j > 0 \land j \in \mathbb{N}_0 \land s = \sum_{i=j+1}^{n} i\}$
$\{n \in \mathbb{N}_0 \land j \leq 0 \land j \in \mathbb{N}_0 \land s = \sum_{i=j+1}^{n} i = \sum_{i=1}^{n} i\}$