

# Automatisierte Logik und Programmierung

Wintersemester 2016/17



**Christoph Kreitz**

Theoretische Informatik, Raum 1.18, Telephon 3060

[kreitz@cs.uni-potsdam.de](mailto:kreitz@cs.uni-potsdam.de)

<http://cs.uni-potsdam.de//alup1-ws1617>



1. Automatisches Schließen – wozu?
2. Was ist automatisierte Logik?
3. Anwendungen und Erfolge
4. Organisation der Veranstaltung

# AUTOMATISCHES SCHLIESSEN – WOZU?

- **Es gibt zu viele Fehler in wissenschaftlichen Arbeiten**
  - Moderne Forschungsgebiete sind extrem komplex  
Genaue Analysen und der Beweis ihrer Korrektheit sind schwierig
  - **Menschen machen Fehler** bei der Ausarbeitung von Details
  - Fehler werden auch beim Reviewprozeß von Publikationen übersehen  
*ca. 40–50% aller veröffentlichten Resultate sind falsch*
- **Besonders schlimm ist es bei der Erstellung von Software**
  - Software und ihre Anforderungen sind meist unüberschaubar
  - **Auch sorgfältig getestete Programme** enthalten größere Fehler  
*ca 3-5 Fehler auf 1000 LOC “Hochqualitätscode” bei Kosten von \$200/LOC*
- **Rein informales Vorgehen hat sich nicht bewährt**
  - Wir konzentieren uns auf Ideen und machen Fehler bei der Ausführung
  - Wir verwenden **implizite Annahmen / Analogien**, die nicht immer stimmen
  - Wir **vertrauen** “Autoritäten” ohne nachzuprüfen

# MATHEMATIK-BEISPIEL: DAS BRIEFMARKEN PROBLEM



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

**Idee:**  $8c =$  $, 9c =$  $, 10c =$  $, 11c =$  $, 12c =$  $, \dots$

## • Einfacher Induktionsbeweis

- Zeige: für alle  $n \geq 8$  gibt es  $i, j \in \mathbb{N}$  mit  $n = i \cdot 3 + j \cdot 5$
- Basisfälle 8, 9, 10 sind lösbar wie oben illustriert.
- Lösung für größere  $n$  wird aus der für  $n-3$  mit weiteren 3c erzeugt

## • Gibt es andere Paare mit derselben Eigenschaft?

- Offensichtlich 1c und jede andere Zahl, 2c und jede ungerade Zahl
- **Kann man beweisen, daß dies alle Möglichkeiten sind?**

**Satz:** Ist jedes  $n \geq a+b$  darstellbar als  $n = i \cdot a + j \cdot b$  ( $i, j \in \mathbb{N}, 1 < a < b \in \mathbb{N}$ )  
dann ist  $a=2$  und  $b$  ungerade oder  $a=3$  und  $b=5$

**Beweis:**

$$\exists i, j. a+b+1 = i \cdot a + j \cdot b \quad \mapsto \quad a \mid (b+1) \text{ oder } b=a+1 \quad (1)$$

$$\exists i, j. a+b+2 = i \cdot a + j \cdot b \quad \mapsto \quad a=2 \text{ oder } a \mid (b+2) \text{ oder } b=a+2 \quad (2)$$

Falls  $a=2$ , dann ist  $b$  ungerade wegen (1)

Falls  $a > 2$ , dann ist  $b > 3$  und (1) liefert zwei Fälle

$a \mid (b+1)$ : wegen  $a > 2$  kann  $a$  nicht  $b+2$  teilen und wegen (2) gilt  $b=a+2$

$$\exists i, j. a+b+3 = i \cdot a + j \cdot b \quad \mapsto \quad a=3 \text{ oder } a \mid (b+3) \text{ oder } b=a+3 \quad (3)$$

–  $b=a+3$  ist unmöglich, da  $b=a+2$ .

–  $a \mid (b+3)$  ist unmöglich, da  $a \mid (b+1)$  und  $a > 2$ .

Also gilt  $a = 3$  und  $b = 5$



$b=a+1$ : wegen (2) folgt wie oben  $a \mid (a+3)$  oder  $a+1 = a+2$ .

~~Beides ist unmöglich.~~ **Möglich für  $a=3$  und  $b=4$**

**Formales Vorgehen hilft, solche Fehler zu vermeiden**

# FEHLER IN SOFTWARE SIND NOCH PROBLEMATISCHER

- **Software ist integraler Bestandteil unseres Lebens**
  - Steuerungsmodule in Alltagsprodukten, e-Commerce, Telephonnetze, Smartphones, Automobilkonstruktion, Luftfahrtkontrolle, ...
- **Softwarefehler sind lästig**
  - Reboot, Datenverlust, Funktionsunfähigkeit, Viren, Netzwerkausfälle...
- **Softwarefehler sind teuer**
  - 1994: **Pentium I Prozessor** liefert falsche Resultate bei Division
  - 1996: **ESA Ariane 501** explodiert wegen Überlauf der 16bit Arithmetik
  - 1999: **Mars Polar Lander & Climate Orbiter** stürzen ab (Einheitenfehler)
  - 2016: **Röntgensatellit Hitomi** durch extreme Rotation zerstört
  - 2016: Comdirekt Bank erlebt **unbefugten Online-Zugriff** auf Fremdkonten
- **Softwarefehler kosten Menschenleben**
  - 1988: **Air France A320** streift Bäume bei Flugshow (Einheitenfehler)
  - 1993: **Lufthansa A320** verweigert Umkehrschub bei Landung im Regen
  - 1995: **Boeing 757** prallt auf Berge auf (Fehler in Navigationsdaten)
- **Softwarefehler erzeugen gefährliche Sicherheitslöcher**
  - **Heartbleed Fehler** in SSL (2014), **CAN-Bus Attacken** auf Autos (2015) ...

# ES IST ETWAS FALSCH IN DER SOFTWAREPRODUKTION

- **Softwareproduktion ist heute immer noch ineffizient**

- Entwurf und Implementierung fokussiert auf **Modellierungs- und Programmiersprachen** anstatt auf Eigenschaften des Problembereichs
- Der Weg von Spezifikation zu Code ist i.w. immer noch “von Hand”
  - hohe **Kosten** für Erstellung und Wartung
  - **suboptimaler Code**, funktioniert selten auf Anhieb
  - keine **Begründung für Korrektheit** ihres Programms
- Probleme führen zu **ad hoc Änderungen** statt Revision des Entwurfs
  - Neue Fehler und ständige Sicherheitsupdates

Wer würde ein Auto kaufen/fahren, was derartig viele Probleme hat?

- **Softwareentwicklung ist mehr als Codierung**

- Logische Verarbeitung von Wissen + kreative Umsetzung von Ideen
- Mehr als Modellierungstools alleine liefern können

**Automatisierte formale Logik kann diesen Prozeß unterstützen**

## BEISPIEL: MAXIMALE SEGMENTSUMME EINER LISTE

Gegeben eine Folge  $a_1, a_2, \dots, a_n \in \mathbb{Z}$  bestimme die Summe  $\sum_{i=p}^q a_i$  eines Segmentes, die maximal bezüglich aller möglicher Segmentsummen ist

2	3	-6	4	5	-3	8	-2	-1	5	-9	2	3	16
---	---	----	---	---	----	---	----	----	---	----	---	---	----

- **Direkte Lösung leicht zu programmieren**

```
let maxseg a ≡
  result := a[1]
  from p = 1 to length(a) do
    from q = i to length(a) do
      sum := 0
      from i = p to q do sum := sum+a[i] end
      if sum > result then result := sum
    end
  end
end
```

- **Algorithmus ist ineffizient ( $\mathcal{O}(n^3)$ )**

- **Wie kann man eine bessere Lösung erzeugen?**

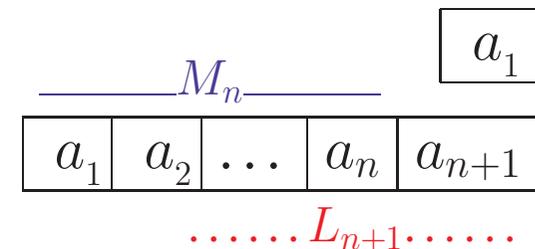
# MAXIMALE SEGMENTSUMME: SYSTEMATISCHE LÖSUNG

Betrachte Eigenschaften von  $M_n \equiv \max\{\sum_{i=p}^q a_i \mid 1 \leq p \leq q \leq n\}$

- **Induktive Analyse liefert**

- $M_1 = a_1$

- $M_{n+1} = \max(M_n, \max\{\sum_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$



- **Definiere  $L_n \equiv \max\{\sum_{i=p}^n a_i \mid 1 \leq p \leq n\}$**

- Dann ist  $L_1 = a_1, \quad L_{n+1} = \max(L_n + a_{n+1}, a_{n+1})$

- **Analyse liefert eleganten, effizienten Algorithmus**

```
let maxseg a ≡
```

```
  Mn := a[1]
```

```
  Ln := a[1]
```

```
  from n = 2 to length(a) do
```

```
    if Ln > 0 then Ln := Ln + a[n] else Ln := a[n]
```

```
    if Ln > Mn then Mn := Ln
```

```
  end
```

- **Logische Analyse führt zu besserer Software**
  - Wissen wird **systematisch** verarbeitet
  - Zusammenhang zwischen Spezifikation und Lösung ist erkennbar
- **Formalisierung logischer Schlüsse eliminiert Fehler**
  - **Simuliere logisches Schließen auf dem Computer**
  - Kreative Steuerung des Entwicklungsprozesses durch Menschen
  - Computer kontrolliert Korrektheit der Herleitung in **logischem Kalkül**
  - Skript der Herleitungsschritte vereinfacht Wartung und Änderungen
  - Effektive **Kooperation** zwischen Mensch und Maschine
- **Automatisierte Logik reduziert Aufwand**
  - Computer führt “triviale” logische Schlüsse von selbst aus
  - Computer **synthetisiert Code** aus formaler Analyse des Problems

## Simulation mathematisch-semantischer Argumentation

- **Anwendung formaler Regeln ohne Nachdenken**

- Umgeht Mehrdeutigkeiten der natürlichen Sprache
- Erlaubt schematische Lösung mathematischer Probleme

- **Kernbestandteile:**

- Formale Sprache (Syntax + Semantik)
- Ableitungssystem (Axiome + Inferenzregeln)

- **Wichtige Eigenschaften**

- Korrekt, vollständig, implementierbar (notwendig)
- Verständliche formale Texte (für Interaktion)
- Konstruktiv, ausdrucksstark (für Programmierung)

Nicht jeder Kalkül hat all diese Eigenschaften gleichzeitig

# WELCHE KALKÜLE BRAUCHEN WIR?

- **Unterstützung für Mathematik und Programmierung**

- Präzises mathematisches Argumentieren (Logik)
- Schließen über **operationales** Programmverhalten (Berechnungskalkül)
- Schließen über **externes** Programmverhalten (Spezifikationskalkül)

- **Bekannte Arten von Kalkülen**

- Mathematische Logik: **Aussagenlogik, Prädikatenlogik, ...**
- Berechnungskalküle: Maschinenmodelle,  **$\lambda$ -Kalkül, ...**
- Spezifikationskalküle (einfachste Form): Typchecking, **Typkalküle, ...**

- **Kann man diese Aspekte in einem Kalkül beschreiben?**

- Kalkül muß Logik mit Berechnung und Typsystem koppeln
- Uniformer Kalkül für Logik, Berechnung, Programmeigenschaften
- Ansätze: **Higher-Order Logic, Konstruktive Typentheorie**

Veranstaltung betrachtet zunächst alle Aspekte separat und dann die einheitliche Theorie

# WELCHE COMPUTERUNTERSTÜTZUNG IST MÖGLICH?

- **Es gibt theoretische Grenzen**

- Arithmetik ist **nicht voll axiomatisierbar**
- Gültigkeit prädikatenlogischer Formeln ist **unentscheidbar**
- Programmterminierung, -korrektheit, -äquivalenz ist **unentscheidbar**
- Computer **brauchen Benutzersteuerung** bei Suche nach Beweisen

- **Interaktive Beweiseditoren**

- Benutzer konstruieren Beweise durch Anwendung von Regeln
- Computer führt Regeln aus und zeigt ungelöste Teilprobleme

- **Automatisierte Beweisprozeduren**

- Entscheidungsprozeduren for entscheidbare Teilprobleme
- Beweissuchverfahren für eingeschränkte Anwendungsbereiche
- **Taktiken**: programmierte Anwendung von Inferenzregeln

- **Integrierte Systeme**

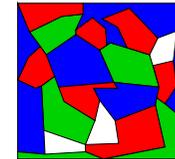
- Interaktive Beweiseditoren mit externen Steuerungsmechanismen

In dieser Veranstaltung wird das **Nuprl** Beweisentwicklungssystem vorgestellt und eingesetzt

# COMPUTERGESTÜTZTES SCHLIESSEN ZEIGT ERFOLGE ...

## 1977: **Vier-Farben Problem**

- Spezialsoftware überprüft tausende kritischer Fälle
- Erneuter Beweis mit generischem Theorembeweiser **Coq** in 2005



## 1993: **Synthese von Scheduling Algorithmen**

- **KIDS** erzeugt korrekte Algorithmen in wenigen Stunden
- Erzeugter Lisp Code 2000 mal schneller als existierende ADA Software

## 1995: **Pentium Bug**

- Model Checking findet Fehler in Hardwaretabellen für Division



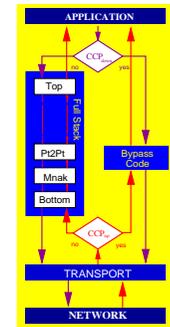
## 1996: **Robbins Hypothese**

- Theorembeweiser **EQP** findet (lesbaren) Beweis in 7 Tagen

The New York Times

## 1998: **Netzwerkverifikation, -optimierung, -entwurf**

- Verifikation findet subtile Fehler in Protokollen und ISO-Standards
- Logische Optimierung steigert Performanz um Faktor 3–10



## 2009–14: **Formal verifizierte Systemkomponenten**

- **CompCert** C Compiler, **sel4** Microkernel, **shadowDB** Datenbank

## 2015: **Keplersche Vermutung**

- **FlySpecK** Projekt formalisiert informalen Beweis in **HOL light**

## 2012–: **Forschungsprogramme HACMS und DeepSpec**

- Erzeugung formal verifizierter Komponenten für eingebettete Systeme

Seit 2012 werden für Hochsicherheitssoftware maschinenprüfbare Korrektheitsbeweise verlangt

- **Entwicklung formaler Logiken** (AluP I)
  - “Klassisch”, konstruktiv, modal, linear temporal, ...
  - Erste oder höhere Stufe , Typentheorien, ...
  - Behandlung von Objekten, Vererbung, Nebenläufigkeit, Echtzeit, ...
- **Automatische Beweisprozeduren** (AluP II, Inferenzmethoden)
  - Matrixmethoden, Induktionsbeweisen, Rewriting, Beweisplaner, ...
  - Entscheidungsprozeduren, kooperierende & verteilte Beweiser, ...
- **Beweisentwicklungssysteme** (AluP II)
  - Interaktive Beweiseditoren, Beweispräsentation, Wissensverwaltung, ...
  - Web-Einbindung, GUI's, Systemschnittstellen, ...
- **Anwendung: Logik in die Software bringen** (AluP II)
  - Verifikation formalen Wissens zu Daten- und Algorithmenstrukturen
  - Strategien für Synthese, Verifikation und Optimierung von Algorithmen
  - Entwicklung und Verifikation sicherheitskritischer Systeme

⋮

- **Zuordnung: theoretische/angewandte Informatik**

- **Veranstaltungen**

- **Vorlesung** Mi/Do 10:15–11:45 (1.02)

Präsentation der zentralen Konzepte / Ideen

Keine Veranstaltung am 12./13.12.2016 und 4.1.2017

- **Übungsaufgaben** gelegentlich

Anregung und Herausforderungen zum Selbsttraining

- **Sprechstunde** Fr 10:30–11:30 (und immer wenn die Türe offen ist)

Fachberatung / Klärung von Schwierigkeiten mit der Thematik

- **Lehrmaterialien**

- Vorlesungsfolien, Handouts, Skript (1995), Fachbücher im Web

- **Empfohlene Vorkenntnisse:**

- Theoretische Informatik II, Logik, (etwas) funktionale Programmierung

- **Erfolgskriterium: Abschlußklausur am 9. Februar 2017**

- Alternativ mündliche Prüfung nach Vereinbarung