Einführung:
Vom Algorithmus zum Programm

1.1 Vom Algorithmus zum Programm
1.2 Programmiersprachen
1.3 Korrektheit, Komplexität und Entscheidbarkeit
1.4 Hard- und Software-Grundlagen

Algorithmusbegriff

Ein Algorithmus ist eine „Berechnungsvorschrift“:

- Die Berechnungsvorschrift ist durch einen endlichen Text kodiert.
- Sie beschreibt die auszuführenden Berechnungen „hinreichend präzise“.
- Die Berechnungen sind aus „elementaren“ Operationen aufgebaut und
- besitzen Aus- und evtl. Eingabewerte.

Hierbei handelt es ich um eine sog. intuitive Definition. In der Informatik wird auch eine formale Definition benötigt, zum Beispiel zum Nachweis, dass für ein bestimmtes Problem kein Algorithmus existiert.

Eigenschaften von Algorithmen

- Algorithmen sollen in der Regel terminieren, d. h. bei jeder Eingabe irgendwann zu einem Ende führen. Es gibt Ausnahmen: z. B. Betriebssysteme oder sogenannte „reaktive Systeme“.

- Einen Algorithmus nennt man deterministisch, wenn er bei gleichen Eingabedaten stets die gleiche Berechnung ausführt.

- Ein Algorithmus heißt determiniert, wenn er bei gleichen Eingabedaten stets die gleichen Ausgabedaten liefert.

Programm und Programmiersprache

Ein Programm ist die Formulierung eines Algorithmus und seiner Datenbereiche in einer Programmiersprache. Eine Programmiersprache erlaubt es, Algorithmen präzise zu beschreiben. Insbesondere legt eine Programmiersprache

- die elementaren Operationen,
- die Möglichkeiten zu ihrer Kombination und
- die zulässigen Datenbereiche

eindeutig fest. Unter „programmieren“ versteht man den Vorgang des Erstellens eines Programms.

Grundlegende Aspekte der Algorithmenentwicklung

Lösbarkeit/Entscheidbarkeit

Gibt es für mein Problem einen Algorithmus?

Korrektheit

Löst mein Algorithmus mein Problem?

Komplexität

Mit welchem Aufwand (z. B. Rechenzeit, Speicherplatz) löst mein Algorithmus das Problem?

Paradigmen zur Algorithmenbeschreibung

In einem imperativen Algorithmus gibt es Variable, die verschiedene Werte annehmen können. Die Menge aller Variablen und ihrer Werte sowie der Programmzähler beschreiben den Zustand zu einem bestimmten Zeitpunkt. Ein Algorithmus bewirkt eine Zustandstransformation.

Ein funktionaler Algorithmus formuliert die Berechnung durch Funktionen. Die Funktionen können rekursiv sein; auch gibt es Funktionen höherer Ordnung.

In einem objektorientierten Algorithmus werden Datenstrukturen und Methoden zu einer Klasse zusammengefasst. Von jeder Klasse können Objekte gemäß der Datenstruktur erstellt und über die Methoden manipuliert werden.

Ein logischer (deduktiver) Algorithmus führt Berechnungen durch, indem er aus Fakten und Regeln durch Ableitungen in einem logischem Kalkül Ziele beweist.

Beispiel: Algorithmus von Euklid

Der folgende, in einer imperativen Programmiersprache formulierte

Algorithmus von Euklid

berechnet den größten gemeinsamen Teiler der Zahlen $x, y \in \mathbb{N}$ mit $x \geq 0$ und $y > 0$:

```
a := x;
b := y;
while b # 0
do r := a mod b;
   a := b;
   b := r
od
```

Anschließend gilt $a = \mathrm{ggT}(x, y)$.

Beispiel: Algorithmus von Euklid

Variable	z_0	z_1	z_2	z_5	z_8	z_{11}	z_{14}
r	–	–	–	36	16	4	0
a	–	36	36	52	36	16	4
b	–	–	52	36	16	4	0

$$\mathrm{ggT}(36, 52) = 4$$

Durchlaufene Zustände: $z_0, z_1, z_2, \ldots, z_{14}$

Zustandstransformation: $z_0 \longmapsto z_{14}$

Datenstrukturen

Programmiersprachen bieten die Möglichkeit, aus elementaren Datenbereichen mithilfe von Konstruktoren komplexe Datenbereiche aufzubauen. Datenbereiche werden häufig Datenstrukturen genannt.

- elementare Datenstrukturen:
  - boolean, char, cardinal, integer, real, enumeration
- Konstruktoren:
  - array (Feld), record (Satz), set (Menge), pointer (Zeiger)
  - Zeiger ermöglichen rekursive Datenstrukturen wie Listen, Bäume und Graphen.

Natürliche und künstliche Sprachen

- Sprache ist ein sich stets weiterentwickelndes, komplexes System von Lauten und Zeichen zum Zwecke der Kommunikation. Es werden natürliche und künstliche Sprachen unterschieden.

- Natürliche Sprachen sind historisch gewachsen. Hierzu zählen z. B. Deutsch, Englisch und Französisch. Sie sind Ausdruck menschlichen Denkens, Fühlens und Wollens und weisen im Unterschied zu künstlichen Sprachen Mehrdeutigkeiten auf.

- Künstliche Sprachen sind Zeichensysteme, die der Verständigung in einem eng begrenzten Fachgebiets dienen, zum Beispiel Programmiersprachen. Sprachen wie Esperanto sind ebenfalls künstliche Sprachen, die sich durch leichtere Schreibung und Grammatik gegenüber natürlichen Sprachen auszeichnen.

aus *Basiswissen Deutsch*, Dudenverlag

Gegenstände der Informatik

Um Objekte mit Rechensystemen zu behandeln, müssen sie in eindeutigen – also künstlichen – Sprachen beschrieben werden. Einige Beispiele sollen dies verdeutlichen:

- Algorithmen: Programmiersprachen (Java)
- Dokumente: Markup-Sprachen (Html, XML), Seitenbeschreibungssprachen (Postscript)
- Modelle, Systeme: Modellierungssprachen (UML)
- Spezifikationen: Spezifikationssprachen (Z, VDM-SL)
- Datenbanken: Anfragesprachen (SQL)

Folgerung

- In der Informatik hat man es mit einer Vielzahl von künstlichen Sprachen zu tun.
- Sie alle beschreiben Sachverhalte in einem relativ kleinen Kontext,
- dafür aber (hoffentlich) präzise, widerspruchsfrei und vollständig.

In dieser Vorlesung betrachten wir die Programmiersprache Java.
In anderen Veranstaltungen (z. B. „Programmieren für Fortgeschrittene“)
lernen Sie weitere Sprach(klass)en kennen.

Einführung: Programmiersprachen

1.1 Vom Algorithmus zum Programm
1.2 Programmiersprachen
1.3 Korrektheit, Komplexität und Entscheidbarkeit
1.4 Hard- und Software-Grundlagen

Entwicklung der Programmiersprachen

Edsger W. Dijkstra:

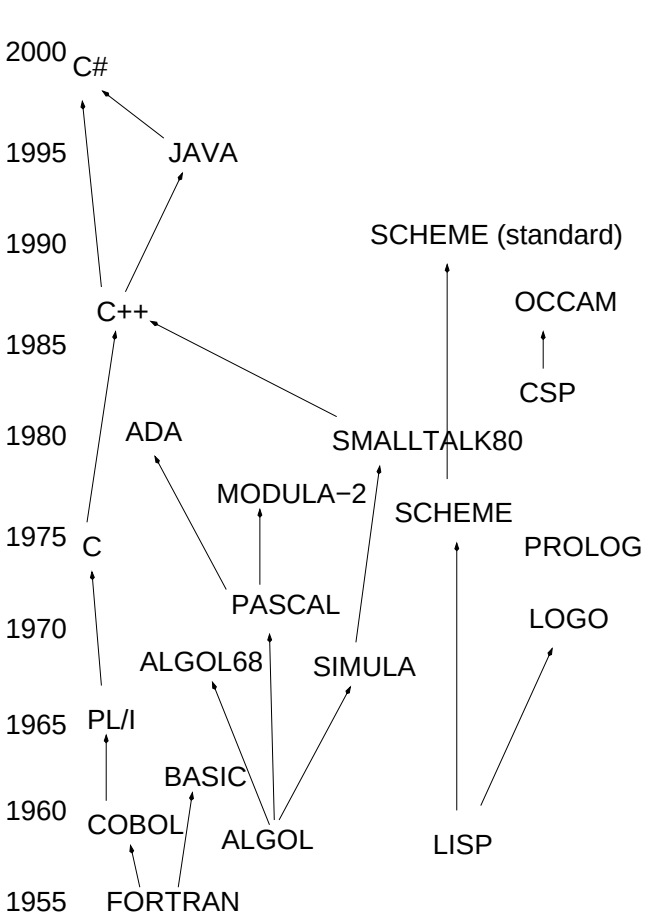
> „Jeder Programmierer weiß, dass es nur eine einzig wahre Programmiersprache gibt. Jede Woche eine neue.“

A. Weinert: *Java für Ingenieure*, 2001, Seite 7:

> „Die Zahl der Programmiersprachen, die die Informatik in den letzten fünfzig Jahren hervorgebracht hat, ist Legion. Ernst zu nehmende Schätzungen sprechen von mehr als 20 000.“

Wenn Weinerts Schätzung zutrifft, sind es 7,7 Programmiersprachen pro Woche!

Entwicklung der Programmiersprachen



Programmiersprachen in der Informatikausbildung

- Algol
- Algol68
- Modula-2
- Modula-2/Scheme
- Java/Scheme
- Java

Definition von Programmiersprachen

Die Lexik einer Programmiersprache bestimmt die textuellen Grundbausteine der Programme. Solche Bausteine sind etwa Schlüsselwörter und Bezeichner. Sie werden z. B. durch Aufzählung oder reguläre Ausdrücke bestimmt.

Die Syntax einer Programmiersprache beschreibt, wie aus den Grundbausteinen vollständige Programme gebildet werden können. In den meisten Fällen wird die Syntax einer Programmiersprache durch eine kontextfreie Grammatik festgelegt.

Die Bedeutung der syntaktisch korrekten Programme ist durch die Semantik der Sprache gegeben. Sie kann beispielsweise mithilfe von Zustandsfolgen (operationelle Semantik) oder durch Funktionen, die den syntaktischen Einheiten zugeordnet sind (denotationale Semantik), definiert werden.

Die Pragmatik einer Programmiersprache untersucht ihre Anwendbarkeit und Nützlichkeit. Sie gehört nicht zur Definition der Sprache.

Definition von Programmiersprachen

Lexik:

Bezeichner = Buchstabe · { Buchstabe, Ziffer }*
Schlüsselwörter: `while`, `do`, `od`

Syntax:

<Anweisungsfolge>	::=	<Anweisung> ; <Anweisungsfolge> | <Anweisung>
<Anweisung>	::=	<Zuweisung> | <While-Anweisung> | ...
<Zuweisung>	::=	<Bezeichner> := <arithmetischer Ausdruck>
<While-Anweisung>	::=	while <logischer Ausdruck> do <Anweisungsfolge> od

(Operationelle) Semantik:

Eine (partielle) Funktion f, die Zustände auf Zustände abbildet: $f(z_0) = z_{14}$

Klassifikation der Programmiersprachen

Die Programmiersprachen lassen sich grob in drei Klassen einteilen:

- Maschinensprachen
  Bits und Bytes, für den menschlichen Leser kaum verständlich

- Maschinenorientierte Sprachen (Assembler)
  stellen die Befehle in einem Mnemo-Code dar

  ```
  ADDIC 23, R0
  STO R0, #12004
  ```

- Problemorientierte Sprachen
  imperative, funktionale, objektorientierte, deklarative Sprachen, Spezialsprachen

Ein Computer versteht nur Maschinensprachen!

Implementierung von Programmiersprachen

Compiler übersetzen Quellprogramme aus problemorientierten Sprachen in äquivalente Zielprogramme in Maschinensprachen:

```
cc -o prog prog.c
prog input output
```

Interpreter lesen das Programm zusammen mit den Eingabedaten ein und führen es aus:

```
scm prog.scm input output
```

Mischverfahren übersetzen das Programm zunächst mit einem Compiler in eine Zwischensprache. Das übersetzte Programm wird anschließend interpretiert:

```
javac prog.java
java prog input output
```

Implementierung von Programmiersprachen

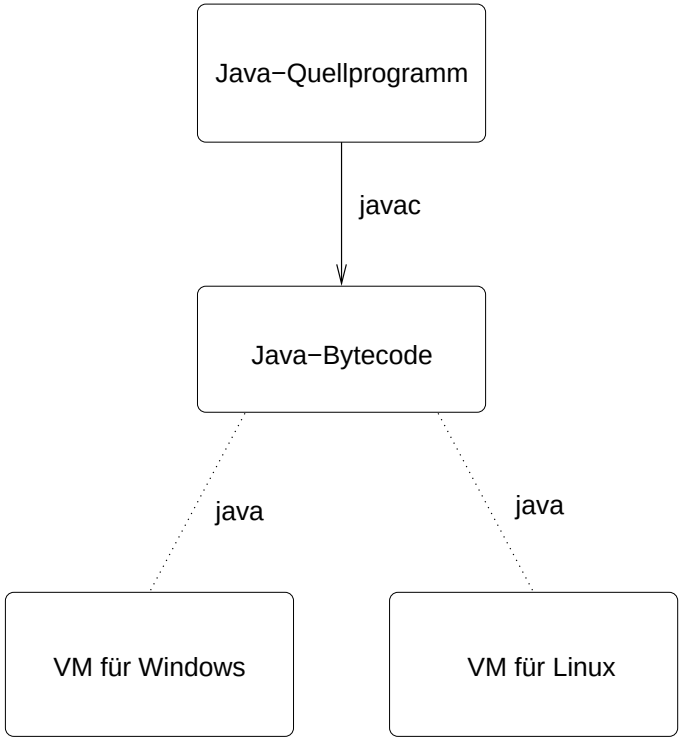
Interpreter müssen das Programm bei jedem Lauf erneut analysieren. Dies bedeutet einen gewissen Effizienzverlust.

Typisch, aber nicht zwingend:

- Compiler: C
- Interpreter: Scheme
- Mischverfahren: Java
- Compiler und Interpreter: Haskell

$\text{Compiler}_1 \neq \text{Compiler}_2$ $\quad$ $\text{Interpreter}_1 \neq \text{Interpreter}_2$

Verarbeitung von Java-Programmen



- Zuerst wird ein Quellprogramm vom Compiler in Bytecode übersetzt.
- Im zweiten Schritt wird der Bytecode vom Interpreter ausgeführt. Der Bytecode kann als Maschinencode der sogenannten virtuellen Java-Maschine (JVM) angesehen werden. Bytecode ist portabel.
- Der Compiler ist maschinenunabhängig, der Interpreter muss für jede Plattform neu entwickelt werden.

Verarbeitung von Java-Programmen

- Interpretierter Code ist langsamer in der Ausführung als kompilierter Code, selbst wenn dieser als Bytecode vorliegt.

- Prinzipiell können Java-Programme auch in Maschinensprache übersetzt werden. Dann geht allerdings die Portierbarkeit verloren.

- Eine Alternativlösung bieten Just-in-Time-Compiler (JIT). Ein JIT ist ein Programm, das den Bytecode einzelner Methoden während der Ausführung in Maschinencode der jeweiligen Plattform übersetzt. So kann die Methode beim nächsten Aufruf deutlich schneller ausgeführt werden. Vorteilhaft ist, dass der Bytecode nicht verändert wird und damit das übersetzte Programm portabel bleibt.

Einführung:
Korrektheit, Komplexität und Entscheidbarkeit

1.1 Vom Algorithmus zum Programm
1.2 Programmiersprachen
1.3 Korrektheit, Komplexität und Entscheidbarkeit
1.4 Hard- und Software-Grundlagen

Spezifikation, Korrektheit und Verifikation

Die Spezifikation beschreibt die Anforderungen an ein Softwaresystem in einer informellen, grafischen und/oder formalen Sprache. Eine Spezifikation sollte vollständig und widerspruchsfrei sein.

Ein Softwaresystem, das eine Spezifikation erfüllt, heißt korrekt bezüglich dieser Spezifikation. Man unterscheidet dabei zwischen partieller und totaler Korrektheit. Ein Programm nennt man partiell korrekt, wenn die Spezifikation erfüllt, die Terminierung von Programmläufen aber nicht notwendigerweise gewährleistet ist. Es heißt total korrekt, wenn zusätzlich die Terminierung sichergestellt ist.

Unter Verifikation versteht man den mathematischen Beweis der partiellen oder totalen Korrektheit eines Programms.

Korrektheit des Algorithmus von Euklid

Die Spezifikation besteht aus einer Vorbedingung und einer Nachbedingung:

Vorbedingung: $x > 0$ und $y \geq 0$
Nachbedingung: $a = \text{ggT}(x, y)$

Der Algorithmus von Euklid ist für Eingaben x und y mit $x > 0$ und $y \geq 0$ partiell und total korrekt. Die Variable a enthält nach Programmende den Wert des größten gemeinsamen Teilers von x und y.
Beweis unter Verwendung einer Schleifeninvarianten: s. Vorlesung.

Mit der Definition $\text{ggT}(0, 0) = 0$ ist der Algorithmus von Euklid für alle Werte x und y mit $x \geq 0$ und $y \geq 0$ partiell und total korrekt. Beweis: s. Vorlesung.

Test und Validierung

Der Test eines Programms ist der probeweise Ablauf des Programms. Damit der Test aussagekräftig ist, müssen die Eingabedaten sorgfältig ausgewählt werden. Ein Test kann nur die Anwesenheit von Fehlern, niemals aber deren Abwesenheit zeigen.

Als Validierung bezeichnet man den Test eines Softwaresystems unter Bedingungen, wie sie im späteren Einsatz herrschen werden. Auch wenn das zu erstellende Programm verifiziert wurde, kann auf eine Validierung nicht verzichtet werden, da ein mathematischer Nachweis der Korrektheit beispielsweise nichts über das Laufzeitverhalten des Programms oder die Auslastung von Leitungen aussagt.

Verifikation: verus – wahr, facere – machen
Validierung: validus – gesund, stark

Komplexität und O-Notation

- Unter Komplexität versteht man den Aufwand, den ein Algorithmus/Programm zur Lösung einer Aufgabe benötigt. Damit ist in den meisten Fällen der erforderliche Speicherplatz oder die Anzahl der durchgeführten Rechenschritte gemeint.

- Mathematisch wird die Komplexität eines Algorithmus/Programms in der Regel durch eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{R}$ beschrieben. Die Größenordnung einer solchen Funktion f wird häufig durch die sogenannte O-Notation nach oben abgeschätzt:

$$O(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid \exists c > 0, n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq f(n) \leq cg(n)\}$$

  für eine Funktion $g : \mathbb{N} \longrightarrow \mathbb{R}$.

Komplexität des Algorithmus von Euklid

Theorem. [G. Lamé, 1845] *Es seien x, y und n mit $x \geq 0$, $y \geq 0$ und $0 \leq x, y < n$ gegeben. Dann gilt: Der Algorithmus von Euklid benötigt höchstens*

$$f(n) := \left\lceil log_{\phi}\left(\sqrt{5}\ n\right)\right\rceil - 2$$

Divisionsschritte, wobei $\phi = \frac{1}{2}\left(1+\sqrt{5}\right)$ ist.

Beweis: s. Vorlesung.

Unter Verwendung der O-Notation erhalten wir

$$f(n) \in O(\log(n)).$$

Symbole zur Größenordnung von Funktionen

Es sei eine Funktion $g : \mathbb{N} \longrightarrow \mathbb{R}$ gegeben.

$$O(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid \exists c > 0, n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq f(n) \leq cg(n)\}$$

$$\Omega(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid \exists c > 0, n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq cg(n) \leq f(n)\}$$

$$\Theta(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid$$
$$\exists c_1 > 0, c_2 > 0, n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$o(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid \forall c > 0 \, \exists n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq f(n) < cg(n)\}$$

$$\omega(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid \forall c > 0 \, \exists n_0 > 0 \, \forall n \geq n_0 . \, 0 \leq cg(n) < f(n)\}$$

Diese Zeichen werden Landau-Symbole genannt. Eine Übersicht finden Sie auf der Web-Seite dieser Vorlesung. Ausführlich wird dieses Thema in den Veranstaltungen Algorithmen und Datenstrukturen und Diskrete Mathematik behandelt.

Symbole zur Größenordnung von Funktionen

- $3000n^2 + 7n + 23 \in \Theta(n^2)$ Man schreibt meistens: $3000n^2 + 7n + 23 = \Theta(n^2)$
- $3000n^2 + 7n + 23 \in O(n^2)$
- $3000n^2 + 7n + 23 \in \Omega(n^2)$
- $23 \in \Theta(1)$
- $a_n x^n + ... + a_1 x + a_0 \in \Theta(x^n)$
- $\Theta(\log_k(n)) = \Theta(\log_l(n))$
- $6n \log_2(n) + 8n + 12 \in \Theta(n \log(n))$

Entscheidbarkeit

- Entscheidbarkeit von Problemen: Gibt es zu jedem Problem einen Algorithmus, der es löst?

- Immer wieder kommt es vor, dass ein Computerprogramm plötzlich keine Reaktion mehr zeigt („abstürzt“ oder „sich aufhängt“). Dahinter verbirgt sich häufig ein Algorithmus, der für eine spezielle Eingabe nicht terminiert. Für kommerzielle Software kann das sehr teuer werden.

- Die Suche nach dem Grund der Nichtterminierung kann sich sehr schwierig gestalten. Daher liegt der Wunsch nahe, einen Algorithmus zu entwickeln, der beliebige Algorithmen auf Terminierung testet. Diese Aufgabenstellung heißt Halteproblem.

Halteproblem 1

Das Halteproblem ist unentscheidbar. Wir zeigen die Aussage indirekt:

- Annahme: Es gibt einen Algorithmus

  `HALT(algorithmus a, eingabe e)`,

  der für einen Algorithmus a und eine Eingabe e genau dann das Ergebnis `true` liefert, wenn a bei Eingabe von e terminiert.

- Der Algorithmus `TEST(algorithmus a)` sei definiert durch

  `TEST(algorithmus a): while HALT(a,a) { ... }`.

  Das heißt, `TEST(a)` terminiert genau dann nicht, falls a bei Eingabe von a terminiert.

Halteproblem 2

Zwei Fälle können eintreten:

- 1. Fall: Der Aufruf `HALT(TEST, TEST)` liefert `true`.

  In diesem Fall terminiert nach Definition von `HALT` der Aufruf `TEST(TEST)`. Hieraus folgt aus der Definition von `TEST`, dass der Aufruf `TEST(TEST)` nicht terminiert, ein Widerspruch.

- 2. Fall: Der Aufruf `HALT(TEST, TEST)` liefert `false`.

  In diesem Fall terminiert nach Definition von `HALT` der Aufruf `TEST(TEST)` nicht. Hieraus folgt aus der Definition von `TEST`, dass der Aufruf `TEST(TEST)` terminiert, ein Widerspruch.

Da in beiden Fällen ein Widerspruch auftritt, kann der Algorithmus `HALT` nicht existieren.

Halteproblem 3

Die beiden vorherigen Seiten

- Halteproblem 1 und
- Halteproblem 2

wurden dem Schulbuch

Peter Hubwieser, Patrick Löffler et al.: Informatik 5 – Lehrwerk für Gymnasien.
Ernst Klett Verlag, Stuttgart, Leipzig, 2010.

entnommen.

Berechenbarkeit 1

- Eine Funktion $f : A \to Y$ heißt berechenbar, wenn es einen Algorithmus A_f gibt, der diese Funktion „realisiert“.

- Die Quadratfunktion $f : \mathbb{N} \to \mathbb{N}$ ist berechenbar, denn es gibt einen Algorithmus, der für jede gegebene natürliche Zahl n das Quadrat n^2 berechnet.

- Es gibt überabzählbar viele Funktionen $f : \mathbb{N} \to \mathbb{N}$, aber nur abzählbar viele Algorithmen. Das heißt, fast keine Funktion ist berechenbar.

Berechenbarkeit 2

- Mithilfe des Berechenbarkeitsbegriffs lässt sich die Entscheidbarkeit formal definieren: Eine Menge $M \subset X$ heißt entscheidbar relativ zu X, wenn die charakteristische Funktion

$$\chi_M(x) = \begin{cases} 1, & x \in M, \\ 0, & x \in X \setminus M, \end{cases}$$

  berechenbar ist.

- Formulieren Sie das Halteproblem als charakteristische Funktion einer geeigneten Menge.

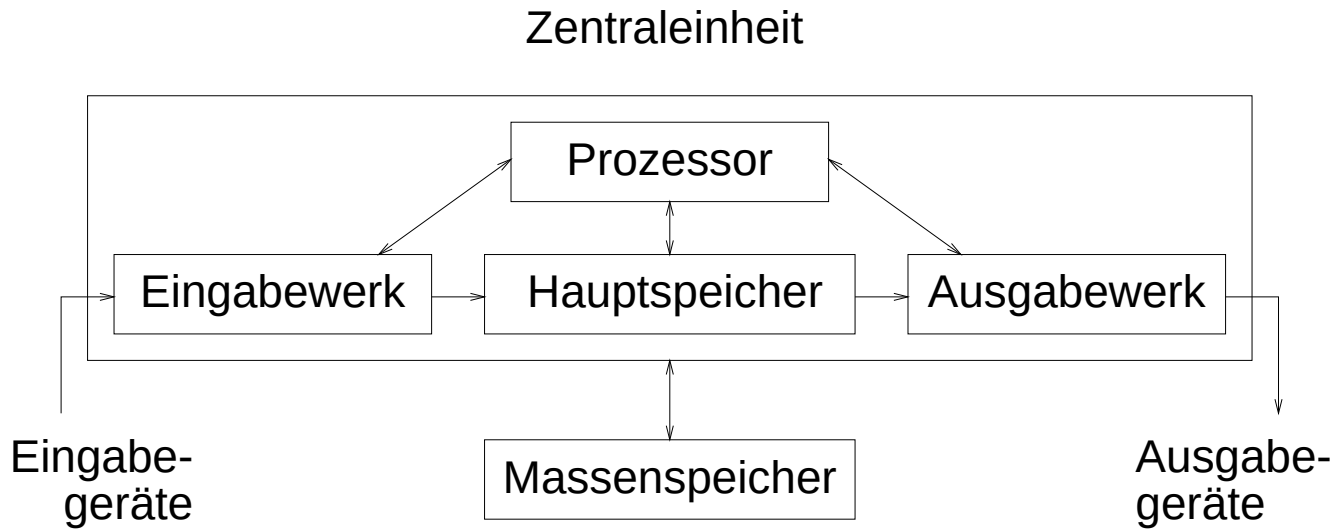
Fazit: Es gibt unentscheidbare Probleme und nicht berechenbare Funktionen.
Mehr zu diesem Thema lernen Sie in den Modulen „Theoretische Informatik“.

Einführung:
Hard- und Software-Grundlagen

1.1 Vom Algorithmus zum Programm
1.2 Programmiersprachen
1.3 Korrektheit, Komplexität und Entscheidbarkeit
1.4 Hard- und Software-Grundlagen

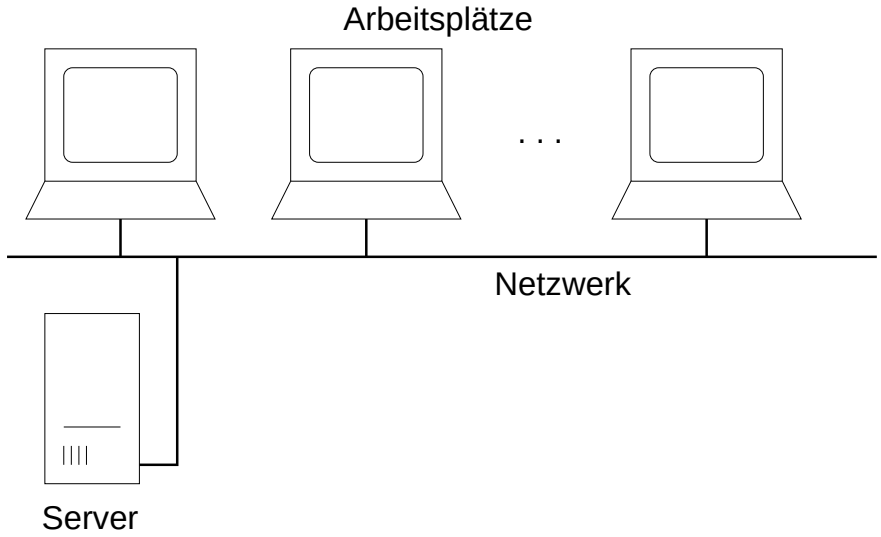
Hardware

Für unsere Zwecke reicht das folgende einfache Modell vom Aufbau eines Rechners. Details lernen Sie in den Modulen „Technische Informatik“, „Rechnernetze“, ... kennen:



Hardware

Eine mögliche Arbeitsumgebung:



Software

- Zur Systemsoftware zählen alle Programme, die für den korrekten Ablauf von Rechnern oder Rechnernetzen erforderlich sind.
- Die Anwendungssoftware wird zur Lösung von Problemen, die nicht ursächlich mit Rechnern zu tun haben, eingesetzt.
- Softwarewerkzeuge unterstützen die Erstellung von System- und Anwendungsprogrammen.

Systemsoftware

Zur Systemsoftware zählen alle Programme, die für den korrekten Ablauf von Rechnern oder Rechnernetzen erforderlich sind:

- Betriebssysteme und ihre Komponenten
- Compiler, Interpreter
- Binder, Lader bzw. Bindelader
- Programme zur Verwaltung von Geräten
- Netzsoftware

Anwendungssoftware

Die Anwendungssoftware wird zur Lösung von Problemen, die nicht ursächlich mit Rechnern zu tun haben, eingesetzt:

- Datenbankprogramme
- Conputeralgebrasysteme
- Office-Software: Textverarbeitung, Tabellenkalkulation, Präsentation, ...
- Internetsoftware: Browser, ...
- Mediensoftware: Grafik-, Photo-, Audio-, Videoprogramme
- ...

Softwarewerkzeuge

Softwarewerkzeuge unterstützen die Erstellung von System- und Anwendungsprogrammen:

- Modellbildung
- Programmierwerkzeuge
- Versionskontrolle
- Integrierte Entwicklungsumgebungen
- ...

Betriebssysteme

- Der Begriff Betriebssystem ist eine zusammenfassende Bezeichnung für alle Programme, die die Ausführung der Benutzerprogramme, die Verteilung der Betriebsmittel auf die einzelnen Benutzerprogramme und die Aufrechterhaltung der Betriebsart (z. B. Stapelbetrieb, Dialogbetrieb) steuern und überwachen.

- Das Betriebssystem bietet seine Dienste dem Benutzer in einer textuellen oder grafischen Oberfläche an.

Betriebssysteme

- Das Betriebssystem kann als eine Erweiterung der Maschine gesehen werden. Der durchschnittliche Programmierer möchte in der Regel beispielsweise nicht die Verwaltung einer Floppy-Disk programmieren, sondern deren Funktionalität als Abstraktion auf hohem Niveau nutzen. In diesem Zusammenhang spricht man auch von einer virtuellen Maschine.

- Das Betriebssystem arbeitet auch als Ressourcenmanager. Moderne Rechensysteme bestehen aus Prozessoren, Speichern, Uhren, Platten, Terminals, Druckern, Netzwerkschnittstellen und vielen weiteren Komponenten. Das Betriebssystem teilt diese Ressourcen untern den verschiedenen Prozessen auf. Dieser Vorgang kann als „Multiplexen in Zeit und Raum“ beschrieben werden.

Wichtige Betriebssysteme

- UNIX-Derivate
  - BSD-Unix (Berkeley Software Distribution)
  - AT&T, System V
  - Linux (Linus Torvalds)
    Distributionen für Linux: RedHat, Suse, Debian, Ubuntu, Knoppix, ...
- Betriebssysteme der Fa. Microsoft
  - MS-DOS
  - Windows 3.x/95/98/Me
  - Windows NT, Windows 2000, Windows XP
  - Windows Vista, Windows 7, Windows 8

Oberflächen von Betriebssystemen

- Ein heutiges Betriebssystem stellt dem Benutzer die Fähigkeiten des Rechners über eine textuelle Oberfläche (Shell) und/oder über eine grafische Oberfläche (GUI, Graphical User Interface) zur Verfügung.

- Beispielsweise gibt es für Unix üblicherweise die Shells sh, bash, csh, tcsh, ksh und einige weitere. Für Linux wurden die grafischen Oberflächen KDE und Gnome entwickelt. Die Wahl der jeweiligen Oberfläche bleibt dem Benutzer überlassen.

Dateiverwaltung

Dateiverwaltungssystem

Komponente eines Betriebssystems, die den gesamten Platz auf externen Speichern verwaltet. Zu den Aufgaben gehören die Lokalisierung von Dateien, die Zuweisung von Speicherplatz und die Buchführung über die Verwendung des Speichers.

Editor

Komponente eines Dateiverwaltungssystems zum Bearbeiten von Texten oder Daten. Verbreitete Editoren unter Unix sind vi, emacs, nedit und gedit. Notepad und Wordpad sind solche für Windows.

Programmierwerkzeuge

- Änderungsverwaltung: diff, patch
- Versionsverwaltungsprogramme: rcs, cvs, svn
- Eingabeanalyse: lex, yacc
- Eingabeverarbeitung: awk
- Programmgenerierung: make

Programmierumgebungen

Programmierumgebungen sind Software-Systeme zur Unterstützung der Programmentwicklung. Typische Bestandteile einer Programmierumgebung sind

- ein sprachspezifischer Editor (Texteditor),
- Compiler und/oder Interpreter,
- Binder, Lader bzw. Bindelader,
- Test- und Debughilfen,
- Quelltextformatierungstools,
- Archivierungswerkzeuge sowie
- Dokumentationsgeneratoren.

Integrierte Entwicklungsumgebungen

Eine Programmierumgebung wird auch integrierte Entwicklungsumgebung (IDE) (integrated development environment) genannt. Integrierte Entwicklungsumgebungen für Java sind beispielsweise

- NetBeans,
- Eclipse,
- IntelliJ IDEA,
- Borland JBuilder und
- Oracle JDeveloper.

Integrierte Entwicklungsumgebungen können ggf. auch für die Arbeit mit mehreren Programmiersprachen geeignet sein.