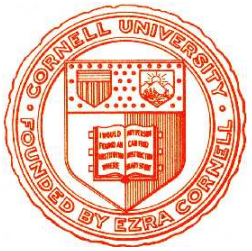
Theoretische Informatik I
Modellierungskonzepte der Informatik

Wintersemester 2014/15



Christoph Kreitz / Sebastian Böhne

Theoretische Informatik

{kreitz,boehne}@uni-potsdam.de

http://cs.uni-potsdam.de/ti1-ws1415

1. Lehrziele und Lernformen
2. Organisatorisches
3. Lernerfolg und Arbeitsethik

Theoretische Informatik: Worum geht es?

- **Theorie befaßt sich mit grundsätzlichen Fragestellungen**
  - Was ist machbar? Was nicht? Warum ist das so?
  - Gibt es allgemeine Methoden zur Lösung eines Problems?
  - Welche Erkenntnisse kann man als gesichert betrachten?
- **Was bedeutet das für die Informatik?**

  Wir müssen uns mit einer Reihe von Fragen auseinandersetzen
  - Wie kann man Softwaresysteme möglichst einfach modellieren?

    ... ohne dabei von einer spezifischen Programmiersprache abhängig zu werden
  - Auf welche Arten kann man Hard-/Softwaremodule kombinieren?
  - Wie hängen verschiedenartige Computerarchitekturen zusammen?
  - Wie kann man sicherstellen, daß Software korrekt ist?
  - Wie flexibel kann man Programmiersprachen gestalten?
  - Was kann man prinzipiell mit Computern lösen, was nicht?
  - Welche Probleme sind effizient lösbar – welche nicht?

  Wir müssen mit mathematischen Methoden vorgehen
  - Abstraktion von irrelevanten Details (Hardware/Programmiersprache)
  - Fokus auf beweisbare Erkenntnisse mit weitreichender Gültigkeit

... Wozu soll das gut sein?

- **Sollte man sich nicht eher um praktische Fragen kümmern?**
  - Konkrete Programme, Apps und Softwaresysteme schreiben?
  - Lernen wie man konkrete Hardware besser ausnutzt?
  - Neue Programmiersprachen und Programmiertricks lernen?
- **Das ist wichtig, aber ... keine Informatik auf Profiniveau**
  - Wie wollen Sie sicher sein, daß Ihre Programme auch funktionieren?
    selbst wenn sich die Hardware ändert
    und die Programmiersprache Ihrer Wahl nicht genutzt werden kann
  - Details ändern sich schnell, allgemeine Modelle und Methoden nicht
- **Theoretisches Wissen bewahrt Sie vor groben Fehlern**
  - Viele Programmierer haben sich schon in Probleme verbissen
    weil sie nicht wußten oder glaubten, daß sie unlösbar sind
  - z.B. Optimale Navigation ist nicht effizient möglich
    Flexible Programmiersprachen sind nicht (effizient) compilierbar
    Korrektheit von Software kann nicht getestet werden

Korrektheit ist das grösste Problem

- **Wieso? Wir haben doch Tests?**
  - Das reicht in anderen Fachgebieten wie z.B. Autobau doch auch?
  - Und wenn wir später Fehler finden, können wir sie doch korrigieren und ein update schicken. Funktioniert das nicht schon seit Jahren?
- **Ein alter Trugschluß und viele glauben immer noch daran**
  - Was für analoge Systeme gilt, gilt nicht immer in der digitalen Welt
  - Korrektheit von Software ist durch Tests nicht beweisbar

    (diese Erkenntnis kann man beweisen!)
  - Nur die allergröbsten Fehler werden durch Testen entdeckt
  - Auch sorgfältig getestete Programme enthalten größere Fehler

    "Hochqualitätssoftware" enthält ca. 2 Fehler pro 100 Zeilen Code
- **Können wir uns Softwarefehler erlauben?**
  - Würden Sie ein Auto fahren, dessen Lenkung manchmal falsch arbeitet?
  - ... ein Flugzeug besteigen, dessen Triebwerk gelegentlich ausfällt?
  - ... Internetbanking verwenden, wenn jeder mitlesen kann, was Sie tun?

Die Fehler in Software sind oft einfach, die Folgen nicht

#1 Habsheim 26.6.1988: Air France Airbus A 320

- **Computergesteuerter Tiefflug auf Flugshow**
  - Vorgabe: Langsamflug 30 Meter über Landebahn
  - Tatsächliche Flughöhe 30 Fuß (9 Meter)
  - Zu langsamer Anstieg am Ende der Landebahn

    Computer verweigert Not-Kommandos des Piloten
  - Höhe der Bäume ca 15 Meter
  - **Flugzeug streift Bäume und stürzt ab – 3 Tote**

#2 Warschau 14.9.1993: Lufthansa Airbus A 320

- **Vorgeschichte: Bangkok 26.5.1991**
  - Lauda Air Boeing 767-300
  - Linkes Triebwerk schaltet beim Steigflug auf Umkehrschub
  - Flugzeug stürzt ab, alle 223 Insassen sterben.
- **1993: Landung mit computerkontrolliertem Airbus**
  - Landung auf regennasser Fahrbahn (Aquaplaning)
  - Räder drehen langsamer als 130km/h

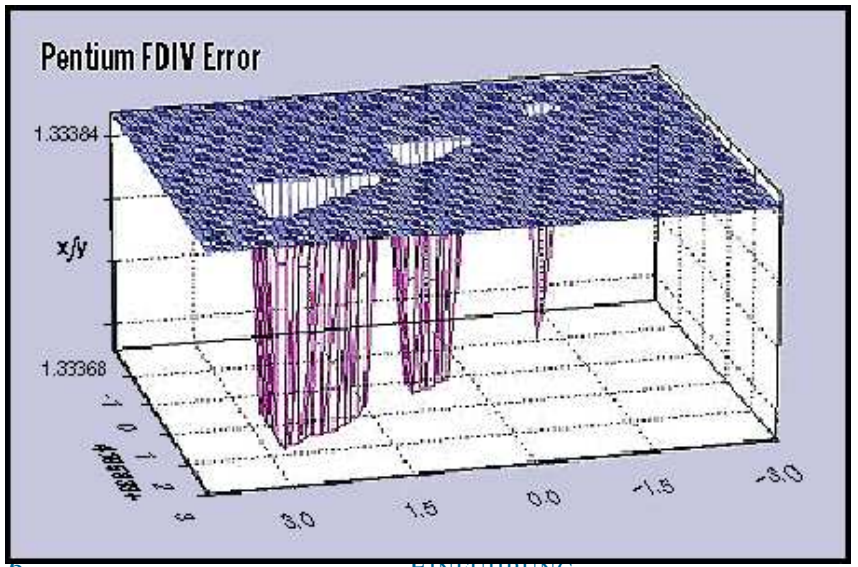
    Computer verweigert Umkehrschub, da "nicht gelandet"
  - Zu schwache Bremsleistung,
  - Flugzeug prallt auf Erdwall am Ende der Landebahn, fängt Feuer
  - **Kopilot stirbt an Folgen des Aufpralls**
  - **Ein Passagier stirbt an Rauchvergiftung**

#3 Oktober 1994: Pentium I Prozessor

- **Damals neuester Prozessor für PC Architekturen**
  - Chip verwendet schnelleren SRT Divisionsalgorithms
  - **Falsche Resultate bei Programmen, die Division verwenden**

    Für alle x, y ist $x - (x/y) * y = 0$

    aber für $x = 4195835$, $y = 3145727$ lieferte der Pentium die Antwort **256**
  - Ursache: Fehlerhafte Umsetzung des Algorithmus in Prozessortabellen
  - 2 Millionen Prozessoren ausgeliefert
  - **450 Millionen Dollar Schaden**



#4 Korou, 4.6.1996: ESA Ariane 501

- **Jungfernflug der neuen ESA Rakete**
  - Nach 37 Sekunden erreicht Steiggeschwindigkeit 32768 interne Einheiten
  - **Überlauf der 16bit Arithmetik**
  - Haupt- und Ersatzrechner schalten ab und senden unsinnige Steuerbefehle
  - Rakete wird instabil und sprengt sich selbst
  - **Kosten des verfehlten Starts: 250 Millionen DM**
  - **Kosten der zerstörten Satelliten: 850 Millionen DM**
  - Entwicklungskosten 11.8 Milliarden DM ... aber **Startsoftware unverändert von Ariane 4 übernommen** obwohl Schubkraft erheblich höher als bei Ariane 4

#5a 23.9.1999: Mars Climate Orbiter

- **Sonde zur Erforschung der Marsatmosphäre**
  - Geplante Umlaufbahn 160km über Marsoberfläche
  - Tatsächliche erreichte Umlaufbahn unter 60km
  - Steuerungssystem überheizt, Sonde stürzt ab
  - **Kosten: 125 Millionen Dollar**
  - Ursache: **Einheitenfehler**

    Navigationsinstrumente liefern Daten in Fuß, Zoll, Meilen, ...

    Software setzt metrischen Einheiten voraus (kein Check)

#5b 3.12.1999: Mars Polar Lander

- **Sonde zur Erforschung der Marsoberfläche**
  - Endphase der Landung beginnt bei 40 Fuß (12 Meter)
  - Bremsraketen schalten ab, wenn Boden berührt
  - Tatsächliche Landephase beginnt bei 40 Metern
  - Landebeine vibrieren in Atmosphäre
  - **Computer identifiziert Landung, schaltet Bremsraketen ab**
  - Sonde stürzt ungebremst aus 30 Metern Höhe ab
  - **Kosten: 165 Millionen Dollar**
  - Ursache: **Einheitenfehler + zu einfache Landekriterien**

Die Liste der Pannen scheint endlos

- **Mehr Flugzeugunfälle**
  - Straßburg 20.1.1992, Air Inter A320: **Aufprall auf Berge, 87 Tote**
    Einheitenprobleme beim Landeanflug (3300 Fuß/min statt 3.3 Grad)
  - Cali, Dezember 1995, Boeing 757: **Aufprall auf Berge, 159 Tote**
    Fehlerhafte Daten im Navigationssystem $\mapsto$ falsche Flugrichtung
- **Microsoft Windows 95, 98, 2000, XP, Vista, 7, 8, (10), ...**
  - System stürzt ab, friert ein, verliert Daten, ...
  - Große Anfälligkeit gegenüber Viren
  - Wachsende Instabilität im Laufe des Betriebs
- **Heartbleed Fehler in Open-SSL** (April 2014)
  - **Zeile im Code vergessen**
  - Ermöglicht Auslesen des Speichers bei sicheren Internet-Verbindungen
- **Shellshock Fehler in Unix-Derivaten** (September 2014)
  - Angreifer können beliebige Kommandos ausführen
  - Alle großen Server der Welt waren seit 1992(!) verwundbar

⋮

Wir haben ein Problem

Können wir die Kontrolle von Luftfahrt, Telephonnetzen, Banken, Strom, Wasser, Militär,... wirklich an Software übergeben?

- **Softwareproduktion ist immer noch wie in den 70er Jahren**
  - Große, unrealistische Projekte mit unklaren Anforderungen
  - Termindruck führt zu vorschneller Auslieferung
  - Probleme führen zu ad hoc Änderungen statt Revision des Entwurfs
  - Implementierung fokussiert auf Modellierungs-/Programmiersprachen statt auf Analyse des Problembereichs
  - Es gibt tatsächlich noch "Programmierer", die sich direkt an ihr Terminal setzen und Programme eintippen, ohne einen Entwurf zu machen
  - Programmierer geben keine Begründung für Korrektheit ihres Programms
- **Wir brauchen besser qualifizierte Software-Entwickler**
  - Fähigkeit, die Korrektheit von Software zu analysieren und verifizieren
  - Logisch-methodisches Denken, Verständnis von Grundlagen & Grenzen

Ein(e) (Wirtschafts-/Bio-,...)Informatiker(in) muss

- **Nachweisen können, daß Programme ihre vorgesehenen Aufgaben korrekt erfüllen**
  ... damit es später keine Klagen oder Desaster gibt
- **Zeigen können, wie gut Programme skalieren**
  ... damit sie auch mit großen Datensätzen effizient arbeiten
- **Modellierungstechniken für Softwaresysteme und die mögliche Arten ihrer Komposition einsetzen und erklären können**
- **Ausdruckskraft und Effizienz der Compilierung programmiersprachlicher Konstrukte einschätzen können**
- **Stärken und Schwächen verschiedener Maschinenmodelle abschätzen können**
- **Grenzen des effizient Lösbaren abschätzen können**
  ... damit nicht mehr versprochen wird, als man nachher liefern kann
- **Grenzen des überhaupt Machbaren erkennen**
  ... damit man sich nicht in unlösbare Aufgabenstellungen verbeißt

Deswegen müssen wir trainieren, trainieren, ...

- **Anhand vereinfachter abstrakter Modelle**

  ... damit Sie sich auf das Erlernen einer Methode konzentrieren können und nicht von der Vielfalt der Aufgaben erdrückt werden

- **Es geht um Verständnis allgemeiner Zusammenhänge**

  – Details ändern sich schnell, Modelle und Methoden nicht

- **Fokus liegt auf Grundlagen, nicht auf Anwendungen**

  – Ohne solide Grundlagen gibt es keine gute praktische Arbeit

  – Das gilt für jeden Beruf, in dem Qualität erwartet wird

  – In der Informatik sind die Grundlagen immer sehr mathematisch

- **Der praktische Nutzen ist nicht immer sofort erkennbar**

  – Das ist im Studium genauso wie im Profisport

  – Haben Sie einmal Profifußballer beim Training beobachtet?
  Konditionstraining, Balltechnik, Standardsituationen, ... kaum Spielen

Konkrete Themen der Theoretischen Informatik

- **Mathematische Methodik in der Informatik** TI-1
- **Automatentheorie und Formale Sprachen** TI-1
  - Endliche Automaten und Reguläre Sprachen – Lexikalische Analyse
  - Kontextfreie Sprachen und Pushdown Automaten – Syntaxanalyse
  - Turingmaschinen, kontextsensitive und allgemeine formale Sprachen

---

- **Theorie der Berechenbarkeit** TI-2
- **Komplexitätstheorie** TI-2

Der Lehrstoff

- **Reihenfolge und Notation folgt Leittext**
  - J. Hopcroft, R. Motwani, J. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, Pearson 2002* (nicht 2011)
  - Vorlesungsfolien sind im Voraus auf dem Webserver erhältlich
  - Videomitschnitte der Vorlesungen von 2005/2006/2011 online verfügbar
- **Lesenswerte Zusatzliteratur**
  - G. Vossen, K.-U. Witt: *Grundkurs Theoretische Informatik. Vieweg 2011*
  - M. Sipser: *Introduction to the Theory of Computation. PWS 2012*
  - A. Asteroth, C. Baier: *Theoretische Informatik, Pearson 2008*
  - J. Hromkovic: *Sieben Wunder der Informatik, Teubner Verlag 2006*
  - I. Wegener: *Theoretische Informatik, Teubner Verlag 2005*
  - U. Schöning: *Theoretische Informatik - kurzgefaßt, Spektrum-Verlag 2008*
  - K. Erk, L. Priese: *Theoretische Informatik, Springer Verlag 2009*
  - H. Lewis, C. Papadimitriou: *Elements of the Theory of Computation, PH 1998*
  - P. Leypold: *Schneller Studieren. Pearson 2005*

Lehr- und Lernformen

- **Selbststudium ist das wichtigste**
  - Lernen durch Bearbeitung verschiedener Quellen (Literatur, Web,...)
  - Trainieren durch Lösung von leichten und schweren Beispielaufgaben alleine und im Team mit anderen
  - Nachweis von Fähigkeiten in Prüfungen und Projekten
  - Ziel ist Verständnis eines Themengebiets (nicht nur der Vorlesung)
  - Unsere Aufgabe ist, Ihnen dabei zu helfen
- **Vorlesung**

  Was soll ich lernen ?
  - Vorstellung und Illustration zentraler Konzepte und Zusammenhänge
  - Knapp und bewußt "unvollständig" – nur als Heranführung gedacht
  - Die Idee (Verstehen) zählt mehr als das Detail (Aufschreiben)
  - Es hilft, schon etwas über das Thema im Voraus zu lesen
  - Stellen Sie Fragen, wenn Ihnen etwas unklar ist !!
  - Nutzen Sie das Tutorium

    wöchentlich Do 12:15–13:45

Lehr- und Lernformen (II)

- **Übungen** — Vertiefung und Anwendung
  - Kurzquiz als Selbsttest – verstehe ich bisher besprochene Konzepte?
  - Betreutes Üben in Gruppen: Lösung von Problemen unter Anleitung
  - Klärung von Fragen allgemeinen Interesses
  - Eigenständige Einarbeitung in neue Konzepte und Fragestellungen
  - Bearbeitung von aufwändigeren Hausaufgaben: Feedback & Korrektur
    - Ziel ist verständliches Aufschreiben einer vollständigen Lösung
    - Arbeit in Gruppen sehr zu empfehlen
    - Lösungen schwieriger Aufgaben werden im Tutorium besprochen
  - Selbst aktiv werden ist notwendig für erfolgreiches Lernen
  - Kommen Sie vorbereitet – Sie lernen mehr dabei
- **Sprechstunden** — Persönliche Beratung
  - Fachberatung zur Optimierung des individuellen Lernstils
  - Klärung von Schwierigkeiten mit der Thematik
  - Hilfe, wenn der Lernfrust überhand nimmt

Das Team

Christoph Kreitz
Raum 1.18, Telephon 3060
kreitz@cs.uni-potsdam.de

Sebastian Böhne
Raum 1.23, Telephon 3014
boehne@uni-potsdam.de

Tutoren

Maxim Görbing	goerbing@uni-potsdam.de
Thomas Kern	tkern@uni-potsdam.de
Chris Kindler	ckindler@uni-potsdam.de
Alexander Schulze	alexanderschulze@gmail.com
Thomas Verweyen	verweyen@uni-potsdam.de

Organisatorisches

- **Zielgruppe: ab 1. Semester**
  - Mathematische Grundkenntnisse sind wichtig (Brückenkurs!)
- **Vorlesung**
  - Wöchentlich Fr 8:15–9:45
- **Tutorium** (optional, aber dringend zu empfehlen)
  - Besprechung von allgemeinen Fragen und schwierigen Hausaufgaben
  - Wöchentlich Do 12:15–13:45
- **Übungen**
  - 5 Gruppen, wöchentlich (Montags – Mittwochs) je 2 Stunden
- **Sprechstunden**
  - C. Kreitz: Fr 10:30–11:30 . . . und immer wenn die Türe offen ist
  - S. Böhne: Mi 12:00–14:00
  - Tutoren: individuell in Übungsgruppen vereinbaren

Leistungserfassung

- **Eine Klausur entscheidet die Note** (Bonuspunkte zählen 5%)
  - Hauptklausur Do 12. Februar 2015, 15:00–18:00 Uhr (Anmeldung!)
  - Probeklausur Fr 19. Dezember 2014, 8:15–9:45 Uhr
- **Zulassung zur Klausur**
  - 50% der Punkte in den Hausaufgaben
    - Gruppen bis 4 Studenten dürfen gemeinsame Lösungen abgeben
    - Gruppen dürfen sich nur nach Rücksprache ändern
    - Klausurzulassungen aus Vorjahren sind nicht gültig
  - Probeklausur zählt wie ein Hausaufgabenblatt
- **Vorbereitung auf die Klausur**
  - Kurzquiz in jeder Übungsstunde ernsthaft bearbeiten
  - Eigenständige Lösung von Haus- und Übungsaufgaben
  - Feedback durch Korrektur der Hausaufgaben und der Probeklausur
  - Klärung von Fragen in Übung und Sprechstunden

Fangen Sie frühzeitig mit den Vorbereitungen an

Welche Vorkenntnisse sollten Sie mitbringen?

Eine gute Oberstufenmathematik reicht aus

- **Verständnis mathematischer Konzepte und Notationen**
  - Elementare Mengentheorie und die Gesetze von $\{x|P(x)\}$, $\cup$, $\cap$
  - Bezug zwischen Mengen, Relationen und Funktionen
  - Datenstrukturen wie Listen, Wörter, Graphen, Bäume ...
  - Elementare Gesetze der Algebra und Logik
  - Elementare Wahrscheinlichkeitsrechnung
  - Zusammenhang zwischen formaler und informaler Beschreibung

  Nötiges Vokabular wird bei Bedarf kurz vorgestellt/wiederholt/eingeübt

- **Verständnis mathematischer Beweismethoden**
  - Wichtig für Analyse von Befehlssequenzen, Schleifen und Rekursion
  - Essentiell, um zeigen zu können, daß etwas nicht möglich ist

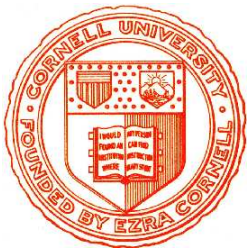
Lernerfolg

- **kommt nicht durch passive Teilnahme**
  - Sammeln von Folien, Übungen, Videos, ...
  - Ansehen von Vorlesungsvideos, Folien und Musterlösungen
  - Rein physische Anwesenheit in Vorlesung und Übung
  - "Bulimielernen" kurz vor der Klausur
- **sondern nur durch eigene Aktivität**
  - Durcharbeiten von Notizen/Folien/Buch/Webseiten vor der Vorlesung
  - Erstellen und Überarbeiten von Notizen zur Vorlesung *(aktives Lesen)*
  - "Weiterdenken": welche Fragen/Ideen folgen aus den Vorlesungsthemen?
  - Fragen stellen während, vor, nach der Vorlesung/Übung/Sprechstunde
    "wer nicht fragt bleibt dumm" – Sie können sich nicht blamieren
  - Erarbeitung von Lösungsideen zu Aufgaben vor der Übungsstunde
- **... und Ehrlichkeit**
  - Sie lernen nichts, wenn Sie Lösungen von anderen übernehmen
  - Erfolgserlebnisse entstehen, wenn Sie Schwierigkeiten selbst überwinden

Wir helfen Ihnen gerne dabei

Theoretische Informatik I

Einheit 1



Mathematische Beweisführung

1. Formale und informale Beweise
2. Verkettung einfacher Argumente
3. Widerlegungsbeweise
4. Induktion und Datenstrukturen
5. Grundsätzliche Methodik

Anhang: Wichtige Grundbegriffe

Beweisführung in der Informatik – warum?

- **Testen von Programmen ist unzureichend**
  - Nur hilfreich zur Entdeckung grober Fehler
  - Viele kleine, aber gravierende Fehler fallen durch das Testraster
    z.B. Pentium Bug (Zeile vergessen), Ariane 5 (Arithmetik-Overflow),
    Mars Polar Lander (Einheitenfehler), Heatbleed (Zeile vergessen), ...
  - Selbst "Hochqualitätssoftware" hat 2-3 Fehler pro 100 Zeilen Code
  - Nur Amateure glauben noch an Fehlerlosigkeit getesteter Programme
- **Programme müssen "bewiesen" werden**
  - Erfolgreicher Beweis zeigt genau, wie das Programm arbeitet
    Erfolgloser Beweisversuch deutet auf mögliche Fehler im Programm
  - In sicherheitskritischen Anwendungen werden seit einigen Jahren
    Programme ohne Korrektheitsbeweis nicht akzeptiert
  - Gute Programme entstehen aus dem Beweis einer Lösungsidee

  Entwickler müssen die Korrektheit von Programmen beweisen können

Aber was ist eigentlich ein Beweis?

Ein Beweis ist ein Argument, das den Leser überzeugt

- **Eine Begründung, warum eine Behauptung gelten soll**
  - Intuitiv einsichtige, logisch schlüssige und lückenlose Erklärung
  - Braucht klare Formulierung der Behauptung und der Annahmen
- *Alle Primzahlen sind ungerade*
  - Unausgesprochene Annahme: es geht um Primzahlen größer als 2
  - Begründung: *jede gerade Zahl ist durch 2 teilbar*
    jede Primzahl ist nur durch 1 und sich selbst teilbar
    jede Primzahl größer als 2 ist nicht durch 2 teilbar
- **Gute Beweise sind genau und übersichtlich**
  - Klar erkennbarer roter Faden in der Argumentation
  - Knapp genug, um verständlich zu sein
  - Genau genug, um fehlende Details rekonstruieren zu können

    Keine Gedankensprünge, außer wenn sie leicht nachvollziehbar sind

Manchmal kann man Verblüffendes beweisen

- *Es gibt genauso viele gerade Zahlen wie natürliche Zahlen*
  - Kann das stimmen? Die geraden Zahlen sind doch nur die Hälfte?
  - Was heißt das überhaupt "gleich viele" bei unendlichen Mengen?
  - Die Anzahl der Elemente ist kein guter Vergleichsmaßsstab
    deswegen verwendet man Bijektionen zwischen Mengen als Kriterium
    A und B sind gleichmächtig, wenn es eine Bijektion $f{:}A{\to}B$ gibt
  - Wie könnte eine Bijektion von $\mathbb{N}$ nach $\mathbb{N}_2$ (gerade Zahlen) aussehen?
    Definiere $f{:}\mathbb{N}{\to}\mathbb{N}_2$ *durch* $f(n) = 2n$, *dann ist* f *injektiv und surjektiv.*
- $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N}{\times}\mathbb{N}$ *und* $\mathbb{Q}$ *sind gleichmächtig*
  - Bijektion von $\mathbb{N}{\to}\mathbb{Z}$ beschreibbar als Aufzählung 0,1,-1,2,-2,3,...
  - $f{:}\mathbb{N}{\times}\mathbb{N}{\to}\mathbb{N}$ zählt Zahlenpaare diagonal durch $f(x,y) = (x{+}y)(x{+}y{+}1){\div}2 + y$
  - Rationale Zahlen sind wie Paare von ganzen und natürlichen Zahlen
- **Wie sieht es aus mit reellen Zahlen?**
  - $\mathbb{R}$ *ist genauso mächtig wie das offene Intervall* $(0,1)$
    Definiere $f{:}(0,1){\to}\mathbb{R}$ *durch* $f(x) = 1/(1{-}x) - 1/x$.
  - Aber $(0,1)$ und $\mathbb{R}$ sind mächtiger als $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{N}{\times}\mathbb{N}$ und $\mathbb{Q}$.

(Folie 16)

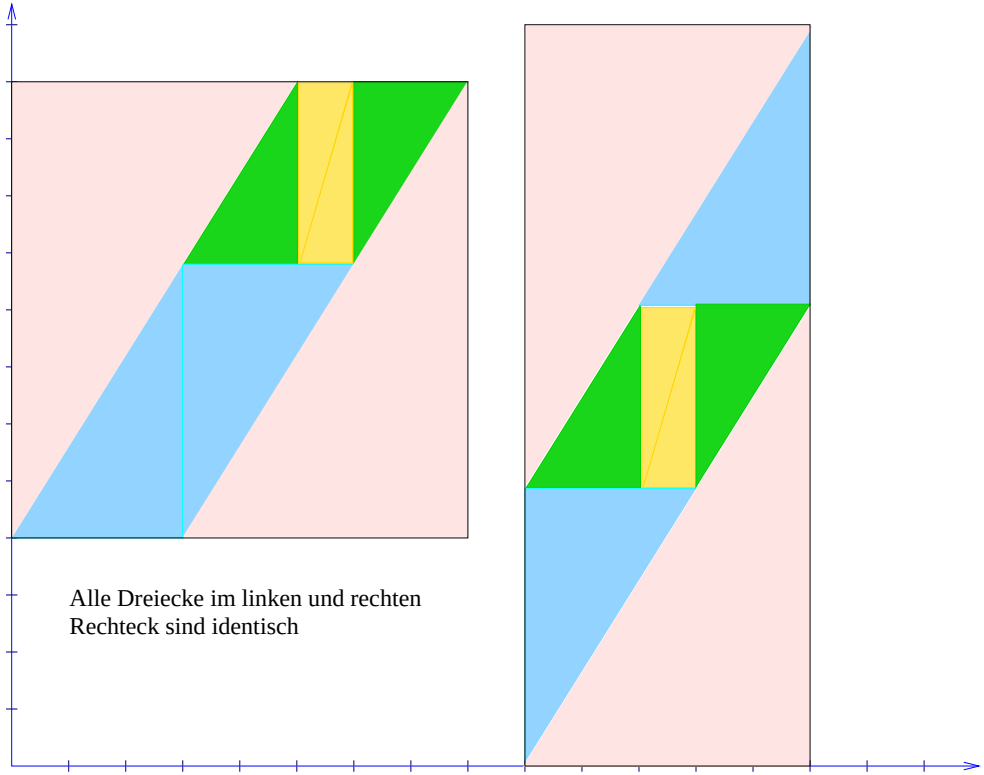
Man kann dasselbe sehr unterschiedlich beweisen

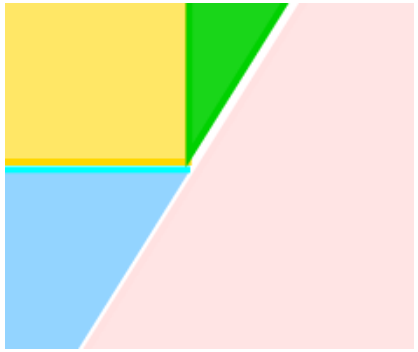
Es gibt zwei irrationale Zahlen x und y, deren Potenz rational ist

- **Standard-Beweis aus Mathematik-Lehrbuch**
  - *Wir wissen, daß $\sqrt{2}$ irrational ist. Betrachte $\sqrt{2}^{\sqrt{2}}$. Diese Zahl ist rational oder irrational. Ist $\sqrt{2}^{\sqrt{2}}$ rational, dann wähle $x{=}y{=}\sqrt{2}$. Sonst wähle $x{=}\sqrt{2}^{\sqrt{2}}$ und $y{=}\sqrt{2}$. Dann ist $x^y = \sqrt{2}^{(\sqrt{2}*\sqrt{2})} = 2$.*
  - Beweis ist korrekt, aber für viele etwas verwirrend. Wir wissen am Ende des Beweises nicht, welchen Wert x und y haben
- **Informatik-Beweise sollten "konstruktiv" sein, wo möglich**
  - *Wähle $x = \sqrt{2}$ und $y = 2{*}\log_2 3$. Wir wissen daß beide Zahlen irrational sind und es gilt $x^y = 2^{1/2*2*\log_2 3} = 3$.*
- **Es bleibt eine Frage offen**
  - *Wie können wir sicher sein, daß $\sqrt{2}$ und $\log_2 3$ irrational sind?* (Folie 15)

Beweise können auch Trugschlüsse enthalten

- **Geometrischer Beweis für Gleichheit von Flächen**





  - Beweis wirkt einleuchtend, aber das Ergebnis scheint unsinnig 64=65??
  - Wo ist der Fehler im Beweis?

- **Anschauung alleine ist kein Beweis**
  - Es muß auch gezeigt werden, daß die Dreiecke wirklich gleich sind
  - Für "Kongruenzbeweise" gibt es klare Kriterien

Eine präzise Sprache ist wichtig

- **Umgangssprache ist oft mißverständlich**
  - Fachspezifisches Vokabular und symbolische Notationen sind hilfreich
  - In der Informatik muß man etwas formaler sein, da Computer nur formale Objekte (Programme, Schaltungen) verarbeiten können
  - In sicherheitskritischen Anwendungen werden nur noch Beweise akzeptiert, die man auch mit Computern überprüfen kann
- *Quadrieren ist kleiner als Potenzieren*
  - Gemeint ist: Für jede natürliche Zahl x gilt $x^2 \leq 2^x$
  - Kürzere mathematische Schreibweise: $\forall x \in \mathbb{N}.\ x^2 \leq 2^x$
  - Die Behauptung läßt sich prüfen für $x = 0, 1, 2, 4, 5, 6, \ldots$
    Aber sie gilt nicht für $x = 3$, denn $3^2 = 9 > 8 = 2^3$
  - Korrekte Formulierung der Behauptung $\forall x \in \mathbb{N}.\ x \geq 4 \Rightarrow x^2 \leq 2^x$
  - Ein (rein) umgangssprachlicher Beweis hierfür ist schwer formulierbar
    Beweis benötigt mathematische Notation und Induktion (siehe Folie 19)

Formale Beweise sind nicht immer verständlicher

- **Formale Notation ist nur eine Notationshilfe**
  - Sie soll helfen, eine gute Idee klar zu formulieren
  - Formale Notation alleine macht einen Beweis noch nicht gut
- *Wenn drei ganze Zahlen sich jeweils um maximal 1 unterscheiden, dann sind zwei von ihnen gleich*
  - Informaler Beweis zeigt den wesentlichen Grund auf:

    Wenn alle drei Zahlen unterschiedlich wären, dann gäbe es eine kleinste, eine mittlere und eine größte und alle drei Zahlen unter schieden sich jeweils um mindestens 1. Damit wäre der Unterschied zwischen der kleinsten und der größten Zahl mindestens 2.
  - Formal: $\forall x, y, z \in \mathbb{N}.(|x-y| \leq 1 \wedge |x-z| \leq 1 \wedge |z-y| \leq 1) \Rightarrow x=y \vee x=z \vee z=y$

    Sei $x, y, z \in \mathbb{N}$ *beliebig. Es gelte* $|x-y| \leq 1 \wedge |x-z| \leq 1 \wedge |z-y| \leq 1$ *und o.B.d.A.* $x \leq y \leq z$. *Wenn* $x=y \vee x=z \vee z=y$ *nicht gilt, dann müsste* $|x-y|=1$, $|x-z| = 1$ *und* $|z-y| = 1$ *gelten. Es folgt* $z=y+1=x+2$, *also* $|x-z| = 2$. *Dies widerspricht aber der Annahme* $|x-z| \leq 1$. *Deswegen muß* $x=y \vee x=z \vee z=y$ *gelten.*

    Beweis zeigt mehr Details, ist aber nicht unbedingt klarer

Trotzdem müssen wir formale Beweise einüben

- **Sie sind das Handwerkzeug, daß jeder kennen muß**
  - Wenn man nicht sicher ist, ob ein Argument wirklich korrekt ist, zeigen formale Beweise die nötigen Details
  - Wenn einen die Intuition im Stich läßt, kann man alles genau aufschreiben und "durchrechnen"
  - Wenn man Aussagen ganz genau formalisiert, kann man die Beweise sogar mit Hilfe von Computern führen (Theorembeweiser)
- **In der Informatik dreht sich alles um Formales**
  - Programme und Fardware sind formale Objekte mit festen Regeln
  - Umgangsprachliche Beschreibungen des Verhaltens von Computern sind oft umständlicher als präzise mathematische Beschreibungen
  - Formale Beweise von Programmeigenschaften sind leichter als informale
  - Computer können die Korrektheit von sicherheitskritischer Software nur dann überprüfen, wenn Beweise vollständig formal geführt wurden
- **Wir betrachten hier zunächst nur sehr einfache Probleme**
  - Beweisführung läßt sich leichter trainieren, wenn die Materie vertraut ist und die Menge der Details überschaubar bleibt

Beweis durch Verkettung einfacher Argumente

Ziel: Zeige, daß eine Behauptung B aus Annahmen A folgt

- **Methodik: Kette von Aussagen $A_1, .., A_n = B$**
  - Zwischenaussagen A_i müssen schlüssig aus dem Vorhergehenden folgen
  - Dabei dürfen nur Annahmen aus A, Definitionen, bewiesene Aussagen, mathematische Gesetze, logische Schlußfolgerungen verwendet werden
  - Einfachste Form der Beweisführung, auch Deduktiver Beweis genannt
- **Beispiel: "Die Summe zweier ungerader Zahlen ist gerade"**
  - Informaler Beweis: *Es seien a und b zwei ungerade Zahlen. Per Definition ist $a = 2x + 1$ und $b = 2y + 1$ für gewisse x und y und die Summe ist $2(x + y + 1)$, also eine gerade Zahl.*
  - Schematische Darstellung ist oft kürzer und präziser

	Aussage	Begründung
1.	$a = 2x + 1$	Gegeben (Auflösung des Begriffs "ungerade")
2.	$b = 2y + 1$	Gegeben (Auflösung des Begriffs "ungerade")
3.	$a + b = 2(x + y + 1)$	(1,2) und Gesetze der Arithmetik

Die schematische Form ist eine Vorstufe maschinenprüfbarer Beweise, aber nicht "besser"

Oft reicht es, Definitionen aufzulösen

- **De Morgan Regel für Mengen: $A \cap B = \overline{\overline{A} \cup \overline{B}}$**
  - Beweis: *Sei x beliebig. Dann gilt*

    $x \in \overline{\overline{A} \cup \overline{B}} \Leftrightarrow x \notin \overline{A} \cup \overline{B} \Leftrightarrow x \notin \overline{A} \wedge x \notin \overline{B} \Leftrightarrow x \in A \wedge x \in B \Leftrightarrow x \in A \cap B$
  - Verwendete Definitionen:

    $A = B \Leftrightarrow \forall x. x \in A \Leftrightarrow x \in B$ (Extensionalitätsregel für Mengen)

    (Zwei Mengen sind gleich, wenn sie die gleichen Elemente haben)

    $x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$ (Durchschnitt von Mengen)

    $x \in A \cup B \Leftrightarrow x \in A \vee x \in B$ (Vereinigung von Mengen)

    $x \in \overline{A} \Leftrightarrow x \notin A$ (Komplement von Mengen)

- **Anwendung der Umkehrfunktion**

  Ist $f: A \to B$ injektiv, dann ist $f^{-1}(f(x)) = x$ für alle $x \in A$
  - *Die Umkehrfunktion f^{-1} ist definiert durch $f^{-1}(y) = z \Leftrightarrow f(z) = y$.*

    Für $y = f(x)$ gilt also $f^{-1}(f(x)) = z \Leftrightarrow f(z) = f(x)$.

    Da f injektiv ist, gilt $f(z) = f(x) \Leftrightarrow x = z$ und damit ist $f^{-1}(f(x)) = x$

  Gilt auch $f(f^{-1}(y)) = y$ für alle $y \in B$?

  Nein! f^{-1} ist nicht für alle $y \in B$ definiert, wenn f nicht auch surjektiv ist

Schematischer Beweis mit Definitionsauflösung

Wenn S endliche Teilmenge einer Menge U ist und das Komplement von S bezüglich U endlich ist, dann ist U endlich

- **Informales Argument beschreibt Kernidee des Beweises**
  - *U ist die Vereinigung zweier endlicher Mengen, also endlich*
- **Schematischer Beweis präsentiert alle Details**

Definition: S endlich $\equiv$ Es gibt eine ganze Zahl n mit $|S| = n$

T Komplement von S $\equiv$ $T \cup S = U$ und $T \cap S = \emptyset$

	Aussage	Begründung
1.	S endlich	Gegeben
2.	T Komplement von S	Gegeben
3.	T endlich	Gegeben
4.	$\|S\| = n$ für ein $n \in \mathbb{N}$	Auflösen der Definition in (1)
5.	$\|T\| = m$ für ein $m \in \mathbb{N}$	Auflösen der Definition in (3)
6.	$T \cup S = U$	Auflösen der Definition in (2)
7.	$T \cap S = \emptyset$	Auflösen der Definition in (2)
8.	$\|U\| = m + n$ für $n, m \in \mathbb{N}$	(4),(5),(6), (7) und Gesetze der Kardinalität
9.	U endlich	Einsetzen der Definition in (8)

In einfachen Beweisen ist das informale Argument oft einsichtiger

Beweisführung durch "Umkehrung"

- **Kontraposition**
  - Anstatt zu zeigen, daß die Behauptung B aus den Annahmen A folgt, beweise, daß nicht A aus der Annahme nicht B folgt
  - Aussagenlogisch ist $\neg B \Rightarrow \neg A$ äquivalent zu $A \Rightarrow B$
- **Häufigste Anwendung: Indirekte Beweisführung**
  - Zeige, daß aus nicht B und A ein Widerspruch (also $\neg A$) folgt
  - Aussagenlogisch ist $\neg(\neg B \wedge A)$ äquivalent zu $A \Rightarrow B$
  - Beispiel: *Wenn für eine natürliche Zahl* x *gilt* $x^2{>}1$, *dann ist* $x{\geq}2$

    Beweis: *Sei* $x^2{>}1$.

    Wenn $x{\geq}2$ *nicht gilt, dann ist* $x{=}1$ *oder* $x{=}0$.

    Wegen $1^2{=}1$ *und* $0^2{=}0$ *ist* $x^2{>}1$ *in beiden Fällen falsch.*

    Also muss $x{\geq}2$ *sein*

Diese Art Beweisführung erscheint zunächst unnatürlich und erfordert etwas Training

Widerlegungsbeweise
Zeige, dass eine Behauptung B nicht gilt

- **Widerspruchsbeweise zeigen, daß B niemals gelten kann**
  - Zeige, daß aus Annahme B ein Widerspruch folgt
  - Beispiel: *Ist S endliche Teilmenge einer unendlichen Menge U, dann ist das Komplement von S bezüglich U unendlich*
  - Beweisidee: *Wenn $\bar{S}$ endlich wäre, dann müsste auch U endlich sein.*[*]
  - Ausführlich: *Wir nehmen an $\bar{S}$ sei endlich. Da S ebenfalls endlich ist, muß U aufgrund des Satzes auf Folie 11 endlich sein. Dies ist ein Widerspruch, da U unendlich ist. Also ist die Annahme falsch und $\bar{S}$ ist nicht endlich*
- **Gegenbeispiele zeigen, daß B nicht immer wahr sein kann**
  - B ist nicht *allgemeingültig*, wenn es ein einziges Gegenbeispiel gibt
  - Beispiel: *Wenn x eine Primzahl ist, dann ist x ungerade* ist falsch
  - Gegenbeispiel: 2 ist eine gerade Zahl, die eine Primzahl ist

---

[*] Wir haben zugunsten einer einfacheren Schreibweise die Notation $\bar{S}$ als Abkürzung für "das Komplement von S bezüglich U" verwendet

Ein klassischer Widerspruchsbeweis

- *Es gibt unendlich viele Primzahlen*
  - *Wir nehmen an, es gäbe nur endlich viele Primzahlen* $p_1, .., p_{max}$.
    Dann ist jede natürliche Zahl $n \in \mathbb{N}$*, die größer als 1 ist, durch mindestens eine dieser Primzahlen teilbar.*
    Sei $x = 1 + \Pi_{i=1}^{max} p_i$ *das Produkt all dieser Primzahlen plus 1.*
    Dann ist x *durch keine der Zahlen* p_i *teilbar* (Beweis unten)
    Dies ist ein Widerspruch. Also ist die Annahme falsch
    und somit gibt es unendlich viele Primzahlen.
- *Für alle* $p \geq 2$ *und* $k \in \mathbb{N}$ *ist* $x = k*p+1$ *nicht durch* p *teilbar*
  - *Wir nehmen an* x *sei durch* p *teilbar, also* $x = j*p$ *für ein* $j \in \mathbb{N}$*. Dann muß* $j > k$ *sein, also* $j-k \geq 1$*, und es gilt* $1 = x-k*p = (j-k)*p \geq 1 * p \geq 2$
    Dies ist ein Widerspruch, also kann x *nicht durch* p *teilbar sein.*

Beweis der Irrationalität von Zahlen

- *Die Wurzel von 2 ist keine rationale Zahl*
  - *Wir nehmen an $\sqrt{2}$ sei rational, d.h. $\sqrt{2} = p/q$ für zwei teilerfremde Zahlen $p, q \in \mathbb{Z}$. Dann ist $p^2 = 2q^2$, also eine gerade Zahl.*

    Damit muß auch p gerade sein, also $p = 2r$ für ein $r \in \mathbb{Z}$.

    Dann ist aber auch $q^2 = 2r^2$, also ist q ebenfalls gerade.

    Damit sind p und q gerade, also nicht teilerfremd.

    Dies ist ein Widerspruch und damit kann $\sqrt{2}$ nicht rational sein.
- *Der Zweierlogarithmus von 3 ist keine rationale Zahl*
  - *Wir nehmen an $\log_2 3$ sei rational, d.h. $\log_2 3 = p/q$ für für zwei Zahlen $p, q \in \mathbb{Z}$. Dann ist $3^q = 2^p$, also eine gerade Zahl.*

    Dies ist ein Widerspruch, und damit kann $\log_2 3$ nicht rational sein.

$\mathbb{R}$ ist überabzählbar (mächtiger als $\mathbb{N}$)

- **Beweis verkettet mehrere Argumente**
  - Wir zeigen, daß es keine surjektive Funktion $f{:}\mathbb{N}{\to}(0,1)$ geben kann
  - Dann kann es auch keine Bijektion $f{:}\mathbb{N}{\to}(0,1)$ geben
  - Da $(0,1)$ und $\mathbb{R}$ gleichmächtig sind, gibt es keine Bijektion $f{:}\mathbb{N}{\to}\mathbb{R}$
  - Damit kann $\mathbb{R}$ nicht abzählbar sein
- *Es gibt keine surjektive Funktion $f{:}\mathbb{N}{\to}(0,1)$*
  - *Wir nehmen an, f existiert, d.h. für alle $x \in (0,1)$ gibt es ein $i \in \mathbb{N}$ mit $x = f(i)$. Wir konstruieren ein $z \in (0,1)$ das von allen $f(i)$ verschieden ist. Dazu bezeichnen wir die j-te Dezimalstelle einer Zahl $x \in (0,1)$ mit x_j und definieren $z_j = 4$, falls $f(j)_j = 5$ ist, und sonst $z_j = 5$. Damit ist $z_j \neq f(j)_j$ für alle j, also kann es kein $i \in \mathbb{N}$ geben mit $z = f(i)$.*
  - *Dies ist ein Widerspruch und somit gibt es kein surjektives $f{:}\mathbb{N}{\to}(0,1)$.*
- **Zusatzfrage: warum nicht einfach $z_j = 9 - f(j)_j$?**
  - Für $z = 0.4999999\ldots = 0.50000000..$ wäre $z_j = 9 - z_j$ kein Widerspruch
  - Auch wenn es unwahrscheinlich ist, daß wir genau diese Zahl erreichen, wäre der Beweis nicht mehr überzeugend

Das war ein sogenannter Diagonalbeweis

Spezielle Form von Widerlegungsbeweisen

Konstruktion von Gegenbeispielen für Aussagen über unendliche Objekte

Wichtig für Unmöglichkeitsbeweise in der Informatik

- **z.B. Terminierung von Programmen ist unentscheidbar**

  Es gibt kein Programm, das testen kann, ob ein beliebiges Programm bei einer bestimmten Eingabe überhaupt anhält

- **Beweis stützt sich auf wenige Grundannahmen**

  1. Programme und ihre Daten sind als Zahlen codierbar

     Schreibweise: $p_i(j) \mathrel{\hat{=}}$ Anwendung des i-ten Programms auf die Zahl j

  2. Computer sind universelle Maschinen

     Bei Eingabe von Programm und Daten berechnen sie das Ergebnis

  3. Man kann Programme beliebig zu neuen Programmen zusammensetzen

     ... und die Nummer des neuen Programms berechnen

Programmterminierung ist unentscheidbar

- **Annahme: es gibt ein Programm für den Terminierungstest**
  - `Term(`i`,`j`)`=1 falls $p_i(j)$ anhält (sonst 0)
- **Konstruiere ein Programm `Unsinn` wie folgt:**

$$\texttt{Unsinn}(i) = \begin{cases} 0 & \text{wenn } \texttt{Term}(i,i)=0 \\ \bot & \text{sonst} \end{cases}$$

- **Weil `Unsinn` ein Programm ist, muß es eine Nummer k haben**

	0	1	2	3	4	...
p_0	$\bot$	$\times$	$\times$	$\bot$	$\times$	...
p_1	$\bot$	$\times$	$\times$	$\times$	$\times$	...
p_2	$\times$	$\times$	$\times$	$\times$	$\times$	...
p_3	$\bot$	$\times$	$\bot$	$\not\times$	$\bot$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

$\times \mathrel{\hat{=}}$ Terminierung, $\bot \mathrel{\hat{=}}$ hält nicht

- **Was macht `Unsinn`=p_k bei Eingabe der eigenen Nummer als Daten?**
  - Wenn $p_k(k)$ hält, dann `Term`(k,k)=1, also hält `Unsinn`(k) nicht an **???**
  - Wenn $p_k(k)$ nicht hält, dann `Term`(k,k)=0, also hält `Unsinn`(k) an **???**
- **Dies ist ein Widerspruch,**

  Also kann es den Test auf Terminierung nicht geben

Induktive Beweise

Beweise eine Behauptung B für alle natürlichen Zahlen

- **Standardinduktion (beginnend ab i)**
  - "Gilt B für i und B für $n{+}1$, wenn B für n gilt, dann gilt B für alle $n{\geq}i$"
  - Beispiel: *Für alle* $x{\geq}4$ *gilt* $2^x{\geq}x^2$

    Induktionsanfang $x{=}4$: *Es ist* $2^x = 16 \geq 16 = x^2$

    Induktionsannahme: *Es gelte* $2^n{\geq}n^2$ *für ein beliebiges* $n{\geq}4$

    Induktionsschritt: *Es ist* $2^{n+1} = 2{*}2^n \geq 2n^2$ (aufgrund der Induktionsannahme)

    und $(n{+}1)^2 = n^2{+}2n{+}1 = n(n{+}2{+}\frac{1}{n}) \leq n(n{+}n) = 2n^2$ (wegen $n{\geq}4$)

    also gilt $2^{n+1} \geq (n{+}1)^2$

  Aufgrund des Induktionsprinzips gilt damit $2^n{\geq}n^2$ für alle $n{\geq}4$

- **Vollständige Induktion**
  - "Folgt B für n, wenn B für alle $i{\leq}j{<}n$ gilt, dann gilt B für alle $n{\geq}i$"
  - Mächtiger, da man nicht den unmittelbaren Vorgänger benutzen muss

Zwei wichtige Summenformeln

- **Gauß'sche Summenformel:** $\sum_{i=1}^{n} i = n(n+1)/2$

  Induktionsanfang $n = 0$: *Es ist* $\sum_{i=1}^{0} i = 0 = 0(0+1)/2$

  Induktionsannahme: *Es gelte* $\sum_{i=1}^{m} i = m(m+1)/2$ *für ein* $m \in \mathbb{N}$

  Induktionsschritt: *Es sei* $n = m+1$.

  Dann ist $\sum_{i=1}^{n} i = \sum_{i=1}^{m} i + (m+1) = m(m+1)/2 + (m+1)$
  $= (m+1)(m+2)/2 = n(n+1)/2$

  Aufgrund des Induktionsprinzips gilt $\sum_{i=1}^{n} i = n(n+1)/2$ für alle $n \in \mathbb{N}$

- **Die "Informatik-Formel":** $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$

  (Beschreibt die größte mit n Bit darstellbare Zahl)

  Induktionsanfang $n = 0$: *Es ist* $\sum_{i=1}^{0} 2^{i-1} = 0 = 2^0 - 1$

  Induktionsannahme: *Es gelte* $\sum_{i=1}^{m} 2^{i-1} = 2^m - 1$ *für ein* $m \in \mathbb{N}$

  Induktionsschritt: *Es sei* $n = m+1$.

  Dann ist $\sum_{i=1}^{n} 2^{i-1} = \sum_{i=1}^{m} 2^{i-1} + 2^{m+1-1} = 2^m - 1 + 2^m$
  $= 2^{m+1} - 1 = 2^n - 1$

  Aufgrund des Induktionsprinzips gilt $\sum_{i=1}^{n} i = n(n+1)/2$ für alle $n \in \mathbb{N}$

Das Briefmarken Problem

Ist es möglich, jedes Porto ab 8 Cent nur mit 3c und 5c Briefmarken zu erzeugen?

- **Lösungsidee** 8c =  , 9c =  , 10c =  ,

  11c =  , ...

- **Informales Argument ist leicht einzusehen**
  - Basisfälle 8, 9, 10 sind lösbar wie oben illustriert.
  - Lösung für größere n wird aus der für $n-3$ mit weiteren 3c erzeugt

- **Induktionsbeweis benötigt vollständige Induktion**

  Behauptung $B(n)$ (für $n \geq 8$): *Es gibt* $a, b \in \mathbb{N}$ *mit* $n = 3 \cdot a + 5 \cdot b$

  Induktionsannahme: *Sei* $n \geq 8$ *beliebig. Es gelte* $B(j)$ *für alle* $8 \leq j < n$

  Induktionsschritt: *Falls* $n = 8, 9, 10$ *wähle* (a, b) = *(1,1), (3,0), bzw. (0,2).*

  Ansonsten gibt es für $j = n-3 \geq 8$ *Zahlen* a', b' *mit* $j = 3 \cdot a' + 5 \cdot b'$. (IA)

  Es folgt $n = j+3 = 3 \cdot (a'+1) + 5 \cdot b'$. *Wähle* $(a, b) = (a'+1, b')$

  Aufgrund des Prinzips der vollständigen Induktion folgt $B(n)$ *für alle* $n \geq 8$

Existenz einer ganzzahligen Quadratwurzel

Für alle $n \in \mathbb{N}$ *existiert ein* $r \in \mathbb{N}$ *mit* $r^2 \leq n < (r+1)^2$

- **Beweis mit Standardinduktion**

  Behauptung $B(n)$: *es existiert ein* $r \in \mathbb{N}$ *mit* $r^2 \leq n < (r+1)^2$

  Induktionsanfang $n = 0$: *wähle* $r = 0$, *dann gilt* $0^2 \leq 0 < (0+1)^2$, *also* $B(0)$.

  Induktionsannahme: *Es gelte* $B(m)$ *für ein* $m \in \mathbb{N}$

  Induktionsschritt: *Es sei* $n = m+1$.

  Wegen $B(m)$ *gibt es ein* r_m *mit* $r_m^2 \leq m < (r_m+1)^2$

  Falls $(r_m+1)^2 \leq m+1 = n$ *dann wähle* $r = r_m+1$ *und sonst* $r = r_m$

  In beiden Fällen gilt $r^2 \leq n < (r+1)^2$ (ausrechnen!), *also* $B(n)$

  Aufgrund des Induktionsprinzips gilt $B(n)$ für alle $n \in \mathbb{N}$

- **Beweis mit vollständiger Induktion**

  Induktionsannahme: *Es gelte* $B(m)$ *für alle* $m < n$

  Induktionsschritt: *Wegen* $B(n \div 4)$ *gibt es ein* r_0 *mit* $r_0^2 \leq n \div 4 < (r_0+1)^2$

  Falls $(2r_0+1)^2 \leq n$ *dann wähle* $r = 2r_0+1$ *und sonst* $r = 2r_0$

  In beiden Fällen gilt $r^2 \leq n < (r+1)^2$ (ausrechnen!), *also* $B(n)$

Der Beweis hat eine ähnliche Struktur, springt aber weiter zurück. Er enthält implizit einen sehr effizienten Algorithmus zur Berechnung von $\lfloor \sqrt{n} \rfloor$. Wie sieht dieser aus??

Simultane (gleichzeitige) Induktion

.. wenn mehrere Aussagen voneinander abhängen

- **Häufig bei rekursiv definierten Funktionen**
  - Beispiel: Sei $f(0) = 0$ und $f(n+1) = 1-f(n)$
  - Zeige: $f(n) = 0$, falls n gerade und sonst 1
- **Problem beinhaltet zwei zusammengehörige Aussagen**

  $B_1(n)$: *Für gerade* n *ist* $f(n)=0$, $B_2(n)$: *Für ungerade* n *ist* $f(n)=1$
- **Induktionsbeweis zeigt beide Aussagen gleichzeitig**

  Induktionsanfang $n=0$: n *ist gerade und es ist* $f(0)=0$, *also gilt* $B_1(0)$

  n *ist nicht ungerade, also gilt* $B_2(0)$ (Voraussetzung nicht erfüllt)

  Induktionsannahme: *Es gelte* $B_1(n)$ *und* $B_2(n)$.

  Induktionsschritt: *Falls* $n+1$ *gerade ist, dann ist* n *ungerade*

  und es gilt $f(n)=1$ *wegen* $B_2(n)$ *also* $f(n+1)=0$ *und damit* $B_1(n+1)$

  Ausserdem gilt $B_2(n+1)$, *da* $n+1$ *nicht ungerade ist*

  Ansonsten ist n *gerade und* $f(n+1)=1$ *wegen* $B_1(n)$, *also gilt* $B_2(n+1)$

  Ausserdem gilt $B_1(n+1)$, *da* $n+1$ *nicht gerade ist*

  Aufgrund des Induktionsprinzips folgen beide Behauptungen für alle n

Simultane Induktion ausformuliert $f(0)=0,\ f(n+1)=1-f(n)$

Um zu zeigen, daß $f(n) = 0$, falls n gerade und sonst 1 ist, zeigen wir durch Induktion, daß für alle $n \in \mathbb{N}$ die folgenden beiden Aussagen gelten

$B_1(n)$: n gerade $\Leftrightarrow$ $f(n)=0$

$B_2(n)$: n ungerade $\Leftrightarrow$ $f(n)=1$

Induktionsanfang $n=0$:

$B_1(n)$: 0 ist gerade und nach Definition ist $f(0) = 0$, also gilt Aussage $B_1(n)$.

$B_2(n)$: 0 ist nicht ungerade und nach Definition ist $f(0)\neq 1$, also gilt die Äquivalenz $B_2(n)$, da jeweils die rechte und linke Seite falsch ist.

Induktionsannnahme: Es gelte $B_1(m)$ und $B_2(m)$ für ein beliebiges $m \in \mathbb{N}$.

Induktionsschritt: Es sei $n = m+1$.

$B_1(n)$: Es ist $n = m+1$ gerade $\Leftrightarrow$ m ist ungerade

$\Leftrightarrow$ $f(m) = 1$ (Induktionsannahme $B_2(m)$)

$\Leftrightarrow$ $f(n) = 1-f(m) = 0$

$B_2(n)$: Es ist $n = m+1$ ungerade $\Leftrightarrow$ m ist gerade

$\Leftrightarrow$ $f(m) = 0$ (Induktionsannahme $B_1(m)$)

$\Leftrightarrow$ $f(n) = 1-f(m) = 1$

Aufgrund des Induktionsprinzips gilt $B_1(n)$ und $B_2(n)$ für alle $n \in \mathbb{N}$

Existenz eines grössten gemeinsamen Teilers

Für alle $a, b \in \mathbb{N}$ existiert ein größter gemeinsamer Teiler $x \in \mathbb{N}$

- **Wann ist x der größte gemeinsame Teiler von a und b**
  – x teilt a, x teilt b, und für jedes y, das a und b teilt, gilt $y \leq x$
- **Beweis benötigt vollständige Induktion auf Paaren**
  Behauptung $B(a,b)$: *es gibt einen größten gemeinsamen Teiler x von a und b*
  Induktionsannahme: *Sei a, b beliebig. $B(a',b')$ gelte für alle $(a',b') < (a,b)$**
  Induktionsschritt: *Falls $a = b$ dann wähle $x = a$.*
  Falls $a > b$ dann gibt es wegen $B(a-b, b)$ einen größten gemeinsamen Teiler x_1 von $a-b$ und b. Wähle $x = x_1$.
  Falls $a < b$ dann gibt es wegen $B(a, b-a)$ einen größten gemeinsamen Teiler x_2 von a und $b-a$. Wähle $x = x_2$.
  In allen drei Fällen ist x der größte gemeinsame Teiler von a und b.
  Aufgrund des Induktionsprinzips folgt $B(a,b)$ für alle $a, b \in \mathbb{N}$
  Der Beweis enthält implizit einen sehr effizienten Algorithmus zur Berechnung von $ggT(a,b)$

*Es ist $(a',b') < (a,b)$ genau dann wenn $a' < a$ oder $a' = a$ und $b' < b$

Induktionsbeweise für andere Datenstrukturen

Viele Datenstrukturen lassen sich induktiv programmieren

- **$\mathbb{N}$: Natürliche Zahlen**
  - $0 \in \mathbb{N}$ (Null ist eine natürliche Zahl)
  - $n \in \mathbb{N} \Rightarrow n+1 \in \mathbb{N}$ (jeder Nachfolger einer natürlichen Zahl ist eine natürliche Zahl)
- **List T: Listen über einem Datentyp T**
  - $[\,] \in \text{List } T$ (leere Liste - ohne Elemente)
  - $l \in \text{List } T \wedge x \in T \Rightarrow x :: l \in \text{List } T$ (Voranstellen eines Elements aus T)
- **Σ^*: Wörter (Strings) über einem Alphabet Σ**
  - $\epsilon \in \Sigma^*$ (leeres Wort - ohne Symbole)
  - $w \in \Sigma^* \wedge a \in \Sigma \Rightarrow w\,a \in \Sigma^*$ (Anhängen eines Symbols aus Σ an ein Wort)
- **Tree T: Bäume mit Markierungen aus T**
  - $x \in T \Rightarrow x \in \text{Tree } T$ (einzelner Knoten mit Markierung x)
  - $x \in T \wedge t_1..t_n \in \text{Tree } T \Rightarrow (x, [t_1, .., t_n]) \in \text{Tree } T$ (Baum mit Wurzel x und Unterbäumen $t_1..t_n$)

Induktionsbeweise folgen der Struktur dieses Aufbaus

Strukturelle Induktion

Beweis auf Basis der Struktur eines induktiven Datentyps

Gilt Behauptung B für das Basiselement des Typs und für ein zusammengesetztes Element, wenn B für seine Unterelemente gilt, dann gilt B für alle Elemente des Typs

– Zahlen: Gilt $B(0)$ und $B(n{+}1)$, wenn $B(n)$ gilt, dann gilt $\forall x \in \mathbb{N}.B(x)$

– Listen: $B([\,]) \wedge (B(l) \Rightarrow \forall x \in T.B(x :: l)) \Rightarrow \forall l \in \text{List}\, T.B(l)$

– Wörter: $B(\epsilon) \wedge (B(w) \Rightarrow \forall a \in \Sigma.B(w\,a)) \Rightarrow \forall w \in \Sigma^*.B(w)$

Häufig eingesetzt für Analyse von Listen- und Baumstrukturen (Suchen, Sortieren, ...) oder von syntaktischen Strukturen (Formeln, Programmiersprachen, ...)

- **Beispiel:** *Die Summe einer Liste l von positiven ganzen Zahlen ist mindestens so groß wie ihre Länge*

Induktionsanfang $l = [\,]$: *Die Summe und die Länge von l sind 0*

Induktionsannahme: *Es gelte* $sum(l) \geq |l|$

Induktionsschritt: *Sei* $x \in \mathbb{Z}^+$ *und* $l' = x :: l$

Dann gilt $sum(l') = sum(l){+}x \geq sum(l){+}1 \geq |l|{+}1 = |l'|$ (IA)

Aufgrund des Induktionsprinzips folgt die Behauptung für alle Listen

Strukturelle Induktion auf Strings

- *Für alle Strings* $u, v \in \Sigma^*$ *gilt* $|u\,v| = |u|+|v|$

  Die Länge der Konkatenation $u\,v$ zweier Strings u und v ist die Summe der Einzellängen $|u|$ und $|v|$

  Wir führen eine strukturelle Induktion über den zweiten String v

  Behauptung $B(v)$: *Für alle* $u \in \Sigma^*$ *gilt* $|u\,v| = |u|+|v|$

  Induktionsanfang $v = \epsilon$: *Es ist* $|u\,v| = |u\,\epsilon| = |u| = |u|+0 = |u|+|\epsilon| = |u|+|v|$

  also gilt $(B(\epsilon))$

  Induktionsannahme: Es gelte $B(w)$ für ein $w \in \Sigma^*$

  Induktionsschritt: *Es sei* $a \in \Sigma$ *beliebig und* $v = wa$.

  Dann ist $|u\,v| = |u\,w\,a| = |u\,w|+1 =_{IA} |u|+|w|+1 = |u|+|w\,a| = |u|+|v|$

  also gilt $(B(v))$

  Wegen des Induktionsprinzips folgt die Behauptung $B(v)$ für alle $v \in \Sigma^*$

Strukturelle Induktion auf Binärbäumen

- **Binärbäume haben zwei Nachfolger pro Knoten**
  - $x \in T \Rightarrow x \in$ BTree T
  - $x \in T \wedge t_1, t_2 \in$ BTree $T \Rightarrow (x, [t_1, t_2]) \in$ BTree T
  - Anzahl der Knoten: $\#(x) = 1,\ \#(x, [t_1, t_2]) = \#(t_1)+\#(t_2)+1$
  - Tiefe eines Baums: $d(x) = 1,\ d(x, [t_1, t_2]) = max(d(t_1), d(t_2))+1$
- **Für jeden Binärbaum t gilt $\#(t) \leq 2^{d(t)}-1$**

  Induktionsanfang $t = x$: *Tiefe und Anzahl der Knoten ist* $1 \leq 2^1-1$

  Induktionsannahme: *Es gelte* $\#(t_1) \leq 2^{d(t_1)}-1$ *und* $\#(t_2) \leq 2^{d(t_2)}-1$

  Induktionsschritt: *Sei* $t = (x, [t_1, t_2])$. *Dann gilt*

$$
\begin{aligned}
\#(t) &= \#(t_1)+\#(t_2)+1 \\
&\leq (2^{d(t_1)}-1) + (2^{d(t_2)}-1) + 1 && \text{(IA)} \\
&\leq (2^{max(d(t_1),d(t_2))}-1) + (2^{max(d(t_1),d(t_2))}-1) + 1 && d(t_1) \leq max(d(t_1), d(t_2)) \\
&= 2 * 2^{max(d(t_1),d(t_2))}-1 = 2^{d(t)}-1
\end{aligned}
$$

  Wegen des Induktionsprinzips folgt die Behauptung für alle Binärbäume

Wie genau/formal muss ein Beweis sein?

- **"Ein Beweis ist ein Argument, das den Leser überzeugt"**

  Beweisidee muß klar erkennbar und einsichtig sein

  Beweis muß genau genug, um Details rekonstruieren zu können

  Formal und detailliert ist nicht immer besser (vgl. Briefmarkenproblem)

- Text muß präzise Sprache verwenden, lesbar und klar verständlich sein

  Formeln / Textfragmente ohne erkennbaren Sinn aneinanderzureihen ist unakzeptabel

  Zwischenschritte müssen mit "üblichen" Vorkenntnissen erklärbar sein

- Gedankensprünge sind erlaubt, wenn Sie die Materie gut genug verstehen, dass Sie nichts mehr falsch machen können

  ... es reicht nicht, dass Sie es einmal richtig gemacht haben

---

- Tip: ausführliche Lösungen entwickeln, bis Sie genug Erfahrung haben.

  Bei Präsentation für Andere: zentrale Gedanken aus Lösung extrahieren

- Test: verstehen Kommilitonen Ihre Lösung und warum sie funktioniert?

Mehr dazu in den Übungen

Methodik des Problemlösens

- **Klärung der Voraussetzungen**
  - Welche Begriffe sind zum Verständnis des Problems erforderlich?
  - Erstellung eines präzisen Modells: abstrahiere von Details
  - Formulierung des Problems im Modell: was genau ist zu tun?
- **Lösungsweg konkretisieren**
  - Welche Einzelschritte benötigt man, um das Problem zu lösen?
  - Welches Gesamtergebnis ergibt sich aus den Einzelschritten?
  - Wie beweist man die Korrektheit des Gesamtergebnisses?
- **Lösung zusammenfassen**
  - Kurz und prägnant: Argumente auf das Wesentliche beschränken
  - Wo möglich mathematisch präzise Formulierungen verwenden

ANHANG

- **Notation für logische Operationen**
  - P, Q, R oder $P(x), Q(x)$ sind Platzhalter für elementare Aussagen
  - Aussagenlogische Formeln: $\neg P$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$
  - Quantoren: $\forall x.P(x)$, $\exists x.P(x)$
- **Notation für mengentheoretische Konzepte**
  - S, M sind typische Platzhalter für Mengen
  - Mengenzugehörigkeit: $x \in S$
  - Beschreibung einer Menge durch Eigenschaften: $\{x \mid P(x)\}$
  - Konstruktion durch Eigenschaften und Funktionen: $\{f(x) \mid P(x)\}$
    ... mit Auswahl aus einer Menge: $\{x \in S \mid P(x)\}$, $\{f(x) \mid x \in S\}$, ...
  - Quantifizierung über Elemente einer Menge: $\forall x \in S.P(x)$, $\exists x \in S.P(x)$
  - Vereinigung: $S_1 \cup S_2 = \{x \mid x \in S_1 \vee x \in S_2\}$
  - Durchschnitt: $S_1 \cap S_2 = \{x \mid x \in S_1 \wedge x \in S_2\}$
  - Differenz : $S_1 - S_2 = \{x \mid x \in S_1 \wedge x \notin S_2\}$
  - Komplement: $\overline{S} = \{x \mid x \notin S\}$
  - Teilmenge $S_1 \subseteq S_2$, echte Teilmenge $S_1 \subset S_2$, Obermenge $S_1 \supseteq S_2$, ...

Mathematisches Vokabular: Funktionen

– Funktion $f : S \to S'$: Abbildung zwischen Grundmengen S und S'
nicht unbedingt auf allen Elementen von S definiert ($\perp$ = undefiniert)

– Domain von f: $\mathit{domain}(f) = \{x \in S \mid f(x) \text{ definiert}\}$ (Definitionsbereich)

– Range von f: $\mathit{range}(f) = \{y \in S' \mid \exists x \in S.\ f(x) = y\}$ (Wertebereich)

– f total: $\mathit{domain}(f) = S$ (andernfalls ist f partiell)

– f injektiv: $x \neq y \Rightarrow f(x) \neq f(y)$

– f surjektiv: $\mathit{range}(f) = S'$

– f bijektiv: f injektiv und surjektiv

– Umkehrfunktion $f^{-1}: S' \to S$: $f^{-1}(y) = x \Leftrightarrow f(x) = y$ (f muß injektiv sein)

– Urbild $f^{-1}(M) = \{x \in S \mid f(x) \in M\}$ (eine Menge, f muß nicht injektiv sein)

– Charakteristische Funktion χ_M von $M \subseteq S$:
$$\chi_M(x) = \begin{cases} 1 & \text{falls } x \in M, \\ 0 & \text{sonst} \end{cases}$$

– Partiell-charakteristische Funktion ψ_M:
$$\psi_M(x) = \begin{cases} 1 & \text{falls } x \in M, \\ \perp & \text{sonst} \end{cases}$$

Mehr Vokabular wird bei Bedarf vorgestellt

Mathematisches Vokabular: Listen

List T: Datenstruktur mit Elementen aus T

- $[x_1, .., x_n]$: geordnete Folge mit Elementen $x_1, .., x_n \in T$
- $[\,]$: leere Liste (ohne Elemente)
- $x :: l$: Voranstellen eines Elements aus T vor Liste l
- $hd(l)$: Kopf (erstes Element) der Liste l $(hd(x :: l) = x)$
- $tl(l)$: Rest der Liste l (nach Entfernen des Kopfes) $(tl(x :: l) = l)$
- $rev(l)$: Umkehrung der Liste l $(rev([x_1, .., x_n]) = [x_n, .., x_1])$
- $l_1 \circ l_2$: Konkatenation zweier Listen $([x_1, .., x_n] \circ [y_1, .., y_m] = [x_1, .., x_n, y_1, .., y_m])$
- $|l|$: Länge der Liste l $(|[x_1, .., x_n]| = n)$
- $map(f, l)$: Anwendung von f auf alle Listenelemente

  $(map(f, [x_1, .., x_n]) = [f(x_1), .., f(x_n)]))$
- $l[n]$ (oder l_n): n-tes Element von l $([x_1, .., x_n][m] = x_m)$

Viele weitere Konzepte definierbar

Mathematisches Vokabular: Wörter und Sprachen

– Alphabet Σ: endliche Menge von Symbolen,
  z.B. $\Sigma = \{0, 1\}$, $\Sigma = \{0, .., 9\}$, $\Sigma = \{A, .., Z, a, .., z, \ , ?, !, ..\}$
– Wort (String, Zeichenreihe): endliche Folge w von Symbolen eines Alphabets
– Σ^*: Menge aller Wörter über Σ
– Σ^+: Menge aller nichtleeren Wörter über Σ
– Σ^k: Menge der Wörter der Länge k mit Symbolen aus Σ
– ϵ: Leeres Wort (ohne jedes Symbol)
– $w\ a$: Anhängen eines Symbols a an das Wort w
– $w\ v$: Konkatenation (Aneinanderhängung) der Wörter w und v
– u^i: i-fache Konkatenation des Wortes (oder Symbols) u
– $|w|$: Länge des Wortes w (Anzahl der Symbole)
– $v \sqsubseteq w$: v Präfix von w, wenn $w = v\,u$ für ein Wort u
– Sprache L: Beliebige Menge von Wörtern über einem Alphabet Σ
  Üblicherweise in abstrakter Mengennotation gegeben
  z.B. $\{w \in \{0, 1\}^* \mid |w| \text{ ist gerade}\}$ $\{0^n 1^n \mid n \in \mathbb{N}\}$
– Eine Sprachen kann auch als Problem P bezeichnet werden

(Das “Problem” ist in diesem Fall, die Zugehörigkeit zur Menge P zu testen)

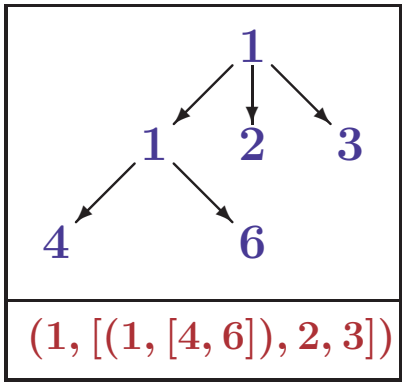
- **Gerichteter Graph: Datenstruktur $G = (V, E)$**
  - V endliche Menge von Knoten, $E \subseteq \{ (v, v') \in V \times V \mid v \neq v' \}$ ("Kanten")
  - Nachfolger von v: durch Kanten verbundene Knoten aus $\{v' \in V \mid (v, v') \in E\}$
  - Nachkommen: transitive Hülle der Nachfolgerrelation
  - Grad eines Knotens: Anzahl der direkten Nachfolger
  - Pfad (von v_1 nach v_n): Folge $[v_1, .., v_n]$ durch Kanten verbundener Knoten
  - v' erreichbar von v: es gibt einen Pfad von u nach v
  - Kreis: Pfad mit identischen Start- und Endknoten (alle anderen sind verschieden)
  - H Subgraph von G ($H \sqsubseteq G$): Knoten und Kanten von H gehören zu G
  - H isomorph zu G ($H \cong G$): Umbenennung der Knoten vn H erzeugt G

  Konzepte analog für ungerichtete Graphen (Kanten haben keine Richtung)

- **Lohnenswerte Literatur**
  - S. Krumke, H. Noltemeier: *Graphentheoretische Konzepte und Algorithmen*, Teubner 2005.
  - C. Meinel, M. Mundhenk: *Mathematische Grundlagen der Informatik*, Teubner 2002.
  - K. Denecke: *Algebra und Diskrete Mathematik für Informatiker*, Teubner 2003.

Mathematisches Vokabular: Bäume

- **Tree T: Datenstruktur mit Elementen aus T**
  - x: Einzelknoten wit Wurzel $x \in T$
  - $(x, [t_1, .., t_n])$: Baum mit Wurzel x und Unterbäumen $t_1..t_n$
- **Darstellbar als kreisfreier Graph mit markierten Knoten**
  - Elemente von T sind Knoten des Baums
  - Kanten verbinden Knoten mit Wurzeln der Unterbäume
  - "Nachfolger" eines Knotens sind geordnet
  - Wurzel: Knoten ohne Vorgänger
  - Blatt / Innerer Knoten: Knoten ohne/mit Nachfolger
  - Binärbaum: Baum, in dem jeder innere Knoten Grad 2 hat
  - Tiefe: Länge des längsten Pfades im Baum



- **Wichtige Eigenschaften**
  - Von der Wurzel gibt es zu jedem Knoten genau einen Pfad
  - Ein Baum mit n Knoten hat $n-1$ Kanten
  - Ein Binärbaum mit n Knoten hat mindestens Tiefe $\lceil \log_2(n+1) \rceil$