Theoretische Informatik I

Wintersemester 2012/13

Christoph Kreitz / Sebastian Böhne

Theoretische Informatik

{kreitz,boehne}@cs.uni-potsdam.de

http://cs.uni-potsdam.de/ti1-ws1213

1. Lehrziele und Lernformen
2. Organisatorisches
3. Gedanken zur Arbeitsethik

Theoretische Informatik: Worum geht es?

Die Wissenschaft von der Berechnung

- **Analyse von Grundsatzfragen**
  - Was kann man mit Computern lösen, was nicht?
  - Für welche Probleme gibt es gute, allgemeingültige Verfahren?
  - Welche Probleme sind effizient lösbar – welche nicht?
  - Wie flexibel kann man Programmiersprachen gestalten?
  - Wie hängen verschiedenartige Computerarchitekturen zusammen?
- **Mathematische Vorgehensweise**
  - Abstraktion von irrelevanten Details (Hardware/Programmiersprache)
  - Erkenntnisse sind beweisbar und haben weitreichende Gültigkeit
- **Der älteste Zweig der Informatik**
  - Theoretische Erkenntnisse gab es lange vor den ersten Computern
  - Aussagen sind langlebiger als in den anderen Informatikzweigen
  - Aber: Erkenntnisse sind selten “unmittelbar” anwendbar

Wozu soll das gut sein?

Denken ist besser als Hacken

- **Verständnis allgemeiner Zusammenhänge**
  - Details ändern sich schnell, Modelle und Methoden nicht
- **Effizientere Arbeitsweise**
  - Abstraktion richtet den Blick auf das Wesentliche
    Wenn Sie zu sehr auf Details schauen, verlieren Sie die Übersicht
  - Abstrakte Lösungen sind auf spezifische Situationen übertragbar
- **Vermeidung der gröbsten Fehler**
  - z.B. Korrektheit von Software kann nicht getestet werden
    Optimale Navigation ist nicht effizient möglich
    Flexible Programmiersprachen sind nicht (effizient) compilierbar
  - Viele Programmierer haben sich schon in unlösbare Probleme verbissen, weil sie das nicht wußten oder nicht geglaubt haben

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
  - Berechenbarkeitsmodelle
  - Aufzählbarkeit, Entscheidbarkeit, Unlösbare Probleme
- **Komplexitätstheorie** TI-2
  - Komplexitätsmaße und -klassen für Algorithmen und Probleme
  - Nicht handhabbare Probleme ($\mathcal{NP}$-Vollständigkeit)
  - Effiziente Alternativen zu konventionellen Verfahren

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 "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 optionale Tutorium wöchentlich Do 14:15–15:45

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

  ... aber nicht Lösung der Hausaufgaben

Das Team

Christoph Kreitz

Raum 1.18, Telephon 3060

kreitz@cs.uni-potsdam.de

Sebastian Böhne

Raum 1.23, Telephon 3069

boehne@uni-potsdam.de

Tutoren

Charlotte Gerlitz

Fred Brumm

Hannes Schröder

Marc Preußner

Mario Frank

Margrit Dittmann

Organisatorisches

- **Zielgruppe: ab 1. Semester**
  - Bei geringen mathematischen Vorkenntnissen besser ab 3. Semester
  - Oft ist es sinnvoll erst an Mathematikveranstaltungen teilzunehmen
- **Vorlesung**
  - Wöchentlich Fr 8:15–9:45
- **Tutorium** (optional)
  - Besprechung von allgemeinen Fragen und schwierigen Hausaufgaben
  - Wöchentlich Do 14:15–15:45, ab 25. Oktober
- **Übungen**
  - 8 Gruppen, wöchentlich (Montags – Mittwochs) je 2 Stunden
- **Sprechstunden**
  - C. Kreitz: Fr 10:30–11:30 . . . und immer wenn die Türe offen ist
  - S. Böhne: Do 10:00–11:30
  - Tutoren: individuell in Übungsgruppen vereinbaren

Leistungserfassung

- **Eine Klausur entscheidet die Note** (Anmeldung erforderlich!)
  - Hauptklausur 15. Februar 2013, 8:30–11:30 Uhr
  - Probeklausur 21. Dezember 2012, 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**
  - 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**
  - Deduktive Beweise für Analyse von Befehlssequenzen
  - Induktionsbeweise für Analyse von Rekursion / Schleifen
  - Widerlegungsbeweise / Gegenbeispiele für Unmöglichkeitsaussagen

Mehr dazu nach der Pause

Lernerfolg

- **kommt nicht durch passive Teilnahme**
  - Downloads von Folien, Übungen, Videos, ...
  - Abspielen der Vorlesungsvideos
  - Durchlesen der Musterlösungen / Überfliegen der Folien
  - Physische Anwesenheit in Vorlesung und Übung
  - "Bulimielernen" kurz vor der Klausur
- **sondern nur durch eigene Aktivität**
  - Geistige Anwesenheit in Vorlesung und Übung
  - Erstellen und Überarbeiten von Notizen zur Vorlesung *(aktives Lesen)*
  - Durcharbeiten von Notizen/Folien/Buch/Webseiten vor der Vorlesung
  - Erarbeitung von Lösungsideen zu Aufgaben vor der Übungsstunde
  - "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

Nutzen Sie Ihre Chancen!

- **Theorie ist bedeutender als viele glauben**
  - Ist Theorie langweilig? überflüssig? unverständlich? . . . eine Plage?
  - Alle großen Softwareprojekte benutzten theoretische Modelle
  - Ohne theoretische Kenntnisse begehen Sie viele elementare Fehler
  - Theorie kann durchaus sehr interessant sein
- **Es geht um mehr als nur bestehen**
  - Das wichtige ist Verstehen
  - Sie können jetzt umsonst lernen, was später teure Lehrgänge benötigt
  - Wann kommen Sie je wieder mit den Besten des Gebietes in Kontakt?
- **Die Türe steht offen**
  - Lernfrust und mangelnder Durchblick sind normal aber heilbar
  - Kommen Sie in die Sprechstunden und stellen Sie Fragen

Vertrauen ist ein kostbares Gut

... missbrauchen Sie es nicht

- **Abschreiben fremder Lösungen bringt nichts**
  - Sie lernen nichts dabei – weder Inhalt noch Durchhaltevermögen
  - Sie erkennen Ihre Lücken nicht und nehmen Hilfe zu spät wahr
  - Sie haben kein echtes Erfolgserlebnis
  - Es schadet der Entwicklung Ihrer Persönlichkeit
- **Wir vertrauen Ihrer Ehrlichkeit**
  - Benutzen Sie externe Ideen (Bücher/Internet) nur mit Quellenangabe
  - Benutzen Sie keine Lösungen von Kommilitonen
  - Geben Sie keine Lösungen an Kommilitonen weiter
  - Klausurlösungen sollten ausschließlich Ihre eigenen sein
  - Keine “Überwachung”, aber wenn es dennoch auffliegt . . .
- **Mehr zur Arbeitsethik auf unseren Webseiten**

Theoretische Informatik I

Einheit 1

Mathematische Beweisführung

1. Deduktive Beweise
2. Widerlegungsbeweise
3. Induktion
4. Methodik des Problemlösens

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
    - Pentium Bug (1994), Ariane 5 (1996), Mars Polar Lander (1999), ...
- **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

  $\mapsto$ Entwickler müssen die Korrektheit ihrer Programme beweisen
- **Informatiker müssen Beweise verstehen / führen können**

Wie führt man stichhaltige Beweise?

Methodik der Beweisführung

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

Methodik der Beweisführung

- **Ziel: Zeige, daß eine Behauptung B aus Annahmen A folgt**
- **Einfachste Methode: Deduktiver Beweis**
  - Aneinanderkettung von Argumenten / Aussagen $A_1, A_2, ..., A_n = B$
  - Zwischenaussagen A_i müssen schlüssig aus dem Vorhergehenden folgen
  - Verwendet werden dürfen nur Annahmen aus A, Definitionen, bewiesene Aussagen, mathematische Grundgesetze und logische Schlußfolgerungen

Methodik der Beweisführung

- **Ziel: Zeige, daß eine Behauptung B aus Annahmen A folgt**
- **Einfachste Methode: Deduktiver Beweis**
  - Aneinanderkettung von Argumenten / Aussagen $A_1, A_2, ..., A_n = B$
  - Zwischenaussagen A_i müssen schlüssig aus dem Vorhergehenden folgen
  - Verwendet werden dürfen nur Annahmen aus A, Definitionen, bewiesene Aussagen, mathematische Grundgesetze und logische Schlußfolgerungen
- **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 meist 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

Ausführlicher schematischer Beweis

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

- **Definitionen**

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

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

Ausführlicher schematischer Beweis

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

- **Definitionen**

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

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

- **Beweis**

	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)

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 $A \Rightarrow B$

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

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 nicht endlich*

Beweis

	Aussage	Begründung
1.	S endlich	Gegeben
2.	T Komplement von S	Gegeben
3.	U unendlich	Gegeben
4.	T endlich	Annahme
5.	U endlich	(1), (4) mit Satz auf Folie 3
6.	Widerspruch	(3), (5)
7.	T nicht endlich	Annahme (4) muss falsch sein

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 nicht endlich*

  Beweis

	Aussage	Begründung
1.	S endlich	Gegeben
2.	T Komplement von S	Gegeben
3.	U unendlich	Gegeben
4.	T endlich	Annahme
5.	U endlich	(1), (4) mit Satz auf Folie 3
6.	Widerspruch	(3), (5)
7.	T nicht endlich	Annahme (4) muss falsch sein

- **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

Diagonalbeweise

Spezielle Form von Widerlegungsbeweisen

Konstruktion von Gegenbeispielen für Aussagen über unendliche Objekte

- **Beispiel: 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)

	0	1	2	3	4	...
p_0	$\times$	$\times$	$\times$	$\perp$	$\times$	...
p_1	$\perp$	$\perp$	$\times$	$\times$	$\times$	...
p_2	$\times$	$\times$	$\perp$	$\times$	$\times$	...
p_3	$\perp$	$\times$	$\perp$	$\times$	$\perp$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

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

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:

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

	0	1	2	3	4	...
p_0	×	×	×	⊥	×	...
p_1	⊥	⊥	×	×	×	...
p_2	×	×	⊥	×	×	...
p_3	⊥	×	⊥	×	⊥	...
⋮	⋮	⋮	⋮	⋮	⋮	...

× $\hat{=}$ Terminierung, ⊥ $\hat{=}$ hält nicht

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:

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

	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	$\bot$	$\times$	$\bot$	$\bot$	$\bot$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

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

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	$\bot$	$\times$	$\bot$	$\bot$	$\bot$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

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

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:**

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

- **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	$\bot$	$\times$	$\bot$	$\bot$	$\bot$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

$\times \mathrel{\hat{=}}$ 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 **???**

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	$\bot$	$\times$	$\bot$	$\times$	$\bot$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

$\times \mathrel{\hat{=}}$ 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**
  - Gilt B für i und B für $n{+}1$, wenn B für n gilt, dann gilt B für alle $n{\geq}i$

Beweise eine Behauptung B für alle natürlichen Zahlen

- **Standardinduktion**
  - Gilt B für i und B für $n{+}1$, wenn B für n gilt, dann gilt B für alle $n{\geq}i$
  - Beispiel: *Wenn* $x{\geq}4$, *dann* $2^x{\geq}x^2$

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

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

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

Induktive Beweise

Beweise eine Behauptung B für alle natürlichen Zahlen

- **Standardinduktion**
  - Gilt B für i und B für $n{+}1$, wenn B für n gilt, dann gilt B für alle $n{\geq}i$
  - Beispiel: *Wenn* $x{\geq}4$, *dann* $2^x{\geq}x^2$

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

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

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

- **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

Strukturelle Induktion

Zeige B für alle Elemente eines rekursiven Datentyps

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

Strukturelle Induktion

Zeige B für alle Elemente eines rekursiven Datentyps

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

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

Induktionsanfang L ist leer: Die Summe und die Länge von L sind 0

Induktionsschritt: Es gelte $sum(L) \geq |L|$

Betrachte die Liste $L \circ x$, die durch Anhängen von x and L entsteht

Dann gilt $sum(L \circ x) = sum(L) + x \geq sum(L) + 1 \geq |L| + 1 = |L \circ x|$

Strukturelle Induktion

Zeige B für alle Elemente eines rekursiven Datentyps

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

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

Induktionsanfang L ist leer: Die Summe und die Länge von L sind 0

Induktionsschritt: Es gelte $sum(L) \geq |L|$

Betrachte die Liste $L \circ x$, die durch Anhängen von x and L entsteht

Dann gilt $sum(L \circ x) = sum(L) + x \geq sum(L) + 1 \geq |L| + 1 = |L \circ x|$

Häufig eingesetzt für Analyse von

· **Listen- und Baumstrukturen** (Suchen, Sortieren, ...)

· **Syntaktische Strukturen** (Formeln, Programmiersprachen, ...)

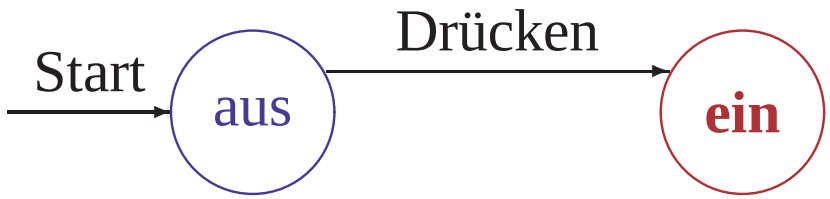
⋮

Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan

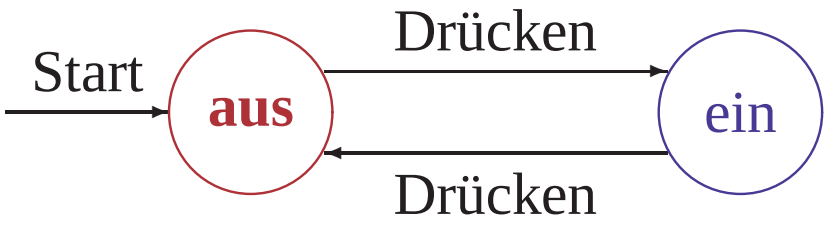
Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



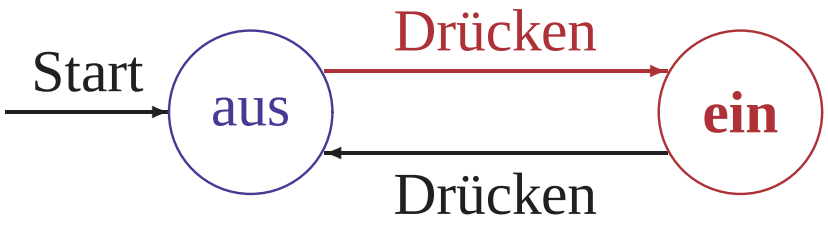
Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



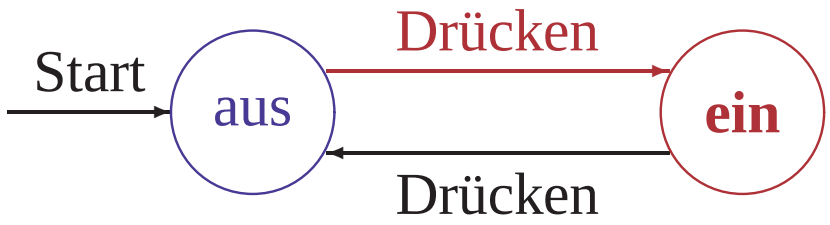
Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



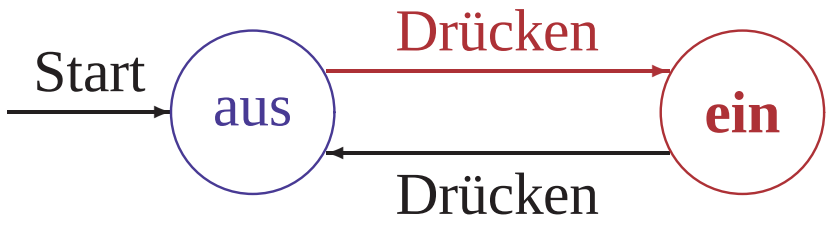
Zeige: Automat ist ein Wechselschalter

$S_1(n)$: Für gerade n ist der Automat nach n-fachem Drücken ausgeschaltet

$S_2(n)$: Für ungerade n ist der Automat nach n-fachem Drücken eingeschaltet

Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



Zeige: Automat ist ein Wechselschalter

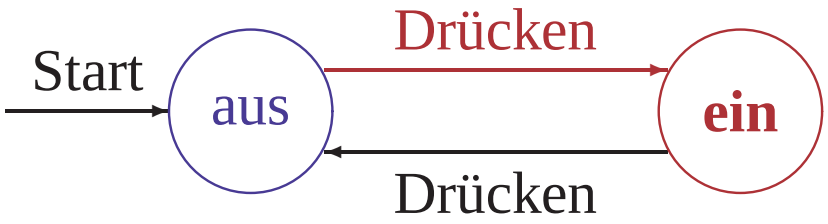
$S_1(n)$: Für gerade n ist der Automat nach n-fachem Drücken ausgeschaltet

$S_2(n)$: Für ungerade n ist der Automat nach n-fachem Drücken eingeschaltet

Induktionsanfang $n{=}0$: n ist gerade, also nicht ungerade; somit gilt $S_2(0)$
der Automat ist ausgeschaltet, also gilt $S_1(0)$

Simultane (gegenseitige) Induktion

Zeige mehrere zusammengehörige Aussagen simultan



Zeige: Automat ist ein Wechselschalter

$S_1(n)$: Für gerade n ist der Automat nach n-fachem Drücken ausgeschaltet

$S_2(n)$: Für ungerade n ist der Automat nach n-fachem Drücken eingeschaltet

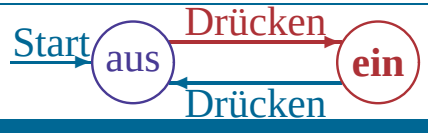
Induktionsanfang $n{=}0$: n ist gerade, also nicht ungerade; somit gilt $S_2(0)$
der Automat ist ausgeschaltet, also gilt $S_1(0)$

Induktionsschritt: Es gelte $S_1(n)$ und $S_2(n)$. Betrachte $n{+}1$

– Falls $n{+}1$ ungerade, dann gilt $S_1(n{+}1)$ und n ist gerade.
Wegen $S_1(n)$ war der Automat aus und wechselt auf ‘ein’. Es gilt $S_2(n{+}1)$

– Falls $n{+}1$ gerade, dann gilt $S_2(n{+}1)$ und n ist ungerade.
Wegen $S_2(n)$ war der Automat ein und wechselt auf ‘aus’. Es gilt $S_1(n{+}1)$

Simultane Induktion ausformuliert



Um zu zeigen, das der Automat ein Wechselschalter ist, zeigen wir durch Induktion, daß für alle $n \in \mathbb{N}$ die folgenden beiden Aussagen gelten

$S_1(n)$**:** n gerade $\Leftrightarrow$ Automat ist nach n-fachem Drücken im Zustand “aus”

$S_2(n)$**:** n ungerade $\Leftrightarrow$ Automat ist nach n-fachem Drücken im Zustand “ein”

Induktionsanfang n=0:

$S_1(n)$**:** 0 ist gerade und der Automat ist zu Beginn im Zustand “aus”, also gilt Aussage $S_1(n)$.

$S_2(n)$**:** 0 ist nicht ungerade und der Automat ist zu Beginn nicht im Zustand “ein”, also gilt die Äquivalenz $S_2(n)$, da jeweils die rechte und linke Seite falsch ist.

Induktionsannnahme: die Aussagen $S_1(m)$ und $S_2(m)$ seien für ein beliebiges $m \in \mathbb{N}$ gezeigt.

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

$S_1(n)$**:** Es ist $n = m{+}1$ gerade

$\Leftrightarrow$ m ist ungerade

$\Leftrightarrow$ Automat ist nach m-fachem Drücken im Zustand “ein” (Induktionsannahme $S_2(m)$)

$\Leftrightarrow$ Automat ist nach n-fachem Drücken im Zustand “aus”

$S_2(n)$**:** Es ist $n = m{+}1$ ungerade

$\Leftrightarrow$ m ist gerade

$\Leftrightarrow$ Automat ist nach m-fachem Drücken im Zustand “aus” (Induktionsannahme $S_1(m)$)

$\Leftrightarrow$ Automat ist nach n-fachem Drücken im Zustand “ein”

Aufgrund des Induktionsprinzips gilt $S_1(n)$ und $S_2(n)$ für alle $n \in \mathbb{N}$

Wie genau/formal muss ein Beweis sein?

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

- Genau genug, um Details rekonstruieren zu können

  Knapp genug, um übersichtlich und merkbar zu sein

  Also nicht notwendig formal oder mit allen Details
- 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

Allgemeine 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

Mathematisches Vokabular I: Wörter und Sprachen

– **Alphabet Σ**: endliche Menge von Symbolen,
  z.B. $\Sigma = \{0, 1\}$, $\Sigma = \{0, .., 9\}$, $\Sigma = \{A, .., Z, a, .., z, \ , ?, !, ..\}$
– **Wörter**: endliche Folge w von Symbolen eines Alphabets
  Auch Zeichenreihen oder Strings genannt
– ϵ: Leeres Wort (ohne jedes Symbol)
– $w\ v$: 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
– Σ^k: Menge der Wörter der Länge k mit Symbolen aus Σ
– Σ^*: Menge aller Wörter über Σ
– Σ^+: Menge aller nichtleeren Wörter über Σ
– **Sprache L**: Beliebige Menge von Wörtern über einem Alphabet Σ
  Üblicherweise in abstrakter Mengennotation gegeben
  z.B. $\{w \in \{0,1\}^* \mid |w| \text{ ist gerade}\}$ $\{0^n 1^n \mid n \in \mathbb{N}\}$
– **Problem P**: Menge von Wörtern über einem Alphabet Σ
  Das "Problem" ist, Zugehörigkeit zur Menge P zu testen

Mathematisches Vokabular II: Funktionen

- **Funktion $f : S \to S'$**: Abbildung zwischen den Grundmengen S und S' nicht unbedingt auf allen Elementen von S definiert
- **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 injektiv!)
- **Urbild $f^{-1}(L)$**: Die Menge $\{x \in S \mid f(x) \in L\}$
- **Charakteristische Funktion χ_L von $L \subseteq S$**:
$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L, \\ 0 & \text{sonst} \end{cases}$$
- **Partiell-charakteristische Funktion ψ_L**:
$$\psi_L(w) = \begin{cases} 1 & \text{falls } w \in L, \\ \bot & \text{sonst} \end{cases}$$

Mehr Vokabular wird bei Bedarf vorgestellt