

Einführung in die Anwendungsorientierte Informatik (Köthe)

Robin Heinemann

November 2, 2016

Contents

1	Klausur 09.02.2016	2
2	Was ist Informatik?	2
2.1	Teilgebiete	2
2.1.1	theoretische Informatik (ITH)	2
2.1.2	technische Informatik (ITE)	2
2.1.3	praktische Informatik	2
2.1.4	angewante Informatik	3
3	Wie unterscheidet sich Informatik von anderen Disziplinen?	3
3.1	Mathematik	3
4	Informatik	3
4.1	Algorithmus	4
4.2	Daten	4
4.2.1	Beispiele für Symbole	4
4.3	Einfachster Computer	5
4.3.1	TODO Graphische Darstellung	5
4.3.2	TODO Darstellung durch Übergangstabellen	5
4.3.3	Beispiel 2:	6
5	Substitutionsmodell (funktionale Programmierung)	7
5.1	Substitutionsmodell	8

1 Klausur 09.02.2016

2 Was ist Informatik?

”Kunst” Aufgaben mit Computerprogrammen zu lösen.

2.1 Teilgebiete

2.1.1 theoretische Informatik (ITH)

- Berechenbarkeit: Welche Probleme kann man mit Informatik lösen und welche prinzipiell nicht?
- Komplexität: Welche Probleme kann man effizient lösen?
- Korrektheit: Wie beweist man, dass das Ergebnis richtig ist?
Echtzeit: Dass das richtige Ergebnis rechtzeitig vorliegt.
- verteilte Systeme: Wie sichert man, dass verteilte Systeme korrekt kommunizieren?

2.1.2 technische Informatik (ITE)

- Auf welcher Hardware kann man Programme ausführen, wie baut man dies Hardware?
- CPU, GPU, RAM, HD, Display, Printer, Networks

2.1.3 praktische Informatik

- Wie entwickelt man Software?
- Programmiersprachen und Compiler: Wie kommuniziert der Programmierer mit der Hardware? **IPI, IPK**
- Algorithmen und Datenstrukturen: Wie baut man komplexe Programme aus einfachen Grundbausteinen? **IAL**
- Softwaretechnik: Wie organisiert man sehr große Projekte? **ISW**
- Kernanwendung der Informatik: Betriebssysteme, Netzwerke, Parallelisierung **IBN**
 - Datenbanksysteme **IDB1**

- Graphik, Graphische Benutzerschnittstellen
- Bild- und Datenanalyse
- maschinelles Lernen
- künstliche Intelligenz

ICG1

2.1.4 angewante Informatik

- Wie löst man Probleme aus einem anderem Gebiet mit Programmen?
- Informationstechnik
 - Buchhandlung, e-commerce, Logistik
- Web programming
- scientific computing für Physik, Biologie
- Medizininformatik
 - bildgebende Verfahren
 - digitale Patientenakte
- computer linguistik
 - Sprachverstehen, automatische Übersetzung
- Unterhaltung: Spiele, special effect im Film

3 Wie unterscheidet sich Informatik von anderen Disziplinen?

3.1 Mathematik

Am Beispiel der Definition $a \leq b : \exists c \geq 0 : a + c = b$ Informatik: Lösungsverfahren: $a - b \leq 0$, das kann man leicht ausrechnen, wenn man subtrahieren und mit 0 vergleichen kann. Quadratwurzel: $y = \sqrt{x} \Leftrightarrow y \geq 0 \wedge y^2 = x (\Rightarrow x > 0)$ Informatik: Algorithmus aus der Antike: $y = \frac{x}{y}$ iteratives Verfahren:

Initial Guess $y^{(0)} = 1$ schrittweise Verbesserung $y^{(t+1)} = \frac{y^{(t)} + \frac{x}{y^{(t)}}}{2}$

4 Informatik

Lösungswege, genauer Algorithmen

4.1 Algorithmus

schematische Vorgehensweise mit der jedes Problem einer bestimmten **Klasse** mit **endliche** vielen **elementaren** Schritten / Operationen gelöst werden kann

- schematisch: man kann den Algorithmus ausführen, ohne ihn zu verstehen (\Rightarrow Computer)
- alle Probleme einer Klasse: zum Beispiel: die Wurzel aus jeder beliebigen nicht-negativen Zahl, und nicht nur $\sqrt{11}$
- endliche viele Schritte: man kommt nach endlicher Zeit zur Lösung
- elementare Schritte / Operationen: führen die Lösung auf Operationen oder Teilprobleme zurück, die wir schon gelöst haben

4.2 Daten

Daten sind Symbole,

- die Entitäten und Eigenschaften der realen Welt im Computer representieren.
- die interne Zwischenergebnisse eines Algorithmus aufbewahren

\Rightarrow Algorithmen transformieren nach bestimmten Regeln die Eingangsdaten (gegebene Symbole) in Ausgangsdaten (Symbole für das Ergebniss). Die Bedeutung / Interpretation der Symbole ist dem Algorithmus egal \triangleq "schematisch"

4.2.1 Beispiele für Symbole

- Zahlen
- Buchstaben
- Icons
- Verkehrszeichen

aber: heutige Computer verstehen nur Binärzahlen \Rightarrow alles andere muss man übersetzen Eingangsdaten: "Ereignisse":

- Symbol von Festplatte lesen oder per Netzwerk empfangen

- Benutzerinteraktion (Taste, Maus, ...)
- Sensor übermittelt Meßergebnis, Stoppuhr läuft ab

Ausgangsdaten: "Aktionen":

- Symbole auf Festplatte schreiben, per Netzwerk senden
- Benutzeranzeige (Display, Drucker, Ton)
- Stoppuhr starten
- Roboteraktion ausführen (zum Beispiel Bremsassistent)

Interne Daten:

- Symbole im Hauptspeicher oder auf Festplatte
- Stoppuhr starten / Timeout

4.3 Einfachster Computer

endliche Automaten (endliche Zustandsautomaten)

- befinden sich zu jedem Zeitpunkt in einem bestimmten Zustand aus einer vordefinierten endlichen Zustandsmenge
- äußere Ereignisse können Zustandsänderungen bewirken und Aktionen auslösen

4.3.1 TODO Graphische Darstellung

graphische Darstellung: Zustände = Kreise, Zustandsübergänge: Pfeile

4.3.2 TODO Darstellung durch Übergangstabellen

Zeilen: Zustände, Spalten: Ereignisse, Felder: Aktion und Folgezustände

Zustände \ Ereignisse	Knopf drücken	Timeout
aus	$\Rightarrow \{\text{halb}\}$	
{4 LEDs an}	%	$(\Rightarrow \{\text{aus}\}, \{\text{nichts}\})$
halb	$(\Rightarrow \{\text{voll}\}, \{8 \text{ LEDs an}\})$	%
voll	$(\Rightarrow \{\text{blinken an}\}, \{\text{Timer starten}\})$	%
blinken an	$(\Rightarrow \{\text{aus}\}, \{\text{Alle LEDs aus, Timer stoppen}\})$	$(\Rightarrow \{\text{blinken aus}\}, \{\text{alle LEDs an}\})$
blinken aus	$(\Rightarrow \{\text{aus}\}, \{\text{Alle LEDs aus, Timer stoppen}\})$	$(\Rightarrow \{\text{blinken an}\}, \{\text{alle LEDs an}\})$

Variante: Timer läuft immer (Signal alle 0.3s) \Rightarrow Timeout ignorieren im Zustand "aus", "halb", "voll"

4.3.3 Beispiel 2:

$$\begin{array}{rcl}
 1\ 0\ 1\ 1\ 0\ 1\ 0 & = 2 + 8 + 16 + 74 = 90_{\text{dez}} & (1) \\
 +0\ 1\ 1\ 1\ 0\ 0\ 1 & = 1 + 8 + 16 + 32 = 57_{\text{dez}} & (2) \\
 \hline
 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1 & = 1 + 2 + 16 + 128 = 147_{\text{dez}} \checkmark & (3)
 \end{array}$$

1. Implementation mit Endlichen Automaten Prinzipien:

- wir lesen die Eingangsdaten von rechts nach links
- Beide Zahlen gleich lang (sonst mit 0en auffüllen)
- Ergebnis wird von rechts nach link ausgegeben

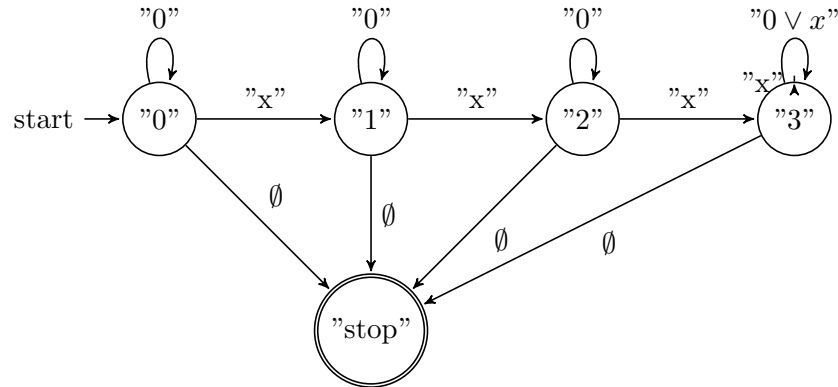
2. **TODO** Skizze der Automaten

Zustand	Ereignis	Ausgeben
start	(0,1)	"1"
start	(1,0)	"1"
start	(0,0)	"0"
start	(1,1)	"0"
carry = 1	(1,1)	"1"
carry = 1	(0,1)	"0"
carry = 1	(1,0)	"0"
carry = 1	\emptyset	"1"

Wichtig: In jedem Zustand muss für **alle möglichen** Ereignisse eine Aktion und Folgezustand definiert werden. Vergisst man ein Ereignis zeigt der Automat undefiniertes Verhalten, also ein "Bug" Falls keine sinnvolle Reaktion möglich ist: neuer Zustand: "Fehler" \Rightarrow Übergang nach "Fehler", Aktion: ausgeben einer Fehlermeldung

(a) **TODO** Skizze Fehlermeldung Ein endlicher Automat hat nur ein Speicherelement, das den aktuellen Zustand angibt. Folge:

- Automat kann sich nicht merken, wie er in den aktuellen Zustand gekommen ist ("kein Gedächtnis")
- Automat kann nicht beliebig weit zählen, sondern nur bis zu einer vorgegebenen Grenze



Insgesamt: Man kann mit endlichen Automaten nur relativ einfache Algorithmen implementieren. (nur reguläre Sprachen)
Spendiert man zusätzlichen Speicher, geht mehr:

- Automat mit Stak-Speicher (Stapel oder Keller) \Rightarrow Keller-automat (Kontextfreie Sprachen)
- Automat mit zwei Stacks oder äquivalent Turing-Maschine kann alles auführen, was man intuitiv für berechenbar hält

Markov Modelle: endliche Automaten mit probabilistischen Übergängen. Bisher: Algorithmen für einen bestimmten Zweck (Problemklasse) Frage: Gibt es einen universellen Algorithmus für alle berechenbare Probleme? Betrachte formale Algorithmusbeschreibung als Teil der Eingabe des universellen Algorithmus.

5 Substitutionsmodell (funktionale Programmierung)

- einfaches Modell für arithmetische Berechnung "Taschenrechner"
- Eingaben und Ausgaben sind Zahlen (ganze oder reelle Zahlen). Zahlenkonstanten heißen "Literals"
- elementare Funktionen: haben eine oder mehrere Zahlen als Argumente (Parameter) und liefern eine Zahl als Ergebnis (wie Mathematik):
 - $\text{add}(1,2) \rightarrow 3$, $\text{mul}(2,3) \rightarrow 6$, analog $\text{sub}()$, $\text{div}()$, $\text{mod}()$
- Funktionsaufrufe können verschachtelt werden, das heißt Argumente kann Ergebnis einer anderen Funktion sein
 - $\text{mul}(\text{add}(1,2), \text{sub}(5,3)) \rightarrow 6$

5.1 Substitutionsmodell

Man kann einen Funktionsaufruf, deessen Argument vekannt ist (das heißt Zahlen sind) durch den Wert des Ergebnisses ersetzen ("substituieren"). Geschachtelte Ausdrücke lassen sich so von innen nach außen auswerten.

$$\text{mul}(\text{add}(1, 2), \text{sub}(5, 3))$$

$$\text{mul}(3, \text{sub}(5, 3))$$

$$\text{mul}(3, 2)$$

6

- Die arithmetischen Operationene `add()`, `sub()`, `mul()`, `div()`, `mod()` werden normalerweise von der Hardware implementiert.
- Die meisten Programmiersprachen bieten außerdem algebraische Funktionen wie: `sqrt()`, `sin()`, `cos()`, `log()`
 - sind meist nicht in Hardware, aber vorgefertigte Algorithmen, werden mit Programmiersprachen geliefert, "Standardbibilothek"
- in C++: mathematisches Modul des Standardbibilothek: `"cmath"`
- Für Arithmetik gebräuchlicher ist "Infix-Notation" mit Operator-Symbolen `"+"`, `"-"`, `"*"`, `"/"`, `"%"`
- $\text{mul}(\text{add}(1,2),\text{sub}(5,3)) \Leftrightarrow ((1+2)*(5-3))$
 - oft besser, unter anderem weil man Klammer weglassen darf
 1. "Punkt vor Strichrechnung" $3+4*5 \Leftrightarrow 3+(4*5)$, `mul`, `div`, `mod` binden stärker als `add`,`sub`
 2. Operatoren gleicher Präzedenz werden von links nach rechts ausgeführt (links-assoziativ)
 $1+2+3-4+5 \Leftrightarrow (((1+2)+3)-4)+5$
 3. äußere Klammer kann man weglassen $(1+2) \Leftrightarrow 1+2$
- Computer wandeln Infix zuerst in Prefix Notation um
 1. weggelassene Klammer weider einfüger

2. Operatorensymbol durch Funktionsnamen ersetzen und an Prefix-Position verschieben

$$1 + 2 + 3 * 4 / (1 + 5) - 2$$

$$(((1 + 2) + ((3 * 4) / (1 + 5))) - 2)$$

$$sub(add(add(1, 2), div(mul(3, 4), add(1, 5))), 2)$$

$$sub(add(3, div(12, 6)), 2)$$

$$sub(add(3, 2), 2)$$

$$sub(5, 2)$$

$$2$$