Universität Greifswald, Institut für Mathematik und Informatik

Alexander Steen, Juniorprofessur Informatik
E-Mail: alexander.steen@uni-greifswald.de
Tel.: +49 3834 420 **4634**, Web: https://alexandersteen.de

Themenangebote für Bachelorarbeiten

Kontext: Mathematische Logik und automatisches Beweisen

Logiken sind formale Sprachen die es erlauben komplexe Sachverhalte mathematisch präzise auszudrücken. Zu diesem Zweck werden die logischen Formeln in einer feststehenden Sprache (der Syntax) ausgedrückt, und erhalten zudem eine eindeutige formale Bedeutung (ihre Semantik). Dadurch können wir mit logischen Formeln Schlussfolgerungen, z.B. mathematische Beweise, korrekt, nachvollziehbar und überprüfbar formulieren. Ein Beweis in einer Logik ist dann eine Kette von Schlussfolgerungen, von einer Menge von Annahmen ausgehend, zu unserer Vermutung (Beweistheorie). Beispiele für konkrete Logiken sind ...

- Aussagenlogik (*propositional logic*, PL), Beispielformel: $\neg(P \wedge Q)$
- Prädikatenlogik erster Stufe (*first-order logic*, FOL), Beispielformel $\forall x.\, \exists y.\, P(x) \vee Q(x, y)$

In dem Gebiet der *computational logic* (komputationale Logik) wird nun versucht, den Prozess des Beweisens von Aussagen zu automatisieren. Software-Systeme, die so etwas umsetzen, werden automatische Theorembeweiser (*automated theorem proving systems*) genannt [RV01]. Ein solcher Theorembeweiser ist z.B. der von mir entwickelte Beweiser **Leo-III** [SB21].[1] Leo-III ist ein Beweiser für Prädikatenlogik höherer Stufe (*higher-order logic*, HOL), HOL ist eine ausdrucksstarke Erweiterung von FOL und kann komplexe mathematische Sachverhalte kompakt darstellen. Das folgende Problem repräsentiert die Behauptung, dass es keine surjektive Funktion $f : S \to \mathcal{P}(S)$ einer Menge in ihre Potenzmenge geben kann (anders formuliert: Potenzmengen sind strikt größer als ihre zugrunde liegenden Mengen, vgl. Cantors Theorem), in einem maschinenlesbaren Format [Sut17]:[2]

```
thf(surjective_decl, type, surjective: ($i > ($i > $o)) > $o).
thf(surjective_def,  definition,
    surjective = (^ [F:$i > ($i > $o)]: (
                    ! [Y: $i > $o]: ? [X: $i]: ( (F @ X) = Y ) ))).
thf(cantor,          conjecture,
    ~ ( ? [F: $i > ($i > $o)] : ( surjective @ F ) )).
```

In diesem Format werden Existenzquantoren $\exists$ und Universalquantoren $\forall$ als `?` und `!`, Negationen $\neg$, Konjunktionen $\wedge$ und Disjunktionen $\vee$ als `~`, `&` und `|`, dargestellt. In den ersten drei Zeilen wird das Prädikat `surjective` definiert, die letzten beiden Zeilen stellen dann die Behauptung auf, dass es so eine Funktion `F` nicht geben kann.

In der Tat kann dies von **Leo-III** in weniger als einer Sekunde automatisch bewiesen werden:

```
> leo3 surjective.cantor.tptp
% No. of inferences in proof: 15
% SZS status Theorem for surjective.cantor.tptp
```

Der Beweis ist leider recht kryptisch, kann aber theoretisch Schritt-für-Schritt nachvollzogen werden (siehe Appendix A, falls von Interesse).

Logik und automatische Theorembeweiser können nicht nur für das Beweisen von mathematischen Sätzen genutzt werden, sondern sind mittlerweile auch oft genutzte Systeme in anderen Fächern, z.B. der Biochemie, der Künstlichen Intelligenz, der Rechtsinformatik und der theoretischen Philosophie (Metaphysik); das Forschungsgebiet kann man grob als *Wissensrepräsentation und Schließen* (*knowledge representation and reasoning*) [vHLP08] umschreiben (umfasst allerdings noch viele weitere Themen).

Die Vorschläge für Abschlussarbeiten thematisieren sowohl die Weiterentwicklung von Logikformalismen (theoretische Arbeit), als auch die praktische Weiterentwicklung der Beweisersysteme (Programmierung, Evaluation), und die Anwendung von Logikformalismen oder -software auf andere wissenschaftliche Gebiete.

[1] Leo-III ist als open-source Software auf GitHub via `github.com/leoprover/Leo-III` verfügbar.
[2] Dieses Format nennt sich TPTP und ist ein Syntax-Standard für Theorembeweiser.

Themenvorschläge (Stand: Juli 2023)

Diese Liste an potenziellen Themen ist nicht abschließend. Falls sie generelles Interesse an der Thematik Logik und Automatisierung haben, hier aber nicht fündig werden, kontaktieren sie mich gerne z.B. via E-Mail. Wir können dann nach gemeinsamen Interessen und Schnittgebieten suchen.

Themen die eher theoretisch (mathematisch bzw. logisch) sind, sind mit **T** markiert. Themen die eher anwendungsorientiert (Implementierung von Software, Evaluation, Anwendung) sind, sind mit **A** markiert. Arbeiten mit **T+A** beinhalten beide Aspekte. Bei Mehrfachangaben sind verschiedene Schwerpunkte, ganz nach ihrem Interesse, möglich.

(i) **Formalisierung und Analyse von mathematischen Strukturen** **T**/**T+A**
Mithilfe von Theorembeweisern, insbesondere sog. Beweisassistenten, können mathematische Strukturen (Graphen, Zahlen, ...) in Logik kodiert und dann semi-automatisch analysiert werden. So können bekannte Aussagen/Sätze verifiziert werden, oder aber auch Fehler in wissenschaftlichen Papieren oder Büchern entdeckt werden.

Ziel der Bachelor ist es, ein begrenztes mathematisches Themenfeld entsprechend zu kodieren und einige bekannte Eigenschaften nachzuweisen.

Voraussetzungen: Grundlegende Vorkenntnisse in (diskreter) Mathematik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich.

(ii) **Untersuchung von biologischen Modellen mit formalen (logischen) Methoden** **T**/**T+A**
Logikformalismen können genutzt werden um (qualitative) biologische Modelle abzubilden, z.B. Metabolismusnetzwerke oder andere biochemische Prozesse.

Ziel der Bachelor ist es, ein näher zu bestimmendes biologisches Modell geeignet in Logik zu kodieren und durch Computersoftware zu analysieren

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik und Biologie sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich.

(iii) **Formalisierung und Sammlung von Modallogik-Problemen aus der Literatur** **T**/**T+A**
Die TPTP (*Thousands of Problems for Theorem Provers*, `https://tptp.org/`) ist eine Datenbank von Test- und Benchmarkproblemen für Theorembeweiser, z.B. zum quantitativen Studium der Effektivität von Theorembeweisersystemen oder zum Auffinden von Fehlern. TPTP stellt auch einen Eingabe-Standard für diese Software bereit.

Ziel der Bachelorarbeit ist es, den neuen TPTP-Eingabe-Standard für Modallogik zu nutzen um Modallogik-Probleme aus der Literatur darin zu kodieren, und vorhandene Beweiser zu nutzen um die modellierten Probleme lösen zu lassen.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik und Logik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich.

(iv) **Beweisverfikation von automatisch generierten Beweisen** **T+A**
Theorembeweiser geben (kryptische) Beweise aus, siehe z.B. in Appendix A. Prinzipiell sind diese Beweise Schritt-Für-Schritt-Protokolle des formalen logischen Beweises. Natürlich können aber auch Theorembeweiser, wie jede Software, fehlerhaft sein.

Ziel der Bachelorarbeit ist es, Beweisausgaben von Leo-III durch ein unabhängiges Tool (Dedukti) verifizieren zu lassen. Hierzu müssen die Leo-III-Beweise geeignet übersetzt und die Beweisregeln von Leo-III in Dedukti kodiert werden.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik/Logik sind vorteilhaft; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an der praktischen Arbeit mit Logik-Software sind unabdinglich.

(v) **Formalisierung von Gruppenentscheidungsmodellen aus der theoretische Philosophie)** **T+A**

Formale Logik wird nicht nur zur Darstellung von mathematischem Wissen genutzt, sondern auch in vielen anderen Fächern. Eines davon ist die theoretische Philosophie. Exemplarisch soll in der Bachelorarbeit ein Logikformalismus für Gruppenentscheidungsprozesse durch eine geeignete Übersetzung nach klassischer Logik automatisiert werden.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik und Logik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich.

(vi) **Erstellung detaillierter Beweise in automatischen Theorembeweisern** T+A

Die formalen Beweise von Leo-III enthalten manchmal "Lücken", nämlich genau dann, wenn eine externe Hilfssoftware von Leo-III genutzt wurde um den Beweis zu finden. Ziel der Bachelorarbeit ist es, diese Lücken zu schließen und einen vollständigen Beweis erstellen zu lassen.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich. Programmierkenntnisse (z.B. in Python, Scala, o.ä.) werden vorausgesetzt.

(vii) **Arithmetik in Leo-III** T+A

Leo-III ist ein universeller Theorembeweiser für höherstufige Prädikatenlogik. Wie viele Theorembeweiser unterstützt Leo-III allerdings kaum bis gar nicht Schlussfolgerungen die arithmetische Aussagen beinhalten, z.B. die Aussage *"Es existiert eine Zahl X die gleich 4 ist"* – $(\exists X.\, X = 4)$ – kann nicht bewiesen werden; obwohl dies natürlich trivialerweise eine gültige Aussage ist (wähle $X = 4$).

Ziel dieser Bachelorarbeit ist es, grundlegende arithmetische Inferenzregeln in den Beweiskalkül von Leo-III zu integrieren. Dazu muss die Code-Basis (in Scala programmiert) gesichtet, und an geeigneten Stellen passende Routinen entworfen, implementiert und eingeführt werden. Kurze Beweise der Korrektheit der vorgeschlagenen Prozeduren sind notwendig.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich. Programmierkenntnisse (z.B. in Python, Scala, o.ä.) werden vorausgesetzt.

(viii) **Interaktive Visualisierung von Kripke-Strukturen** A

Kripke-Strukturen geben eine konkrete Interpretation von Formelsymbolen an (z.B. Belegung in einer Wahrheitstabelle), nach der eine logische Formel ausgewertet wird. Ist eine Formel in allen Strukturen gültig, so ist die Formel eine Tautologie. Gibt es mindestens eine Struktur, welches eine Formel ungültig macht, so heißt die Formel widerlegbar und die entsprechende Struktur wird Gegenmodell genannt.

Gegenmodelle spielen die Rolle von Gegenbeispielen und geben somit Rückmeldung über spezifische Situationen in denen eine bestimmte Aussage nicht zutrifft. Diese Information hilft unwahre Annahmen zu reflektieren, Spezialfälle zu identifizieren, oder auch Fehler zu finden und zu beheben.

In der sog. Modallogik [BvBW07] können Modelle und Gegenmodelle als gerichtete Graphen dargestellt werden. Modallogiken erlauben es Informationen außerhalb des üblichen mathematischen Spektrums auszudrücken, z.B., Information über Glauben, Wissen, Notwendigkeit, Zeit, etc.

In dieser Bachelorarbeit sollen Ausgaben von Modell-Generierern (Software-Systeme die automatisch nach (Gegen-)Modellen suchen, z.B. Nitpick [BN10]) eingelesen und geeignet visualisiert werden. Dabei sollen insbesondere Visualisierungen von Modellen der Modallogik erster Stufe addressiert werden. Idealerweise sollte die Visualisierung eine interaktive Benutzeroberfläche haben.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik sind nötig; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich. Programmierkenntnisse (z.B. in Python, Scala, o.ä.) werden vorausgesetzt.

(ix) **Weiterentwicklung eines browserbasierten Annotationseditors für juristische Anwendungen** A

Durch die Browseranwendung NAINEXT sollen juristische Texte durch Userinteraktion annotiert, und so in einen Logik-ähnlichen Formalismus übersetzt werden. So sollen juristische Schlussfolgerungen durch Theorembeweiser semi-automatisiert werden.

In der Bachelorarbeit soll das Tool verbessert werden, und neue Fähigkeiten entwickelt werden.

Voraussetzungen: Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich. Programmierkenntnisse (primär in JavaScript) sind erforderlich.

(x) **Entwicklung didaktischer Logik-Tools für die Hochschullehre** A
Logikgrundlagen können oftmals sehr theoretisch und wenig anschaulich sein. Ziel der Bachelorarbeit ist es, einige grundlegende Verfahren aus der Logik-Automatisierung so zu implementieren dass sie als didaktische Hilfsmittel verwendet werden können.

Voraussetzungen: Grundlegende Vorkenntnisse in diskreter Mathematik/Logik sind vorteilhaft; Kenntnisse in Theorembeweisen sind jedoch nicht erforderlich, Interesse an logischer Modellierung sind unabdinglich.

Literatur

[BN10] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.

[BvBW07] Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*. North-Holland, 2007.

[RV01] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[SB21] Alexander Steen and Christoph Benzmüller. Extensional Higher-Order Paramodulation in Leo-III. *J. Autom. Reason.*, 65(6):775–807, 2021.

[Sut17] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[vHLP08] Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, editors. *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. Elsevier, 2008.

A Leo-III Beweis von Cantors Theorem

```
% SZS output start Refutation for surjective.cantor.tptp
thf(surjective_type, type, surjective: (($i > ($i > $o)) > $o)).
thf(surjective_def, definition, (surjective = (^ [A:($i > ($i > $o))]: ! [B:($i > $o)]:
    ? [C:$i]: ((A @ C) = B)))).
thf(sk1_type, type, sk1: ($i > ($i > $o))).
thf(sk2_type, type, sk2: (($i > $o) > $i)).
thf(1,conjecture,((~ (? [A:($i > ($i > $o))]: (surjective @ A)))),file('surjective.
    cantor.tptp',cantor)).
thf(2,negated_conjecture,((~ (~ (? [A:($i > ($i > $o))]: (surjective @ A))))),inference(
    neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = (B))))
    )),
  inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,((? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = (B)))),
  inference(polarity_switch,[status(thm)],[3])).
thf(5,plain,(! [A:($i > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(cnf,[status(esa)
    ],[4])).
thf(6,plain,(! [A:($i > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(lifteq,[status(thm
    )],[5])).
thf(7,plain,(! [B:$i,A:($i > $o)] : (((sk1 @ (sk2 @ (A)) @ B) = (A @ B)))),inference(
    func_ext,[status(esa)],[6])).
thf(9,plain,(! [B:$i,A:($i > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | (~ (A @ B)))),inference(
    bool_ext,[status(thm)],[7])).
thf(250,plain,(! [B:$i,A:($i > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | ((A @ B) != (~ (sk1 @ (
    sk2 @ (A)) @ B))) | ~ ($true))),
  inference(eqfactor_ordered,[status(thm)],[9])).
thf(270,plain,((sk1 @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A
     @ A))))),
  inference(pre_uni,[status(thm)],[250:[bind(A, $thf(^ [C:$i]: ~ (sk1 @ C @ C))),
    bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(8,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | (A @ B))),inference(
    bool_ext,[status(thm)],[7])).
thf(18,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | ((A @ B) != (~ (sk1
    @ (sk2 @ (A)) @ B))) | ~ ($true))),
  inference(eqfactor_ordered,[status(thm)],[8])).
thf(32,plain,((~ (sk1 @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:$i]: ~ (sk1 @
     A @ A))))),
  inference(pre_uni,[status(thm)],[18:[bind(A, $thf(^ [C:$i]: ~ (sk1 @ C @ C))),
    bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(372,plain,($false),inference(rewrite,[status(thm)],[270,32])).
thf(373,plain,($false),inference(simp,[status(thm)],[372])).
% SZS output end Refutation for surjective.cantor.tptp
```