Mathematisches Argumentieren und Beweisen mit dem Theorembeweiser Coq

Sebastian Böhne*, Maria Knobelsdorf^, Christoph Kreitz*

*Fachbereich Informatik, Universität Potsdam
August-Bebel-Str. 89, 14482 Potsdam

^Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg

Zusammenfassung: Informatik-Studierende haben in der Mehrzahl Schwierigkeiten, einen Einstieg in die Theoretische Informatik zu finden und die Leistungsanforderungen in den Endklausuren der zugehörigen Lehrveranstaltungen zu erfüllen. Wir argumentieren, dass dieser Symptomatik mangelnde Kompetenzen im Umgang mit abstrakten und stark formalisierten Themeninhalten zugrunde liegen und schlagen vor, einen Beweisassistenten als interaktives Lernwerkzeug in der Eingangslehre der Theoretischen Informatik zu nutzen, um entsprechende Kompetenzen zu stärken.

1 Ausgangslage

In der Studieneingangsphase der Informatik-Bachelorstudiengänge in Deutschland stellen die mathematischen und theoretischen Fachanforderungen in den korrespondierenden Einführungsveranstaltungen nach wie vor eine große Herausforderung dar. Problematisch für die Studierenden sind dabei weniger konkrete Wissenslücken denn generelle Schwierigkeiten im Umgang mit mathematischen Formalismen, dem damit verbundenen Abstraktionsgrad, sowie der Übertragung mathematischer Methoden und Problemlösestrategien auf die Informatik (Knobelsdorf und Frede, 2016). Hier kann immer wieder durch die Lehrenden beobachtet werden, dass einige Studierende bereits große Schwierigkeiten haben, grundlegende mathematische Formalismen überhaupt zu lesen und ihren Inhalt zu

verstehen. Diejenigen Studierenden, die diese Hürde überwinden, sind dann nicht automatisch auch in der Lage, eine Lösung anhand der mathematischen Formalismen zu entwickeln. Die entwickelten Beweisideen für ein konkretes Problem formal korrekt aufzuschreiben, stellt dann noch einen weiteren notwendigen Schritt dar, den viele Studierende erst noch erlernen müssen. Insgesamt wird deutlich, dass Studierende einen großen Entwicklungsbedarf hinsichtlich ihrer formal-mathematischen Fachkompetenzen haben. Daher ist die Entwicklung entsprechender didaktischer Maßnahmen und Werkzeuge, die den Studierenden helfen, zugehörige Kompetenzen zu entwickeln, so außerordentlich wichtig.

In diesem Artikel stellen wir einen fachdidaktischen Ansatz für die Eingangslehre der Theoretischen Informatik dar, der mit einem Beweis-assistenten arbeitet und den oben genannten Bedarf aufgreift. Der Ansatz verfolgt das Ziel, Grundlagen im mathematischen Argumentieren und Beweisen zu vermitteln und die formal-präzise Darstellung und Anwendung theoretischer Konzepte einzuüben. Dadurch sollen die Studierenden ein Verständnis für den Zusammenhang zwischen Mathematik und Informatik entwickeln sowie befähigt werden, Fortschritte im strukturierten und abstrakten Denken zu machen. In Abschnitt 2 werden wir unseren Ansatz theoretisch begründen und als konkretes Werkzeug den Beweisassistenten Coq vorstellen. Das Vermittlungskonzept mit Coq sowie konkrete Inhalte und Aufgaben werden in Abschnitt 3 vorgestellt. Der Artikel schließt mit einem Fazit und Ausblick auf weitere Schritte in Abschnitt 4.

2 Beweisen lernen mit einem Beweisassistenten

Computergestützte Systeme zur Durchführung mathematischer Beweise werden seit den 1960ern entwickelt. Man unterscheidet zwischen Systemen, die Beweise erzeugen, sowie solchen, die Menschen beim Erzeugen von Beweisen unterstützen. Letztere werden auch Beweisassistenten oder interaktive Theorembeweiser genannt. Beweisassistenten wurden bereits des öfteren in weiterführenden Veranstaltungen in der Theoretischen Informatik und Mathematik eingesetzt (Henzand Hobor 2011.), (Delahaye, 2005), (Andrew et al., 2004). In der Eingangslehre der Theoretischen Informatik hingegen sind Beweisassistenten bisher nur sehr vereinzelt eingesetzt worden (Sakowicz and Chrząszcz, 2007). Insgesamt wird dabei überwiegend eine sehr positive Resonanz bei den Studierenden festgestellt, so dass die Entwicklung und Erprobung eines didaktischen Konzepts für die Nutzung von

Beweisassistenten auch in der Studieneingangsphase als vielversprechend erscheint. In weiteren Verlauf dieses Abschnitts werden wir Beweisassistenten genauer vorstellen und theoretisch herleiten, warum ihr Einsatz in der Eingangslehre sinnvoll ist.

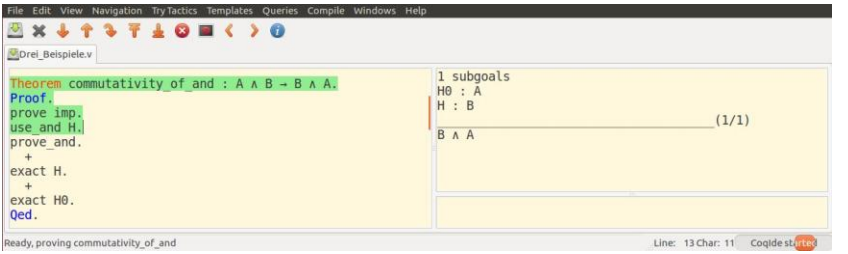
Es existieren eine ganze Reihe von Beweisassistenten. Einer von ihnen, Coq (Bertot und Casteran, 2004), ist Gegenstand dieses Artikels. Andere Beweisassistenten sind beispielsweise Isabelle (Nipkow, Paulson und Wenzel, 2002) oder NuPRL (Constable et al., 1986), (Allen et al., 2006), bei weitläufigerer Auslegung des Begriffs auch Agda (Bove, Dybjer und Norell, 2009) oder Idris (Brady, 2013). Einige der Beweisassistenten, darunter auch Coq, fußen auf der Typentheorie. Insbesondere werden Theoreme als Typen und Beweise für Theoreme als Elemente des jeweiligen Typs aufgefasst. Fast alle Theoreme beinhalten Implikationen und/oder Allquantifikationen. Zugrundeliegende Beweise solcher Theoreme sind dann Funktionen (im getypten λ-Kalkül), so dass solche Beweisassistenten eng verzahnt mit funktionaler Programmierung sind.

Bei Beweisassistenten im engeren Sinne des Begriffes kann der Nutzer Schritt für Schritt Wissen aus den Voraussetzungen generieren und das bisherige Ziel durch ein leichter zu zeigendes, aber hinreichendes Ziel ersetzen. Konkret werden Beweise durch eine Auflistung von sogenannten Taktiken innerhalb einer dem Programmieren ähnlichen Entwicklungsumgebung gewonnen, wobei den Taktiken systemintern Inferenzregeln zugrunde liegen. Das System führt die durch die Taktiken vermittelten Inferenzregeln aus. Dabei prüft es jeden Beweisschritt auf seine formale Korrektheit. Dies ist in erster Näherung mit dem Compilereinsatz beim Programmieren vergleichbar.

Für den Einsatz in der Studieneingangsphase ist es entscheidend, dass die Studierenden zum Arbeiten mit einem Beweisassistenten keinerlei typentheoretisches Vorwissen haben müssen. Weder müssen sie wissen, dass Theoreme Typen sind, noch, dass die zugehörigen Beweise Elemente solcher Typen, meist also Funktionen, sind. Für sie hat ein Beweis eines Theorems einfach einen Namen und kann durch eine Abfolge von Schritten gewonnen werden. Dadurch kommen Beweisassistenten wie Isabelle, NuPRL oder Coq überhaupt erst für eine didaktische Verwendung innerhalb einer Theorieveranstaltung in Frage.

Betrachten wir zur Verdeutlichung einen Beweis für $A \wedge B \rightarrow B \wedge A$ in einem Beweisassistenten. Um die Implikation zu beweisen, müssen wir $A \wedge B$ annehmen und zeigen, dass daraus $B \wedge A$ folgt. Der Beweisassistent muss daher eine entsprechende Taktik (*prove_imp*) zur Verfügung stellen. Aus

unserem Wissen um A ∧ B können wir dann zunächst das Wissen um die Einzelbestandteile, also A und B, generieren, wobei auch hier wieder eine entsprechende Taktik (*use_and*) vorhanden sein muss. Um schließlich B ∧ A zu beweisen, müssen wir einzeln sowohl B als auch A beweisen (*prove_and*). Dies ist nicht schwer, da uns die entsprechenden Aussagen bereits zur Verfügung stehen und Beweisassistenten erlauben in dem Fall daher den Abschluss des jeweiligen Beweises (*exact*). Die konkrete Umsetzung des Theorems in Coq ist im folgenden Schnappschuss zu sehen:



Auf der linken Seite ist der Beweis als Abfolge von Taktiken zu sehen, wobei die bereits ausgeführten Taktiken grün markiert sind. Das jeweils aktuelle Beweisziel (hier nach Ausführung der Taktik use_and) ist rechts zu sehen.

Ein möglicher Einwand gegen einen Einsatz von Beweisassistenten in der Studieneingangsphase könnte der Umstand sein, dass keiner von ihnen ursprünglich für eine didaktische Verwendung konzipiert wurde. Stattdessen sind sie für wissenschaftliche und industrielle Anwendungen gedacht und richten sich an Theoretiker und Mathematiker (Nederpelt und Geuvers, 2014). Trotzdem besitzen sie auch aus didaktischer Sicht verschiedene Vorteile gegenüber Stift und Papier, die unserer Meinung nach für die Eingangslehre in der Theoretischen Informatik außerordentlich wichtig sind:

- Lernende erhalten unmittelbares Feedback auf ihre Beweisidee, indem sie genau angezeigt bekommen, ob ein Beweisschritt erfolgreich anwendbar ist. Dies steht im Gegensatz zur üblichen Praxis, wo Studierende in den Übungen oftmals gar kein individualisiertes Feedback bekommen und die Rückgabe der Hausaufgaben erst eine Woche später erfolgt. Es kann dann auch nur eine Rückmeldung zum Gesamtprodukt und nicht zu den einzelnen Schritten bei der Beweisfindung gegeben werden. Dies wiederum verhindert einen konstruktiven Umgang mit fehlerhaften Ansätzen.

- Bei Beweisassistenten bekommen Lernende eine direktes Feedback, welche Schritte sie noch ausführen müssen, damit ein Beweis vollständig abgeschlossen ist. Genauer zeigt ein Beweisassistent immer an, welcher Teilschritt als nächstes zu zeigen ist und welche Voraussetzungen dabei genutzt werden können. Dies ist gerade zu Beginn sehr hilfreich, denn den Studierenden fehlt oft eine klare Vorstellung davon, wie sie konkret vorgehen müssen, um einen Beweis zu entwickeln (Knobelsdorf und Frede, 2016).
- Lernende werden an eine präzise und formal korrekte Vorgehensweise gewöhnt, da Beweisassistenten zu einer kompromisslosen Anwendung mathematischer Formalismen zwingen. Es ist bspw. nicht mehr möglich, statt dem formalen Argument eine Zeichnung aufs Blatt zu bringen und auf die Milde des Korrektors zu hoffen. Die formalen Regeln müssen auf genau die formalen Objekte angewandt werden. Hat etwa eine Eingabe in eine Funktion den falschen Typ, so geht es an dieser Stelle nicht weiter, bevor der Studierende die richtige Einsetzung macht. Dies hat den entscheidenden Vorteil, dass Fehler frühzeitig erkannt und in konstruktiver Art und Weise behoben werden können.
- Lernende können zu Beginn verschiedene Beweisideen ausprobieren, indem sie mit dem jeweiligen System spielen. Das garantiert höhere Flexibilität gegenüber Stift- und Papierbeweisen, in denen man nur sehr schlecht Dinge streichen oder verbessern kann, ohne den Beweis neu aufschreiben zu müssen. Aus dem gleichen Grund sollte ein Beweisassistent auch besser darin unterstützen, mit einem Beweis überhaupt zu beginnen.

Für die Entwicklung eines didaktischen Lehr-Lern-Konzepts haben wir den Beweisassistenten Coq ausgewählt, da er eine sehr gute Dokumentation sowie eine vergleichsweise einfach zu verstehende Schnittstellengestaltung besitzt. Auch die Installation von Coq ist deutlich einfacher umzusetzen. Ferner gehört Coq zu den Beweisassistenten, die am meisten in der Hochschullehre, insbesondere in der Studieneingangsphase, erprobt wurden.

3. Entwicklung eines didaktischen Lehr-Lern-Konzepts

3.1 Vermittlungsrahmen

Für die konkrete Vermittlung relevanter Kompetenzen in der Auseinandersetzung mit Coq und Beweisaufgaben aus der Theorie schlagen wir den Cognitive Apprenticeship Ansatz vor (Collins, 1991). Diesen didaktischen Ansatz haben wir bereits im Rahmen der Lehrveranstaltung „Theoretische Informatik I“ im Bachelorstudiengang Informatik an der Universität Potsdam erfolgreich eingesetzt (Knobelsdorf, Kreitz und Böhne, 2014), (Knobelsdorf 2015).

Der von Collins et al. erarbeitete Ansatz plädiert dafür, die in einem Fach eingesetzten Fach- und Handlungskompetenzen stärker in den Fokus der Lehre zu setzen. In der universitären Lehre wird traditionell auf deklaratives Fachwissen und damit auf die aus den Tätigkeiten der Fachcommunity hervorgegangenen Wissens-„Produkte“ fokussiert. Das wird in der Theoretischen Informatik besonders deutlich, wo Modelle, Theoreme und korrespondierende Beweise vorgestellt werden, während die für die Entwicklung benötigten Fachkompetenzen eher implizit thematisiert werden. Letztere müssen sich die Studierenden daher aus den entsprechenden Vorlesungsinhalten selbst rekonstruieren. Ein Unterfangen, das viele Studierende vor große Herausforderungen stellt, insbesondere wenn sie keine Affinität für diesen Bereich der Informatik haben. Dies liegt nicht zuletzt an der kognitiven Natur besagter Fachkompetenzen, deren korrespondierende Handlungen nur in Teilen beobachtbar sind und damit auch nur teilweise nachvollzogen werden können.

Collins et al. argumentieren nun, dass die Vermittlung solcher Fachkompetenzen daher im Wesentlichen von der Fähigkeit der Lehrenden abhängt, ihre Handlungen bewusst zu reflektieren und sprachlich explizit darzulegen. Collins et al. schlagen daher einen didaktischen Ansatz vor, der sich an der traditionellen Handwerksausbildung orientiert und zum Ziel hat, die kognitiven Fach- und Handlungskompetenzen zu explizieren und in den Fokus der Vermittlung zu setzen. Dieser Anfang der 1990er Jahre vorgeschlagene Ansatz greift damit die aktuelle Kompetenzorientierung im Bildungsbereich vor.

Der Ansatz schlägt mehrere Formen der Vermittlung und Gestaltung eines Lehr-Lern-Prozesses vor, die sich an einer konstruktivistischen Didaktik und

dem situierten Lernen orientieren (Reinmann-Rothmaier und Mandl, 2001). Dabei sind die folgenden drei Ansätze für uns besonders relevant:

1. Modeling: Die lehrende Person demonstriert relevante Fachhandlungen in der konkreten Auseinandersetzung am Beispiel. Lernende werden befähigt, diese beobachten und nachahmen zu können.
2. Coaching: In der aktiven Erprobung des Beobachteten werden Lernende von der lehrenden Person begleitet und erhalten direktes Feedback.
3. Scaffolding: Die Entwicklung der Fach- und Handlungskompetenzen wird durch ein helfendes Gerüst vorgegeben.

3.2 Gestaltung eines möglichen Kursablaufs

Wir wollen uns nun mit der Frage auseinandersetzen, wie ein konkreter Kurs auszusehen hat, der in der Lage ist, den Umgang mit mathematischen Formalismen mithilfe des Cognitive Apprenticeship Ansatzes erfolgreich zu schulen.

Zunächst wird der Modeling-Aspekt klassisch durch Vorlesungen realisiert, wobei im Gegensatz zu üblichen Vorlesungen das handwerkliche Rüstzeug nicht implizit bleibt, sondern den expliziten Gegenstand der Vorlesung darstellt. Genauso wie bei anderen Lehrveranstaltungen auch werden Übungen zu den Inhalten der Vorlesung angeboten, wobei auf die Übungen mindestens genauso viel Zeit verwendet wird wie auf die Vorlesungen selbst. Die Übungen unseres Kurses unterscheiden sich dabei jedoch drastisch von der üblichen Variante: Zwar werden die Studierenden in den Übungen, wie es üblich ist, ganz normal vom Lehrteam begleitet, welches die üblichen Hilfen anbieten kann, vor allem aber übernimmt nun der Beweisassistent, hier Coq, ständiges und individualisiertes Feedback zu den Lösungsansätzen der Studierenden (Coaching). Darüberhinaus wird durch das von uns "modifizierte" Coq-System, durch die Auswahl der Aufgaben sowie durch vorbereitete Coq-Dateien den Studierenden ein Gerüst an die Hand gegeben, das die Lernenden schrittweise in der Beweisführung unterstützt (Scaffolding). Die zu bearbeitenden Aufgaben sind so gestellt, dass sie unmittelbar das in der Vorlesung Vorgestellte aufgreifen und im Sinne des Cognitive-Apprenticeship-Ansatzes, sowie allgemeiner einer konstruktivistisch orientierten Didaktik, den Lernenden die Möglichkeit zur Erprobung und Ausgestaltung geben.

Am Ende der Übungen verbleibt ein gewisser Zeitrahmen (ca. 30 Minuten) für die Arbeitssicherung. Erst jetzt wird der Übungsleiter vor der gesamten Gruppe aktiv, indem er wesentliche Eckpunkte der Aufgabenlösung

vorstellt und dabei auf die stattgefundene Bearbeitungsphase eingeht. Die Studierenden haben hier die Möglichkeit, weitere Fragen zu stellen und gemeinsam abschließend in der Gruppe zu diskutieren.

3.3 Auswahl und Begründung der Themeninhalte

Als Inhalte für einen möglichen Kurs schlagen wir eine thematische Zweiteilung vor: Logik, genauer Aussagen- und Prädikatenlogik (beliebiger Stufe), und Datentypen, genauer: Verbundsdatentypen, Listen, Natürliche Zahlen sowie Binärbäume. Der thematische Start mit Logik erfolgt im wesentlichen aus zwei Gründen:

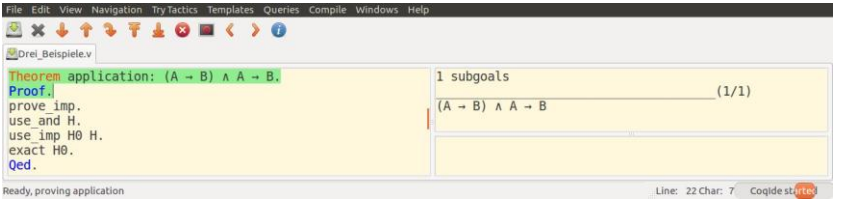
1. Die Logik stellt eine ungetrübte mathematische Argumentationsstruktur in Reinform dar. Die übliche mathematische Argumentation hingegen vermengt inhaltliche Intuition und logische Argumentation. Für die Lehre hat letzteres den entschiedenen Nachteil, dass die von den Mathematikexperten verwendeten logischen Methoden und Kompetenzen für die Studierenden unsichtbar bleiben müssen, da sie von inhaltlichen Erwägungen überdeckt werden. Durch die Konzentration auf Logik rufen wir die Existenz solcher zugrundeliegender Methoden und Kompetenzen ins Bewusstsein und machen sie explizit.
2. Die Klarheit der Logik spiegelt sich auch in der einfachen Handhabung innerhalb des Beweisassistenten wider. Während inhaltliche Theorien einiger Vorbereitung in Coq bedürften, können die Studierenden bei Aussagen zur Logik direkt mit der Arbeit beginnen. Dazu bedarf es lediglich einiger vorbereiteter Taktiken, die von den Studierenden aufgerufen werden können, so wie sie auch vorgegebene Prozeduren oder Methoden beim Programmieren nutzen.

Die Auseinandersetzung mit Datentypen, als konkretem inhaltlichen Thema in der zweiten Hälfte eines möglichen Kursablaufs führt dann zurück in den mathematischen "Argumentationsalltag". Die Behandlung von Datenstrukturen ergibt sich dabei notwendigerweise, weil die Inhalte erst konstruiert werden müssen, über die später Beweise zu führen sind. Darüberhinaus bieten sich die konkret gewählten Datentypen besonders an, da sie relativ einfach sind und zum Kernkanon in der Informatiklehre gehören.

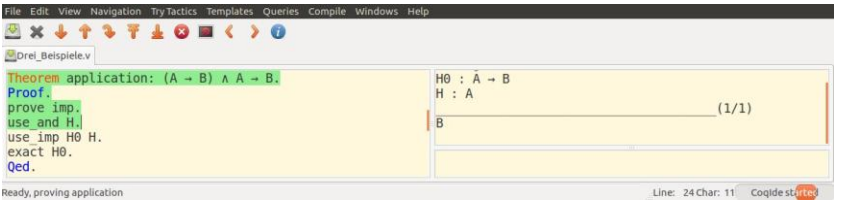
3.4 Erläuterungen an zwei Beispielen

Im Folgenden stellen wir zwei Beispieltheoreme und ihren Beweis in Coq vor. Das erste Theorem ist Teil der Aussagenlogik, wohingegen das zweite die Kommutativität der natürlichen Zahlen behauptet.

Das erste Theorem lautet: $(A \rightarrow B) \wedge A \rightarrow B$. In Coq geben wir dem Theorem einen Namen, application, und beginnen den Beweis mit dem Schlüsselwort Proof:

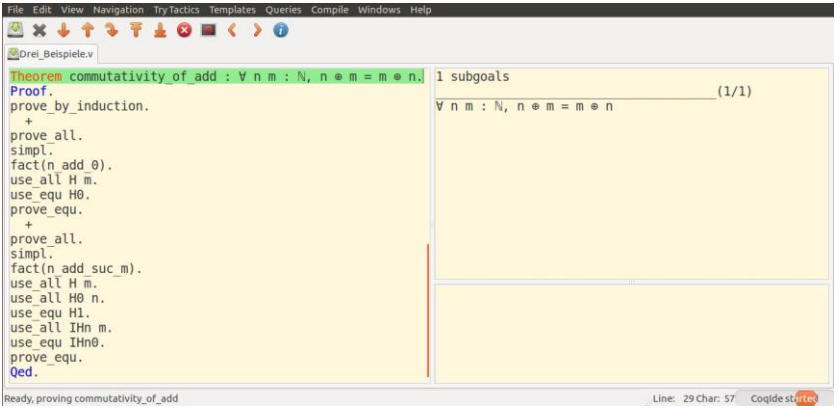


Mit dem Schlüsselwort *prove_imp* geben wir nun an, dass wir eine Implikation beweisen wollen. Daraufhin steht uns die Prämisse im rechten Fenster zur Verfügung, um die Konklusion herzuleiten. Vermittelst *use_and* können wir die uns zur Verfügung stehende Konjunktion in ihre beiden Bestandteile aufteilen:



Wir wenden nun mit *use_imp* die Implikation $A \rightarrow B$ und die Prämisse A an, um B zu erhalten. Dies ist aber genau die zu beweisende Aussage, weswegen wir unseren Beweis mit *exact* abschließen können.

Das zweite Theorem lautet: $\forall\ n\ m : \mathbb{N}\ ,\ n \oplus m = m \oplus n$. Seinen Beweis wollen wir nicht in seinen Einzelheiten besprechen. Stattdessen erläutern wir lediglich die wesentlichen Unterschiede gegenüber dem obigen Beispiel. Zunächst stellen wir den Beweis in Coq vor:



Der Beweis in diesem Beispiel ist ein Induktionsbeweis und deutlich länger als im vorhergehenden Beispiel , obwohl wir bereits auf die beiden Lemmata *n_add_0* und *n_add_suc_m* zurückgreifen. Diese Lemmata wurden mithilfe von Induktion bewiesen. Dieses Mittels bedienen wir uns auch in diesem Beweis. Es sind also zwei Teilziele zu zeigen: Der Induktionsanfang und der Induktionsschritt, die jeweils durch ein '+' sichtbar voneinander getrennt dargestellt sind. Der Beweis besitzt hier also mehr Struktur als im obigen Beispiel. Darüberhinaus werden in diesem Beispiel Regeln rund um die Allquantifikation und die Gleichheit genutzt, für die es im aussagenlogischen Fall keine Entsprechung gibt.

4 Fazit und Ausblick

Mit dem vorgestellten Konzept wird ein für die Eingangslehre der Theoretischen Informatik neues Lernwerkzeug vorgeschlagen. Dieses kann nicht nur innovative Impulse für eine Neugestaltung der Lehre in diesem Bereich liefern, sondern insbesondere den Lernprozess und die damit einhergehende Fachkompetenzentwicklung individuell unterstützen und die Studierenden dazu anregen, selbstbestimmtes Lernen und Arbeiten in diesem Bereich zu intensivieren.

Das hier vorgestellte Lehr-Lern-Konzept mit dem Einsatz von Coq als Lernwerkzeug ist nicht auf bestimmte Themen, Syntax oder einen speziellen Ablauf beschränkt. Es lässt sich ohne prinzipielle Probleme auf alle mathematischen und theoretischen Inhalte übertragen. Zugestandenermaßen erfordert jedes Themengebiet eine didaktische Formalisierung und die entsprechende Vorbereitung des Systems. Die technische Umsetzung so wie

die grundsätzlichen Kernüberlegungen können aber im Wesentlichen übernommen werden. Mögliche neue Einsatzgebiete sind denkbar für die Studieneingangsphase der Mathematik und auch der Physik, wo ähnlich der Informatik mathematische Grundlagen erlernt und dann auf das eigene Fach übertragen werden.

Um den Einsatz unseres Konzepts zu erproben, planen wir Anfang Oktober 2016 am Fachbereich Informatik der Universität Hamburg einen entsprechenden Kurs durchzuführen. Die Veranstaltung ist als zweiwöchige Blockveranstaltung konzipiert und richtet sich an Bachelor-Studierende der Informatik und Wirtschaftsinformatik, die im Wintersemester 2016/2017 ins dritte oder höhere Fachsemester kommen. Parallel dazu wird eine empirische Evaluation des Konzepts stattfinden. Die dabei gewonnen Erkenntnisse sollen genutzt werden, um das Konzept weiterzuentwickeln und ein für Coq passendes Schnittstellendesign zu konzipieren, welches insbesondere Studierende in der Studieneingangsphase anspricht.

Literatur

Allen, S., Bickford, M., Constable, R., Eaton, R., Kreitz, C., Lorigo, L. und Moran, E. 2006. Innovations in Computational Type Theory using Nuprl. Journal of Applied Logic, 4(4):428-469.

Bertot, Y. und Casteran, P. 2004. Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions, Springer.

Bove, A., Dybjer, P. und Norell, U. 2009. A Brief Overview of Agda - A Functional Language with Dependent Types. In TPHOLs 2009, LNCS 5674, Springer, 73-78.

Brady, E. 2013. Idris, a General Purpose Dependently Typed Programming Language: Design and Implementation. Journal of Functional Programming, 23(5):552-593.

Collins, A., Brown, J. S. und Holum, A. 1991. Cognitive apprenticeship: Making thinking visible. American Educator, 6(11):38-46.

Constable, R., et.al. 1986. Implementing Mathematics with the Nuprl Development System, Prentice-Hall.

Delahaye, D., Jaume, M. und Prevosto, V. 2005. Coq, un outil pour l'enseignement. Technique et Science Informatiques, 24(9):1139-1160.

Henz, M. und Hobor, A. 2011. Teaching Experience: Logic and Formal Methods with Coq. In Certified Programs and Proofs, Lecture Notes in Computer Science, Vol. 7086, Springer, 199-215.

Knobelsdorf, M. 2015. The Theory Behind Theory - Computer Science Education Research Through the Lenses of Situated Learning. In Proceedings of ISSEP Conference, Lecture Notes in Computer Science, Vol. 9378, Springer, 21-21.

Knobelsdorf, M. und Frede, C. 2016. Analyzing Student Practices in Theory of Computation in Light of Distributed Cognition Theory. In Proceedings of the 12th ICER Conference, ACM, in press.

Knobelsdorf, M., Kreitz, C. und Böhne, S. 2014. Teaching theoretical computer science using a cognitive apprenticeship approach. In Proceedings of the 45th SIGCSE Conference, ACM, 67-72.

Nederpelt, R. und Geuvers, H. 2014. Type Theory and Formal Proof: An Introduction, Cambridge University Press.

Nipkow, T., Paulson, L. und Wenzel, M. 2002. Isabelle/HOL A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, Vol. 2283, Springer.

Reinmann-Rothmeier, G. und Mandl, H. 2001. Unterrichten und Lernumgebungen gestalten. In A. Krapp und B.Weidenmann (Hrsg): Pädagogische Psychologie. Weinheim Beltz/PVU, 601-646.

Sakowicz, J. und Chrząszcz, J. 2007. Papuq, a Coq assistant. In Proceedings of PATE'07 conference, Elsevier, 79-96.