Formale Grundlagen der Informatik 3 –

1. Einleitung und Übersicht

Christoph Walther
TU Darmstadt

1 Organisatorisches

- Vorlesungstermin
  - *regulär:* Dienstags, 16h15 – 17h45, S2|02/C205
  - *bei Bedarf (nach Ankündigung):* Freitags, 11h40 – 13h10, S2|02/C205 (für Ergänzungsstoff/Übungen)
- Auf der Web-Seite der Vorlesung sämtliche Informationen zu
  - Terminen,
  - Übungen,
  - Klausuren,
  - Vorlesungsmaterialien,
  - ...
- ***Fachliche*** Unklarheiten und Fragen werden am besten geklärt und beantwortet (mit absteigender Priorität)
  - in der Vorlesung
  - im Forum (URL => s. Web-Seite der Vorlesung)
  - per Email (Adresse => s. Web-Seite der Vorlesung)
  - in den Sprechstunden (Termine => s. Web-Seite der Vorlesung)

2 Motivation und Thema der Vorlesung

- Aufgabe des Informatikers:
  Entwickle korrekte und effiziente algorithmische Lösungen für Probleme.
- Hier Thema: *Korrektheit*
- Ist für jede Ingenieurdisziplin unabdingbares Kriterium:
  - *Beispiel:* Wann ist der Entwurf einer Brücke *korrekt*?
    => Wenn der Entwurf die gegebene *Spezifikation* erfüllt.
  - Was ist Spezifikation einer Brücke?
    => *Forderungen* bzgl. Belastbarkeit, Toleranz für Schwankungen, ...
  - Wie formuliert man das?
    => mit *mathematischen Formeln* (Differentialgleichungen etc.).
  - Wie zeigt man Korrektheit?
    => mit *mathematischen Methoden* (Statik, Mechanik, etc.).
  - Was hilft Aufwand für Spezifikation & Korrektheitsnachweis zu reduzieren?
    => *Rechnerunterstützung*.

- Genauso in der *Informatik*:
  - *Beispiel:* Wann ist der Entwurf eines Sortierverfahrens *korrekt*?
    => Wenn der Entwurf die gegebene *Spezifikation* erfüllt.
  - Was ist Spezifikation eines Sortierverfahrens?
    => Forderungen an das Ergebnis (geordnete Permutation der Eingabe).
  - Wie formuliert man das?
    => mit *mathematischen Formeln* (bzgl. einer formalen Logik).
  - Wie zeigt man Korrektheit?
    => mit *mathematischen Methoden* (formale Beweise führen).
  - Was hilft Aufwand für Spezifikation & Korrektheitsnachweis zu reduzieren?
    => *Rechnerunterstützung*.
- **Frage:** Würden Sie vertrauensvoll eine Brücke betreten, die anstatt mit ingenieurmäßigen Methoden nach Kriterien wie “Viel (Beton) hilft viel”, “Das haben wir schon immer so gemacht”, . . . entworfen wurde ?
- **Frage:** Würden Sie vertrauensvoll ein Programm verwenden, das anstatt mit ingenieurmäßigen Methoden nach Kriterien wie “Viel (Code) hilft viel”, “Das haben wir schon immer so gemacht”, . . . entworfen wurde ?

- **Daher:**
  Formale Methoden spielen immer größer werdende Rolle in der Informatik.
- Unterschied zwischen diversen Ingenieurdisziplinen und der Informatik:
  - Je nach *Anwendungsgebiet* wird eine *geeignete Mathematik* (Statik, Mechanik, Analysis, ...) verwendet.
  - Geeignete Mathematik für die *Informatik*: *Formale Logiken*.
- Bei uns an der **TUD**: *FGdI 1, 2, 3* und weiterführend
  - *Formale Modellierung* (FOC Kanonik), *formale IT-Sicherheit, ...*
    (=> Prof. Dr. H. Mantel)
  - *Korrektheit nebenläufiger Systeme, Modelchecking, ...*
    (=> N. N.)
  - *Korrektheit sequentieller Programme, Automatisches Beweisen, ...*
    (=> Prof. Dr. Chr. Walther)

3 Lernziele *FGdI 3*

(1) Formale Modellierung von Problemlösungen

(2) Formale Spezifikation von Korrektheitseigenschaften

(3) Führen von Korrektheitsbeweisen mit Unterstützung eines Verifikationswerkzeugs

(4) Vertiefung, Anwendung und Erweiterung von Konzepten und Methoden der formalen Logik (=> FGdI 2).

3.1 Zu (1), (2) und (3)

- Formale Modellierung, Spezifikation und Verifikation ist aufwendig.
- Mit Papier & Bleistift können nur sehr kleine Beispiele bearbeitet werden.
- **Daher:** Wir verwenden ein Verifikationswerkzeug (=> *VeriFun* System).[1]
- ***VeriFun***: Werkzeug zur formalen Modellierung, Spezifikation und Verifikation von $\mathcal{L}$-Programmen.

[1] Version 3.2.2

- Was ist hier speziell?
  - Bedienung von *VeriFun*
  - Funktionale Programmier- und Spezifikationssprache $\mathcal{L}$
- **Aber:** *Andere Verifikationswerkzeuge* werden *ähnlich bedient*, stellen *ähnliche Konzepte* zur Verfügung, ...
- **Analogie:** Hat man für die Übungen einer Informatikvorlesung *eine* Programmiersprache kennengelernt, so ist das Erlernen weiterer Programmiersprachen wesentlich einfacher.
- **Kurzum:** Will man *Verifikation praktisch üben*, so muß man sich mit (irgend)einem *konkreten System* vertraut machen.
- **Formale Modellierung, Spezifikation und Verifikation mit *VeriFun*:**
  - Einführung in der Vorlesung.
  - Anwendung in den Übungen & Testaten.

3.2 Zu (4)

- Wir modellieren und spezifizieren in $\mathcal{L}$. Was bedeutet das?
  - Wir müssen die *formale Semantik* von $\mathcal{L}$*-Programmen* definieren.
  - Wir müssen mathematisch präzise definieren, wann *Aussagen* über $\mathcal{L}$-Programme *wahr* sind.
- Wir *beweisen* mit ***VeriFun*** *Aussagen* über $\mathcal{L}$-Programme. Was bedeutet das?
  - Welche formalen Grundlagen und mathematische Konzepte sind hier erforderlich?
  - Was ist ein *Beweis* in ***VeriFun***?
  - Welche *Kalküle* verwenden wir zum *Beweis* von Aussagen?
  - Welche *Eigenschaften* besitzen die *Kalküle*?
  - Ist jede *bewiesene* Aussage *wahr*?
  - Kann jede *wahre* Aussage auch *bewiesen* werden?
  - ...
- Was ist hier speziell?
  - Verwendete Kalküle
  - Einige Datenstrukturen zur syntaktischen Repräsentation von Mengen, Relationen, etc.

- **Aber:** *Andere Verifikationswerkzeuge* verwenden *ähnliche Kalküle, ähnliche Repräsentationskonzepte, . . .*
- **Kurzum:** *Verwendet* man (irgend)ein konkretes *Verifikationswerkzeug*, so muß man sich über *die Bedeutung der Systemergebnisse* im Klaren sein.
- **Formale Grundlagen und Konzepte der Verifikation:**
  - In der Vorlesung.
  - Vertiefung in den Übungen.

4 Inhaltsübersicht

- **Kapitel 2** *Spezifikation und Verifikation*
  - *Fallstudie*: Wir beweisen mit Rechnerunterstützung die Korrektheit eines Sortierverfahrens.
  - Bei welchen Problemen müssen wir dabei dem System “helfen”?
  - Wie gehen wir dafür zielgerichtet vor?
  - Welchen eigenen Aufwand müssen wir bei rechnergestützten Korrektheitsbeweisen treiben?

- **Kapitel 3** *Spezifizieren, Beweisen und Testen*
  - Wie legen wir formal präzise fest, was wir konkret von einem Programm fordern?
  - Mit welchen formalen Methoden können wir zeigen, daß ein Programm unseren Forderungen genügt?
  - Welche Unterstützung können wir dabei von einem Rechner erwarten?
  - Wie können wir unsere Programme mit Rechnerunterstützung testen?

- **Kapitel 4** *Beweise, Kalküle, Herleitungen*
  - Warum überhaupt “formale” Logik?
  - Warum keine formale Logik in Mathematikbüchern?
  - Welche Rolle spielt die formale Logik in der Informatik?
  - Wie kann man mit Konzepten der formalen Logik algorithmische Lösungen beschreiben?

- **Kapitel 5** *Syntax und Semantik von $\mathcal{L}$-Programmen*
  - Wir definieren eine einfache funktionale Programmiersprache.
  - Durch Angabe eine Interpreters legen wir präzise fest, wie Programme unserer Programmiersprache ausgeführt werden.

- **Kapitel 6** *Induktion und Rekursion*
  - Was sind die mathematischen Grundlagen von rekursiven Definitionen?
  - Um Eigenschaften über rekursiv definierte Prozeduren zu beweisen, wird Induktion benötigt – was sind die mathematischen Grundlagen von Induktionsbeweisen?

  - Wie kann man die mathematischen Konzepte für Induktion durch Datenstrukturen beschreiben, so daß Induktionsbeweise maschinell geführt werden können?

- **Kapitel 7** *Strukturelle Induktion und Induktion nach Rekursion von Prozeduren*
  - Wie kann man aus der Definition eines Datentyps ein Induktionsprinzip gewinnen?
  - Wie kann man aus der Definition einer Prozedur ein Induktionsprinzip gewinnen?
  - Warum sind solche Induktionsprinzipien bei Programmkorrektheitsbeweisen nützlich?

- **Kapitel 8** *Terminierung*
  - Was heißt genau “eine Prozedur terminiert”?
  - Wie kann man die Terminierung von Prozeduren beweisen?
  - Wie beweist man Terminierung mit Rechnerunterstützung?

- **Kapitel 9** *Axiome für $\mathcal{L}$-Programme*
  - Wie gewinnt man Axiome aus Datentypdefinitionen, Prozedurdefinitionen und bewiesenen Lemmata?

- **Kapitel 10** *Symbolische Auswertung*
  - Wie werden Induktionsanfänge und -schritte mittels der Axiome von $\mathcal{L}$-Programmen durch “Symbolische Auswertung” maschinell “vereinfacht”?

- **Kapitel 11** *Prädikatenlogik und Symbolische Auswertung*
  - Wiederholung: Syntax und Semantik der Prädikatenlogik 1. Stufe
  - Eigenschaften und Grenzen der Symbolischen Auswertung
  - Wann ist Benutzerunterstützung bei Beweisen erforderlich?

- **Kapitel 12** *Semantik von $\mathcal{L}$-Lemmata*
  - Wann sind die durch Lemmata spezifizierten Programmeigenschaften “wahr”?