Praktische Informatik für Wirtschaftsmathematiker, Ingenieure und Naturwissenschaftler I (PIWIN I, 3 V + 1 Ü) WS 2002/03

16. Vorlesungswoche

Berechenbarkeit: Wo liegen Grenzen der Informatik ?

Was lässt sich berechnen, was nicht ?

Zusammenfassende Übersicht der Vorlesunginhalte

Unterlagen:

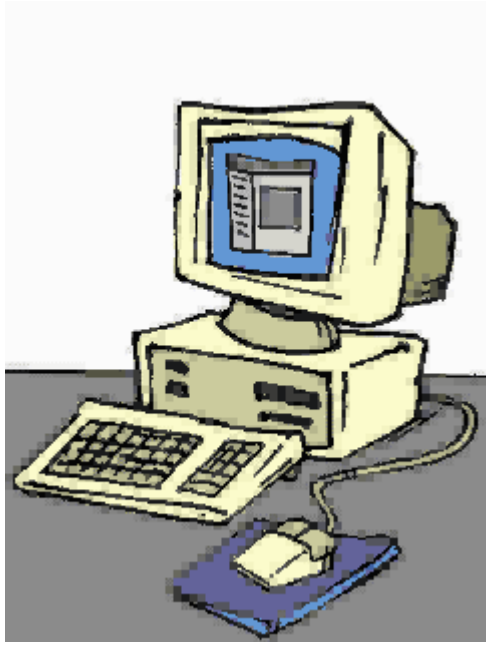
Skript zur Vorlesung PIWIN I von Prof. Moraga WS 00/01,

Kapitel 4, Berechenbarkeit und Entscheidbarkeit

(http://ls1-www.cs.uni-dortmund.de/Lehre/vorlesung_history.html)

Berechenbarkeit und Entscheidbarkeit

- Algorithmusbegriff
- Turingmaschine
- Entscheidbarkeit
- partielle Entscheidbarkeit
- Church´sche These
- Halteproblem
  - Unentscheidbarkeit des Halteproblems



Berechenbarkeit

Berechenbarkeit = algorithmische Lösbarkeit von Aufgaben

Reihe von formalen Ansätzen für die Beschreibung von Algorithmen und Berechenbarkeit

- Turingmaschine (Turing)
- Rekursive Funktionen (Gödel)
- λ-Kalkül (Church)
- Markov-Algorithmen (Markov)

Diese Ansätze haben sich als gleich mächtig erwiesen!

Ein Dämpfer für ungebremste Euphorie:

Gödel (1931): Unentscheidbarkeit der Arithmetik

Es existiert kein Algorithmus, der für jede beliebige Aussage über natürliche Zahlen feststellt, ob sie gilt oder nicht.

Turingmaschine

- einfaches, abstraktes Modell eines Rechensystems, jedoch hinreichend mächtig um prinzipiell Berechnungen vornehmen zu können
- Vorstellung:
  - endlicher Automat mit unendlichem Speicherband,
  - Speicherband hat initial die Eingabe, am Schluss die Ausgabe, dient zwischendurch als Zwischenspeicher
  - Automat kann auf dem Band nach links/rechts navigieren und Einträge lesen und schreiben
- Definition: Turingmaschine (A,S,d,s_0)
  - A, Bandalphabet. A enthalte insb. Leerzeichen b (blank)
  - S, endliche Zustandsmenge mit Anfangszustand s_0
  - d: $S \times A \longrightarrow S \times A \times \{L, R, annehmen, ablehnen\}$
- Berechnungsschritt einer Turingmaschine

  Sei q aktueller Zustand, a das Zeichen im Feld unter dem LS-Kopf,

  d(q,a)=(q´,a´,m), dann besteht nach dem Übergang Zustand q´, auf dem Band steht a´ und die Maschine bewegt sich ggfs gemäß m nach links/rechts oder stoppt mit annehmen/ablehnen.

Entscheidbarkeit

Definition: $L \subseteq A^*$ heißt entscheidbar, falls es eine Turingmaschine TM gibt, die für jedes $x \in A^*$ erfüllt:

x ist Element von L gdw TM akzeptiert x

x ist nicht Element von L gdw TM verwirft x

Definition: $L \subseteq A^*$ heißt partiell entscheidbar, falls es eine Turingmaschine TM gibt, die für jedes $x \in A^*$ erfüllt:

x ist Element von L gdw TM akzeptiert x

Bei partieller Entscheidbarkeit kann die TM auch nicht terminieren, falls x nicht Element von L ist.

A^* beschreibt die Menge aller Zeichenketten über einem Alphabet A

Church´sche These

„intuitiv berechenbar = mit Turingmaschinen berechenbar“

Turingmaschinen (und äquivalente Ansätze) sind eine adäquate Formalisierung des intuitiven Algorithmusbegriffes.

Es gibt keinen Algorithmus der nicht als Turingmaschine beschrieben werden kann.

Anmerkung: bei Turingmaschinen handelt es sich um ein theoretisches Konzept, das natürlich nicht für reale Berechnungen tatsächlich zum Einsatz kommt. Nichtsdestotrotz lassen sich (wenn auch mühsam) auch komplizierte Operationen bewerkstelligen, z.B.

Polynomauswertung, Multiplikation von Matrizen etc.

Unentscheidbarkeit von Problemen

- meint: Aufgabenstellung ist von der Art, dass ein Algorithmus nicht existiert (also auch nicht gefunden werden kann).

- Halteproblem
  - seien Programm P (z.B. als Turingmaschine) und Eingabe D gegeben
  - Frage: Hält P auf D ?

- vorab: Halteproblem ist unentscheidbar

- betrachte Szenario
  - partielle Funktionen $f : A^* \rightarrow A^*$ werden berechnet, A ist fest
  - kodiere Algorithmen als Texte über A
- ermöglicht Trick:
  - Funktionen erhalten Algorithmen als Eingabe
  - füttere Funktion mit eigenem Algorithmus

Unentscheidbarkeit des Halteproblems

- Definition

  Sei $x \in A^*$. Dann bezeichnet φ_x die vom Algorithmus mit Text x berechenbare Funktion. Ist x kein sinnvoller Algorithmentext, sei die Funktion nicht definiert.

- Definition

  Sei $f : A^* \to A^*$ eine partielle Funktion. Der Urbildbereich von f ist

  $$dom\, f = \{ x \in A^* \mid f(x) \text{ ist definiert} \}$$

Sei a aus A.

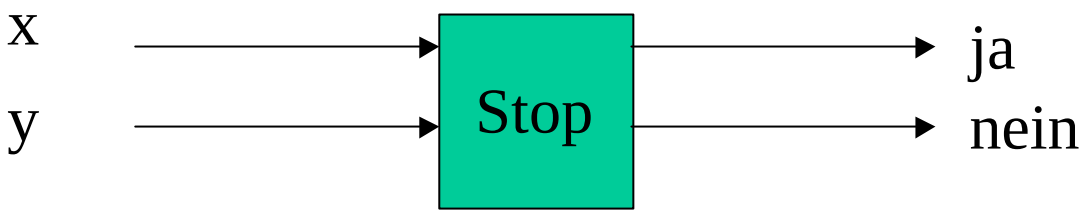
Satz: Die (totale) Funktion $h : A^* \to A^*$ mit

$h(x) = \varepsilon$ falls $x \in dom\, \varphi_x$ und $h(x) = a$ sonst

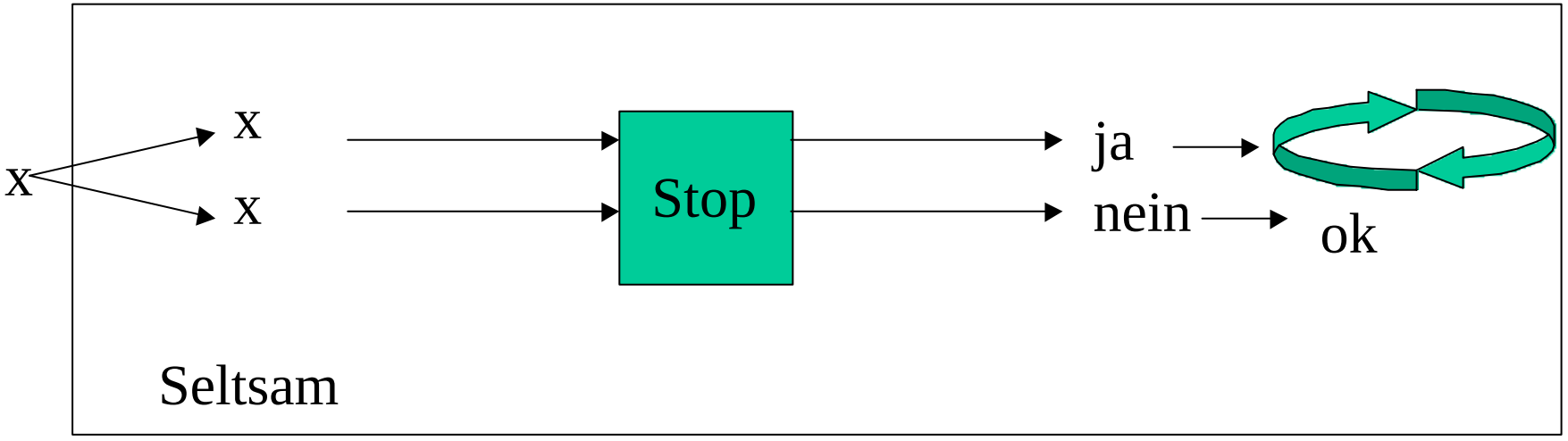
ist nicht berechenbar.

Anschaulicher Beweis

- Maschine STOP liefert für Eingabe (Algorithmus x, Eingabe y) Ausgabe ja falls x(y) hält und nein falls x(y) nicht hält.



- definiere neue Maschine SELTSAM



- Widerspruch: Hält SELTSAM bei Eingabe SELTSAM ?

Mathematischer Beweis

- Annahme: sei h berechenbar, definiere gemäß Churchscher These, eine dann ebenfalls berechenbare Funktion g:

$$g : A^* \to A^*, \text{mit } g(x) = \begin{cases} j_x(x)a & \text{falls } h(x) = \mathring{a} \\ e & \text{sonst} \end{cases}$$

$g(x) \neq j_x(x)$ für alle x aus A*

Da g berechenbar, existiert Algorithmus y aus A*, der g berechnet.

Daher $g = j_y$ daraus folgt aber $j_y(y) = g(y) \neq j_y(y)$

Offensichtlicher Widerspruch, der sich nur durch Aufgabe der Annahme: h ist berechenbar lösen lässt.

Berechenbarkeit hat seine Grenzen

- Es gibt keinen Algorithmus,
  - der für Programme feststellen kann, ob sie auf beliebige Eingaben terminieren.
  - feststellen kann, ob eine berechnete Funktion total ist, injektiv, surjektiv, bijektiv ist, ...
  - feststellen kann, ob zwei gegebene Algorithmen dieselbe Funktion berechnen
  - feststellen kann, ob ein Algorithmus korrekt ist (also eine gegebene Funktion berechnet)
  - ...

- Achtung: Dies bedeutet nicht, dass man nicht in Einzelfällen zu Resultaten kommen kann !!!

V1: Zusammenfassung, Abschlussübersicht

- Informatik
  - was ist Informatik
  - Teilgebiete der Informatik
- Rechensystem
- endlicher Automat
- Spezifikation - Algorithmus - Programm - Prozeß
- Darstellung von Daten im Rechner
  - Text, Zeichenketten
  - Zahlen
    - Mathematik: natürliche / ganze / rationale / reelle Zahlen
    - Informatik: Zahlendarstellung auf endlichem Speicherplatz
      integer Zahlen, Zahlendarstellung
      Fließkomma Zahlen, Zahlendarstellung

V2: Zusammenfassung, Abschlussübersicht

- Spezifikation,
  - Anforderungen: vollständig, detailliert, unzweideutig, widerspruchsfrei
- Algorithmus
  - Anforderungen:
    - Relation über das Kreuzprodukt einer Eingabe- und einer Ausgabemenge. Dadurch werden für jede Eingabe die zulässigen Ausgaben festgelegt.
    - besteht aus wohldefinierten Elementaroperationen, auf einer geeigneten Maschine ausführbar
    - legt die Abfolge der Schritte fest, wobei jeder Schritt genau eine Elementaroperation umfasst.
    - Beschreibung endlicher Länge.
    - benutzt nur endlich viele Speicherplätze zur Ablage von Zwischenergebnissen.
    - Terminierung, begrenzte Schrittanzahl (Anf. gelegentlich eingeschränkt)
    - Determiniertheit: Die Eingabe-Ausgabe-Relation ist rechtseindeutig.
    - Determinismus: In jedem Zustand, der bei Ausführung des Algorithmus erreicht wird, ist jeweils nur ein einziger Folgeschritt als nächster ausführbar.
- Programm
- formale Sprache, Grammatik, Syntax und Semantik

V3: Zusammenfassung, Abschlussübersicht

- Basiskonstrukte imperativer Sprachen
  - Variable und Zuweisung,
    - Unterschied zum Begriff der Variable in der Mathematik
    - Datentyp
  - Konstrollstrukturen:
    - Sequenz
    - Alternative/Fallunterscheidung
    - Wiederholung/Iteration
  - Blockstrukturierung, Funktion/Prozedur,
  - Rekursion

V4: Zusammenfassung, Abschlussübersicht

- Funktion, Prozedur, Methode
  - Parameterübergabe
  - Rückgabewert
  - Abbildung im Speicher: Aufrufstack
  - Existenz von Variablen in Abhängigkeit von Funktionsaufrufen
- Rekursion,
  - unterschiedliche Arten,
  - Umwandlung in iterative Funktionen
  - akkumulierende Parameter
  - Beispiele
    - Türme von Hanoi,
    - Fakultätsfunktion,
    - Fibonacci-Zahlen

V5: Zusammenfassung, Abschlussübersicht

- Datenstrukturen: Arrays
- Algorithmen: Sortieren
  - naives Verfahren, SelectionSort
    - bestimme und entferne Minimum aus einer Menge von Werten
  - Heapsort
    - gleiche Idee wie SelectionSort nur Ausführung basiert auf binärem Baum, der auf der Adressierung von Arrayeinträgen 1,2,...,n aufbaut
    - binärer Baum, Tiefe logarithmisch in der Anzahl Knoten
  - Quicksort
    - divide&conquer Ansatz
    - zerlege Menge anhand eines mittleren Wertes in 2 Teilmengen
    - best case: Kardinalität der Teilmengen gleich, Menge wird halbiert
    - worst case: 1 Teilmenge hat Kardinalität 1

V6: Zusammenfassung, Abschlussübersicht

- Bewertung von Algorithmen
  - Speicherplatz und Laufzeit
  - Testen, Messen
  - Mathematisches Modell: O-Notation
  - Betrachtungsstandpunkt:

    Größenordnung des Aufwands in Abhängigkeit von der Problem/Eingabegröße

    konstante Faktoren (und Summanden) werden nicht berücksichtigt
- Beispiele
  - SelectionSort: Best Case, Worst Case, Average Case: $O(n^2)$
  - Heapsort: Worst Case $O(n \log n)$ $(2\ n \log n)$
  - Quicksort:
    - Worst Case $O(n^2)$
    - Average Case $O(n \log n)$ $(1{,}386\ n \log n)$

V7, V8: Zusammenfassung, Abschlussübersicht

- Objektoriertierte Programmierung
  - Objekt
    - vereinigt Attribute und Methoden
  - Klasse
    - abstrakte Klasse
  - Attribut
  - Methode,
    - Überschreiben von Methoden, (gleiche Signatur)
    - Überladen von Methoden, (unterschiedliche Signatur)
  - Vererbung
  - Polymorphie
  - spätes Binden

V9: Zusammenfassung, Abschlussübersicht

- Dynamische Datenstrukturen
  - Grundidee: Verkettung mittels Referenzen, die als Attribute definiert werden
  - Listen
    - einfach verkettete, lineare Listen
      - mit Fuß/Verweis auf das Listenende
    - doppelt verkettete Listen
  - Bäume
    - binäre Bäume, Suchbäume
  - Typische Operationen
    - Einfügen
    - Suchen/Ändern
    - Löschen
    - Traversieren: Breiten/Tiefendurchlauf
  - Rekursive Methoden für rekursive Datenstrukturen

V10: Zusammenfassung, Abschlussübersicht

- Dynamische Datenstrukturen
  - Graphen:
    - Adjazenzmatrix, -listen
    - Algorithmen für Graphen
      - Traversierung: Tiefensuche, Breitensuche
      - kürzeste Wege
      - starke Zusammenhangskomponenten
      - Traveling Salesman
    - Datenstrukturen für Graphen in Java
  - Stacks & Queues
- weitere Datenstrukturen für Mengen
    - charakteristische Vektoren
    - Hashing

V11: Zusammenfassung, Abschlussübersicht

- Abschluss zur Objektorientierten Programmierung
  - Interfaces
    - Mehrfachvererbung
    - Unterschied zu (abstrakten) Klassen
  - Exceptions
    - Kontrollfluss bei der Behandlung von Ausnahmefällen
    - try-throw-catch Mechanismus
  - Packages
    - Pakete zur Strukturierung von grossen Softwaresystemen
    - Bibliotheken

V12: Zusammenfassung, Abschlussübersicht

- Begriffe: Softwaretechnik
- Vorgehensmodelle

  Wasserfallmodell, modifizierte Phasenmodelle, V-Modell, W-Modell, Spiralmodell, Prototyping
- Versionsmanagement, Konfigurationsmanagement

Phasen im SW-Entwurf

- Planungsphase: legt grob fest, was mit wieviel Aufwand erstellt werden soll
  - Lastenheft, Projektplan und Projektkalkulation
- Definitionsphase: legt genau fest, was erstellt werden soll
  - Produktdefinition: Pflichtenheft, Produktmodell, Benutzungsoberfläche und Benutzungshandbuch
- Entwurfsphase: legt fest, wie SW-Architektur gestaltet wird
  - Software-Architektur, Spezifikation von Systemkomponenten
- Implementierungsphase: erzeugt Software
  - Programmcode
  - füllt, realisiert Systemkomponenten
  - schließt Testphase mit ein
- Installation & Wartungsphase

V13: Zusammenfassung, Abschlussübersicht

- Konzepte der Strukturierten Analyse
  - Funktionalität: Funktionsbaum
  - Wechselwirkung Funktionen&Daten: Datenflußdiagramm
  - Daten: Data Dictionary
  - Dynamik: Kontrollstrukturen, Entscheidungstabellen
- Entity-Relationship Modelle
  - Grundbegriffe:
    - Entität, Entitätsmenge, Attribut, Schlüsselattribut
    - Assoziation, Kardinalitäten, rekursive Assoziation, Rolle
    - graphische Darstellung
  - Semantische Datenmodellierung
    - Aggregation (ist-teil-von)
    - Generalisierungshierarchie
  - Tabellendarstellung, Bezug zu Datenbanken

V 14: Zusammenfassung, Abschlussübersicht

- Notationen in der objektorientierten Analyse

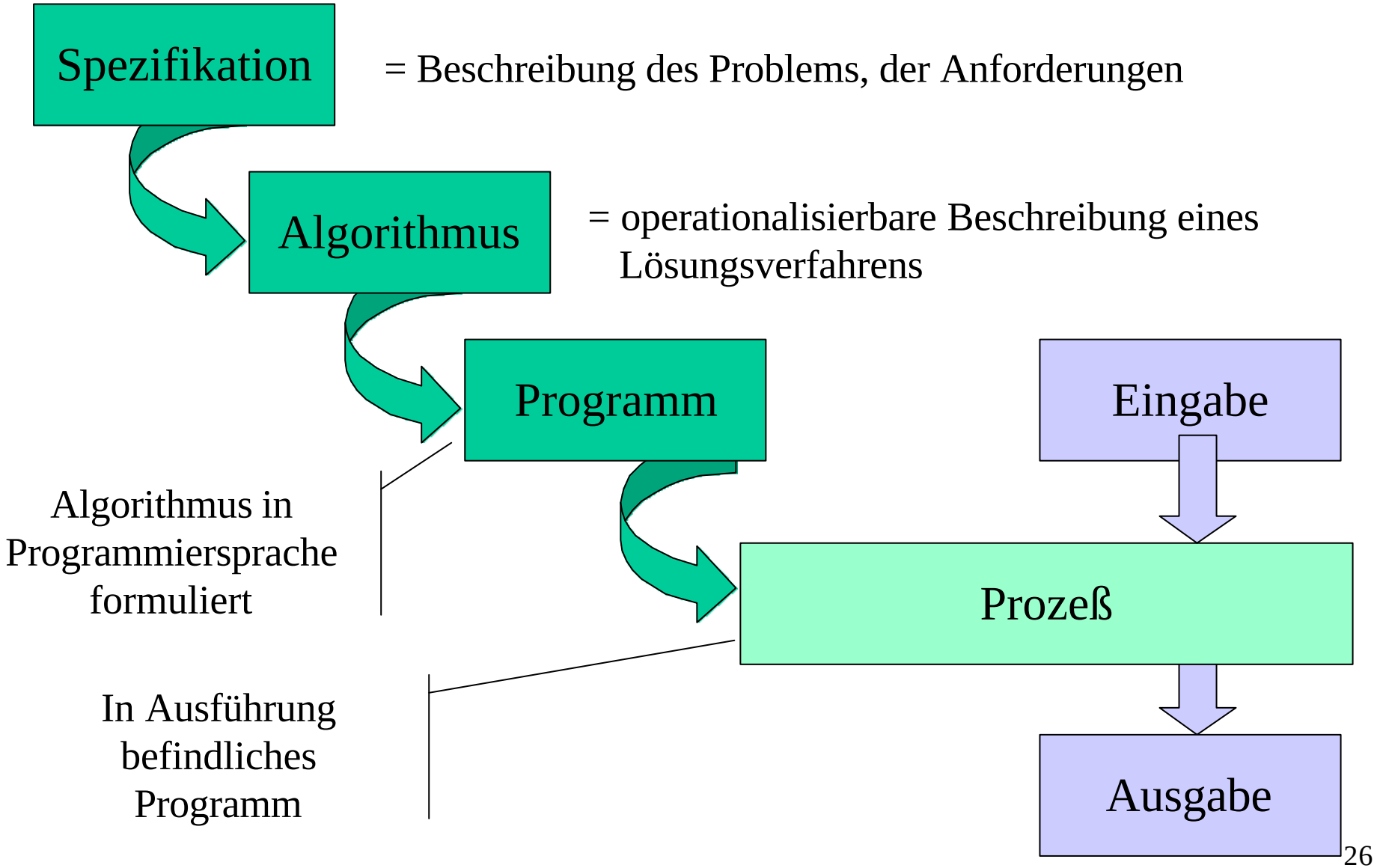
  Unified Modeling Language (UML)

  – Anwendungsfalldiagramme
  – Klassendiagramme
    - Generalisierung, Spezialisierung
    - Assoziationen, Komposition, Aggregation
  – Verhalten zwischen Objekten: Interaktionsdiagramme
    - Sequenzdiagramme
    - Kollaborationsdiagramme
  – internes Verhalten eines Objektes::
    - Zustandsdiagramme
  – Abläufe:
    - Aktivitätsdiagramme
  – Paketdiagramme
  – Verteilungsdiagramme
- Entwurfsmuster / Design Patterns

V 15: Zusammenfassung, Abschlussübersicht

- Vorgehensweise bei der objektorientierten Analyse
  - Grobanalyse
    - Abgrenzung der Problemstellung,
    - Zerlegung in Teilprobleme, Festlegung von Schnittstellen
    - Anwendungsfälle
  - statisches Modell
    - Klassendiagramme, Klassenstruktur,
    - Klassen, Methoden, Attribute
    - Vererbungsstrukturen, Generalisierung, Spezialisierung
    - Assoziationen zwischen Objekten, Kardinalitäten
  - dynamisches Modell
    - Interaktionen zwischen Objekten
    - Verhalten von Objekten, Lebenszyklen, Zustände

Stationen im Entwurf von Algorithmen und Programmen



Allerletzte Übersicht

- Begriffe
  - Spezifikationen, Algorithmen, formale Sprachen, Grammatik
- Programmiersprachenkonzepte
- Grundlagen der Programmierung
  - imperative Programmierung
  - objektorientierte Programmierung
- Algorithmen und Datenstrukturen
- Effizienz und Komplexität von Algorithmen
- Programmentwurf, Softwareentwurf
- Berechenbarkeit und Entscheidbarkeit von Problemen

- Viel Spaß bei PIWIN II im Sommersemester