Informatik I
Spezifikation, Algorithmus, Programm

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

1. Spezifikation
   - Vor dem Schreiben eines Programmes muß das Problem zunächst spezifiziert werden
2. Algorithmus
   - Ablauf von Aktionen, die das Problem schrittweise lösen
3. Programm
   - konkrete Formulierung des Algorithmus in einer Programmiersprache

> As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become a gigantic problem.
>
> Edsger W. Dijkstra (1930-2002)
> ACM Turing Award Lecture 1972

Definition Spezifikation

- vollständige, detaillierte und unzweideutige Problembeschreibung
- dabei heißt
  - Vollständig: alle Anforderungen und alle relevanten Rahmenbedingungen sind angegeben
  - Detailliert: Angabe aller zugelassenen Hilfsmittel, Grundaktionen
  - Unzweideutig: klare Kriterien, wann eine vorgeschlagene Lösung akzeptabel ist

- Eingabespezifikation: Es muß genau spezifiziert sein, welche Eingabegrößen erforderlich sind und welchen Anforderungen diese Größen genügen müssen, damit das Verfahren funktioniert
- Ausgabespezifikation: Es muß genau spezifiziert sein, welche Ausgabegrößen (Resultate) mit welchen Eigenschaften berechnet werden
- Sprachgebrauch manchmal:
  - Zusicherung, die der Algorithmus für die erarbeiteten Ergebnisse gibt
  - Vorbedingungen, die der Algo von den eingegeben Daten fordert

Beispiel

- Kaffee kochen
  - Es soll eine Thermoskanne Kaffee bereitgestellt werden
- Offene Fragen
  - Vollständigkeit
    - darf eine Kaffeemaschine benutzt werden?
  - Detailliertheit
    - welche Aktionen sind sind erlaubt?
    - Filter einlegen, Kaffeepulver einfüllen, etc.
  - Unzweideutigkeit
    - muß der Kaffeefilter anschließend in den Müll?

- ggT(M, N)
  - "Für beliebige Zahlen M und N berechne den größten gemeinsamen Teiler ggT(M,N), also die größte Zahl, die sowohl M als auch N teilt"
- offene Fragen
  - Vollständigkeit
    - Welche Zahlen M,N sind zugelassen?
    - Positive Zahlen oder auch negative oder rationale? Ist 0 erlaubt?
  - Detailliertheit
    - Welche Operationen sind erlaubt?
    - '+', '-' oder auch div und mod?
  - Unzweideutigkeit
    - Was heißt berechnen?
    - Soll das Ergebnis ausgedruckt werden? Soll es in einer Variablen gespeichert werden?
    - Wo kommen die beiden Zahlen M und N her?

Beschreibung von Spezifikationen

- Präziseste Sprache zur Spezifikation ist die mathematische Logik
- In der Praxis werden häufig weniger formale Beschreibungen in natürlicher Sprache verwendet (Pflichtenhefte)
  - Häufig umfangreich, mehrdeutig, inkonsistent
  - Ist die Ausgabespezifikation das, was der Kunde wollte?
  - Bekommt der Kunde von seinen "Zulieferern" das, was die Eingabespezifikation sagt?
  - Tut der Algorithmus das, was in der Spezifikation steht?
  - Tut der Algo das, was der Kunde wollte?
  - Siehe Software-Engineering

Formale Spezifikation

- Paar *P* und *Q* von logischen Aussagen (Aussagenlogik!)
- *P* : Vorbedingung / Eingabespezifikation
  - alle relevanten Eigenschaften, die vor Ausführung des Algorithmus gelten
    - Kaffeemaschine aus, Wassertank leer, ...
- *Q* : Nachbedingung / Zusicherung
  - alle relevanten Eigenschaften, die nach Ausführung des Algorithmus gelten
    - Thermoskanne gefüllt, Kaffeefilter im Biomüll, ...
- Schreibweise: { aussagenlogische Formeln ... }

Beispiel

- ggT(M,N):
  - Vorbedingung:
    {*M* und *N* sind ganze Zahlen mit $0 < M < 32767$ und $0 < N < 32767$}
  - Nachbedingung:
    {*z* ist Teiler von *M* und *N* und für jede andere Zahl *z'*, die auch *M* und *N* teilt, gilt $z' \le z$}
- Häufig benutzte Konvention
  - Großbuchstaben bezeichnen Eingabedaten, also durch den Algorithmus nicht veränderbare Größen
  - Kleinbuchstaben bezeichnen Variablen, also veränderbare Größen

Algorithmen

- Definition Algorithmus:

  geordnete Menge eindeutiger, diskreter, durchführbarer Schritte, die die geg. Vorbedingung in endlich vielen Schritten die geg. Nachbedingung überführen.

- also: eine präzise Vorschrift, wie in endlich vielen Schritten ein Problem gelöst werden kann
- Beispiel: Konvertierung zwischen Dezimal ↔ Binär

Herkunft des Wortes „Algorithmus"

Muhammad ibn Musa abu Djafar al-Choresmi, ca. 780-850 n. Chr.

- arabischer Mathematiker, geboren in Choresmien (heute Usbekistan)
- lebte und wirkte in Bagdad im "Haus der Weisheit"
- war beteiligt an der Übersetzung der Werke griechischer Mathematiker ins Arabische
- schrieb ein "Kurzgefasstes Lehrbuch für die Berechnung durch Vergleich und Reduktion" (Lehrbuch zum Rechnen mit Dezimalzahlen)
- lateinische Übersetzung dieses Buches ("liber algorismi") kam durch Kreuzfahrer nach Europa
- verfasste auch ein Buch mit dem Titel "Al-Mukhtasar fi Hisab al-Jabr wa l-Muqabala"

Algebra

Algorithmus

Eine sehr alte Beschreibung eines Algorithmus'

Verdoppeln einer ganzen Zahl nach Adam Riese (16. Jahrhundert)

Duplieren
Lehret wie du ein zahl zweyfaltigen solt. Thu ihm also:
Schreib die zahl vor dich /mach ein Linien darunter/ heb an zu forderst / Duplir die erste Figur. Kompt ein zahl die du mit einer Figur schreiben magst / so setz die unden. Wo mit zweyen / schreib die erste / Die ander bahalt im sinn. Darnach duplir die ander / und gib darzu / das du behalten hast / und schreib abermals die erste Figur / wo zwo vorhanden / und duplir fort biß zur letzten / die schreibe gantz auß / als folgende Exempel aufweisen.

41232	98765	68704
82464	197530	137408

Adam Riesen, Rechenbuch, 1574

Algorithmus und Spezifikation

- Spezifikation $\{P\}$ $\{Q\}$ beschreibt Problem
- Algorithmus A sei Lösung des Problems
- Schreibweise: $\{P\}$ A $\{Q\}$

Wird der Algorithmus A in einer Situation gestartet, in der P gilt, dann gilt nach Beendigung des Algorithmus Q

- Algorithmenentwurf ist wie Gleichung mit einer Unbekannten X: {P} X {Q}
  - gesucht: Algorithmus X, der P in Q überführt
- Achtung: nicht jede Spezifikation hat eine Lösung
  - Beispiel: {M<0} X {x ∈ R und x=log(M)}
- falls eine Lösung (Algo) X existiert, gibt es unendlich viele
  - Beispiel: {M>0} X {x=2*M}
    - Addiere M und M
    - Multipliziere M mit 2
    - Berechne $(M+2)^2-M^2-4$
  - Ziel: den "kürzesten" Algorithmus zu finden →
    - Praktische Informatik: Algorithmenentwurf
    - Theoretische Informatik: Komplexität dieses Algo / minimale Komplexität

- wir gehen immer von der Terminierung eines Algorithmus A aus
  - terminiert A nicht, ist {P}A{Q} trivialerweise erfüllt
    - ("... gilt nach Beendigung des Algorithmus Q")
  - insbesondere kann {P}A{false} nur dann erfüllt werden, wenn A nicht terminiert
- ein nicht terminierender Algorithmus löst aber kein Problem

Begriffe in der Definition für "Algorithmus"

- Ausführung des Algorithmus erfolgt in diskreten Schritten
  - einzelne, einfache Elementaraktionen
  - Welche dies sind hängt vom sog. Algorithmischen Modell ab
    - Maschineninstruktionen?
    - Hochsprachliche Konstruktionen? (log(x) ?)
    - Bitkomplexität? (Addition beliebig langer Zahlen)
- Endliche Beschreibung: das Verfahren muß in einem endlichen Text vollständig beschrieben sein

- Geordnete Menge
  - Reihenfolge der Schritte genügt gewissen Regeln
  - Schritte müssen nicht unbedingt nacheinander ausgeführt werden
    - es ist erlaubt, daß mehrere Schritte parallel von verschiedenen Einheiten ausgeführt werden (parallele oder verteilte Algorithmen)
    - bei einer einzigen aktiven Einheit allerdings gibt es einen ersten, zweiten, etc. Schritt (sequentielle Algorithmen)
- Determiniertheit: Der Verfahrensablauf ist zu jedem Zeitpunkt fest vorgeschrieben
  - zum Zeitpunkt der Ausführung eines Schrittes muß eindeutig und vollständig festgelegt sein, was zu tun ist
  - Gleiche Eingaben sollen zum gleichen Ablauf und Ergebnis führen
  - Ausnahme: randomisierte Algorithmen, deren Ablauf von (mathematisch bestimmten) Zufallsgrößen abhängt
  - Weitere Ausnahme: Parallelisierung

- Effektivität: Jeder Schritt des Verfahrens muß effektiv (d.h. tatsächlich) „mechanisch" ausführbar sein
  - Bem.: „Effektivität" ist nicht zu verwechseln mit „Effizienz" („Wirtschaftlichkeit")
- Nicht erlaubt:

1. Erstelle eine Liste aller positiven ungeraden Zahlen
2. Sortiere die Liste in absteigender Reihenfolge
3. Gib das letzte Element der Liste aus

⇒Ergebnis wäre 1, aber Schritte sind nicht ausführbar

- Terminierung: Der Algorithmus hält nach endlich vielen Schritten mit einem Ergebnis an — sofern die Eingaben der Eingabespezifikation genügt haben!
- nicht erlaubt: Schrittfolgen, die nie ein sinnvolles Ergebnis liefern:
  - Starte mit Zahl 0
  - Addiere 2
  - Falls Ergebnis ungerade, halte an, sonst weiter bei Schritt 2.
- Sonderfälle:
  - Betriebssystem, Datenbanksystem, ... (Ausgaben fortwährend)
  - Bei numerischen Algorithmen (z.B. Nullstellenberechnung), möchte man häufig offen lassen, wann genau die Berechnung in der Praxis abbrechen soll

- es gibt Algorithmen(?), von denen man nicht *weiß*, ob sie in jeder Situation terminieren
- Beispiel: Ulam-Funktion oder Syrakus-Problem
  1. Starte mit beliebiger positiver ganzer Zahl n
  2. Falls n ungerade, ersetze n durch 3*n+1, sonst ersetze n durch n/2.
  3. Fahre fort, bis n = 1 erreicht ist.
  - nicht bekannt, ob folgender Algo für beliebige Startwerte terminiert
- Beispielläufe:

5, 16, 8, 4, 2, 1

6, 3, 10, 5, 16, 8, 4, 2, 1

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Korrektheit

- Partielle Korrektheit: Jedes berechnete Ergebnis genügt der Ausgabespezifikation, sofern die Eingaben der Eingabespezifikation genügt haben
  - d.h. Ein- und Ausgabeparameter genügen im Falle der Terminierung immer den Spezifikationen
  - Achtung: partielle Korrektheit sagt nichts über die Terminierung!
- Totale Korrektheit: Ein Rechenverfahren ist ein total korrekter Algorithmus, wenn es partiell korrekt ist und für jede Eingabe, die der Eingabespezifikation genügt, terminiert.
  - Unterschiedliche Beweisverfahren für beide Fälle notwendig
  - Manchmal ist es sinnvoll, auf die Anforderung der Terminierung zu verzichten

Ausnahmen

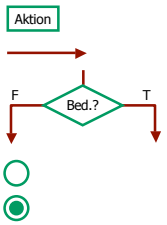
- ... bestätigen die Regel: nach unserer Begriffsbestimmung gäbe es also keine
  - nicht-terminierende Algos (Algo liefert manchmal, oder für manche Eingaben, kein Ergebnis, kommt aber so selten vor, daß man in Kauf nimmt)
  - nicht-deterministische, randomisierte Algos (Ablauf und Ergebnis kann, bei gleicher Eingabe, jedesmal anders sein)
  - ...
- Diese Begriffe werden aber durchaus verwendet!
  - Methode erfüllt alle Anforderungen an einen Algorithmus, bis auf die mit „nicht" gekennzeichnete

Darstellung / Beschreibung von Algorithmen

- zahlreiche Methoden, Algorithmen darzustellen, z.B.
  - informelle Beschreibung , "Roman" (s. Adam Riese)
  - Flußdiagramme
  - Struktogramme
  - Pseudocode
  - UML-Diagramme
  - Programmcode
  - ...

Flußdiagramm

- Flußdiagramm
  - grafische Notation für Algorithmen
  - leicht verständlich
  - kann bei umfangreichen Algorithmen unübersichtlich werden



auszuführende Elementaraktion

Reihenfolge der Elementaroperationen

Test: Bedingung im Karo.
Ist sie erfüllt, weiter bei T,
sonst weiter bei F

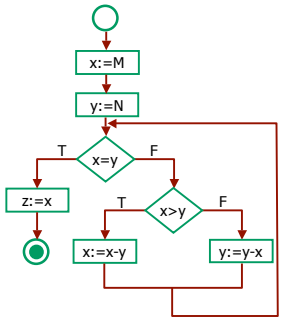
Beginn der Ausführung des Algorithmus

Ende der Ausführung des Algorithmus

Beispiel: ggT(M, N)

- Eigenschaften von T=ggT(M, N)
  1. Fall M=N: ggT(M, M) = M
  2. Fall M>N: ggT(M, N) = ggT(M-N, N)
     - Denn
       für jedes T mit M=A*T und N=B*T gilt
       M-N = (A-B)*T
  3. Fall M<N: ggT(M, N) = ggT(N, M)
- Dies ist schon eine Beschreibung des Algorithmus' in Pseudocode!

ggT(M, N) als Flußdiagramm



Das algorithmische Modell

- Das algorithmische Modell definiert, welches genau die Elementaren Aktionen sind.
- Variablen lesen (= Benutzung der Variablennamen im Programm)
- Variablen schreiben (Zuweisung eines Wertes zu einer Variablen)
- Werte verknüpfen (Benutzung von Operatoren oder Funktionen)
- Beispiele:
  - x = x - y
  - x = M mod N
  - y = sin(x)

- Werte in Variablen oder Ausdrücken testen (demnächst)
- Sprünge, Verzweigungen, Funktionsaufrufe
- Konstante "Menge" von Daten ausgeben

Vom Algorithmus zum Programm

- Programm := Formulierung eines Algorithmus in einer für den Computer verständlichen (Programmier-)Sprache
- Unterschiedlichste Programmiersprachen
  - imperative Sprachen
    - befehlsorientiert
    - z.B. BASIC, PASCAL, C, C++, Java
  - deklarative Programmiersprachen
    - mathematische / logische Formulierung eines Problems
    - System versucht mit Inferenzregeln, eine Tatsache zu beweisen
    - z.B. PROLOG, ...
  - objektorientierte Programmiersprachen
    - Objekte: Daten und Methoden zur Manipulation der Daten (genauer s.u.)
    - z.B. Smalltalk, C++, Java, ...
  - Funktionale Sprachen
    - Wesentliches Element ist der Funktionsaufruf und die Liste
    - Z.B. Lisp, ML, Haskell
  - Hybride
  - es kommen keine weitere Paradigmen hinzu (?)

Pseudocode

- Siehe ggT
- Hilfsmittel zur Formulierung eines Algorithmus
- Syntax ist nicht genau festgelegt
  - hier: PASCAL ähnlich
  - sollte intuitiv zugänglich sein
- in Entwurfsphase auch verbale Beschreibung von Blöcken möglich
  - z.B. "Daten einlesen" oder "alphabetisch sortieren"
- leicht in imperative Programmiersprache übertragbar

Beispiel ggT

t = ggT (m, n)
Input: m, n ∈ N, m, n > 0
Output: t ∈ N, t > 0

```
while m ≠ n:
    if m < n:
        swap m,n
    m ← m − n
output m
```

```
solange m ≠ n:
    falls m < n:
        swap m,n
    m ← m − n
drucke m
```


Was ist ein Programm?

- Erinnerung: Algorithmenbegriff beinhaltet ein „effektiv" („mechanisch") durchführbar
- Entwicklung von Algorithmen kann weitgehend ohne eine konkrete Maschine (=Programmiersprache) erfolgen

Programm [griech.>schriftliche Bekanntmachung<, >Tagesordnung<] Ankündigung, Plan, Ziel; festgelegte Folge (z.B. bei einer Veranstaltung oder im Fernsehen und Rundfunk); Ankündigungszettel; Warenangebot […]

vollständige Anweisung für automatisch gesteuerte Maschinen wie Werkzeug- oder Textilmaschinen, Rechenanlagen u.a., nach der die zur Bearbeitung einer Aufgabe erforderlichen Arbeitsgänge ablaufen sollen

Programmierung, Datenverarbeitung: das Erstellen eines Programms.

Programmiersprache, Datenverarbeitung: zum Abfassen von Programmen für digitale Rechenanlagen geschaffene Sprache.

dtv-Lexikon in 20 Bänden, Bd. 14, 1999

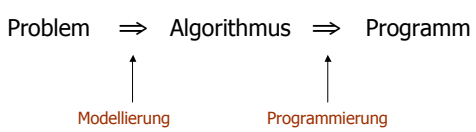
„Definition" des Begriffs 'Programm' in informatiknahen Quellen:

Um einen Algorithmus in einer von einer Maschine ausführbaren Form beschreiben zu können, verwendet man eine formale Sprache. Eine solche Beschreibung eines Algorithmus' heißt ein Programm, die formale Sprache eine Programmiersprache.

Manfrad Broy: *Informatik, Teil I*, Springer, 1992

- Bemerkungen
  - Moderne Programmiersprachen sind so entworfen, daß Programme auf verschiedenen Computern ablaufen können. Es wird von speziellen Eigenschaften der Computer i.A. abstrahiert
  - Eigenschaften heutiger Computer fließen auch in das Design von Programmiersprachen mit ein
    - Programme sollen nicht nur effektiv durchführbar sein, sondern auch effizient!

Zusammenhänge



"Algorithmisches Denken"

- Problem: Muffins backen
- Rezept:

```
Muffins:
Verrühre 200g Butter, 200g Zucker, 4 Eier 1 Pck
Backpuler, 1 Pck Vanillezucker, 250 g Mehl, 3 Eßl.
Rum und drei mittelgroße geschälte und zerteilte
Äpfel; fülle den Teig in Muffinförmchen; backe bei
175-200 Grad für etwa 30 min; bestäube die Muffins
mit etwas Puderzucker
```

- Aufgabe:
  - Apfelzeit berücksichtigen
- Unser Rezept ist von einer sehr einfachen Form, denn es besteht lediglich aus einer Aneinanderreihung einfacher elementarer Anweisungen. Komplexere Algorithmen können weitere Strukturen enthalten.

- Apfelzeit ist eine Bedingung (ist nicht immer erfüllt) → *bedingte Anweisungen*:

```
Muffins:
Verrühre 200 g Butter, 200 g Zucker,4 Eier,1 Pck
        Backpulver,1 Pck Vanillezucker,250 g Mehl,3
        Eßl. Rum;
falls es Apfelzeit ist
        füge drei mittelgroße geschälte und
             zerteilte Äpfel hinzu;
sonst
        füge 100 g Schokoladenstückchen hinzu;
fülle den Teig in Muffinförmchen;
backe bei 175-200 Grad für etwa 30 min;
bestäube die Muffins mit etwas Puderzucker
```


- Backzeit implementieren → *Wiederholungen (Schleifen)*:

```
Muffins:
Verrühre 200 g Butter, 200 g Zucker,4 Eier,1 Pck
       Backpulver,1 Pck Vanillezucker,250 g Mehl,
       3 Eßl. Rum;
falls es Apfelzeit ist
         füge drei mittelgroße geschälte und
              zerteilte Äpfel hinzu;
sonst
         füge 100 g Schokoladenstückchen hinzu;
fülle den Teig in Muffinförmchen;
solange die Muffins noch nicht gar sind
         backe bei 175-200 Grad;
bestäube die Muffins mit etwas Puderzucker
```


- Anzahl Personen berücksichtigen → Algorithmus muß man parametrisieren durch Variablen, die von außen gesetzt werden:

```
Muffins:
Frage nach, wieviele Personen da sind;
Speichere die Antwort in der Variablen x;
y=x/4;
Verrühre y*200 g Butter, y*200 g Zucker,y*4 Eier,y Pck
       Backpulver,y Pck Vanillezucker,y*250 g Mehl,
       y*3 Eßl. Rum;
falls es Apfelzeit ist
         füge y*drei mittelgroße geschälte und zerteilte
         Äpfel hinzu;
sonst
         füge y*100 g Schokoladenstückchen hinzu;
fülle den Teig in Muffinförmchen;
solange die Muffins noch nicht gar sind
         backe bei 175-200 Grad;
bestäube die Muffins mit etwas Puderzucker
```


- Strukturierung für Übersichtlichkeit und Wiederverwendbarkeit von Teilen des Rezeptes → Algorithmus aufteilen auf mehrere *Teilprogramme*:

```
Muffins:
Frage nach, wieviele Personen da sind;
Speichere die Antwort in der Variablen x;
y=x/4;
Verrühre y*200 g Butter, y*200 g Zucker,y*4 Eier,
       y Pck Backpulver,y Pck Vanillezucker,y*250 g Mehl
       y*3 Eßl.Rum;
falls es Apfelzeit ist
       füge y*drei mittelgroße geschälte und zerteilte
          Äpfel hinzu;
sonst
       füge y*100 g Schokoladenstückchen hinzu;
fülle den Teig in Muffinförmchen;
solange Teilprogramm Stäbchentest liefert „nicht fertig"
         backe bei 175-200 Grad;
bestäube die Muffins mit etwas Puderzucker
```



```
Teilprogramm Stäbchentest:
stecke ein Holzstäbchen in das erste Muffin;
falls Teig kleben bleibt
       liefere "noch nicht fertig";
sonst
       liefere "fertig"
```

Der Algorithmus ist jetzt schon ganz schön kompliziert geworden. Er ist in gewisser Hinsicht noch einfach, denn es handelt sich um eine *sequentielle* Folge von Anweisungen. Das heißt, ein einziger Koch ist hier am Werk, der die Anweisungen nacheinander ausführt. Alternativen sind *parallele* Programme, in denen mehrere Anweisungen gleichzeitig durch verschiedene Prozessoren ausgeführt werden können. Beim Backen könnte etwa der Küchenjunge schonmal die Äpfel schälen, während der Chefkoch den Teig zubereitet. Die Aktionen müssen in parallelen Programmen geeignet *synchronisiert*, d.h. abgestimmt werden. Etwa der Küchenjunge muß mit den Äpfeln fertig sein, bevor der Chefkoch sie zum Teig fügen kann.

Anwendung von Moore's Law auf Algorithmen

- Moore's Law: Die Anzahl Transistoren pro Chip-Fläche verdoppelt sich alle 2 Jahre
  - Konsequenz: Die Leistungsfähigkeit (Geschwindigkeit) verdoppelt sich alle 2 Jahre
  - Speicherkapazität verdoppelt sich alle 2 Jahre
  - Problemgrößen verdoppeln sich alle 2 Jahre
- Konsequenz: auf der neuen Maschine braucht ein Algorithmus länger

  - "Beweis":
    - Algorithmus benötigt Zeit $N \log N$ auf alter Maschine
    - Auf neuer Maschine: $(2N \log 2N) / 2 = N \log 2N = N \log N + N$
- Konsequenz: Entwicklung schneller Algorithmen lohnt sich!