

Ein erweiterbares interaktives Online-Übungssystem mit Aufgaben zu Aussagen- und Prädikatenlogik

Immo Schulz-Gerlach, Christoph Beierle
Fakultät für Mathematik und Informatik
FernUniversität in Hagen
58084 Hagen

Immo.Schulz-Gerlach@fernuni-hagen.de, Christoph.Beierle@fernuni-hagen.de

Abstract: Um insbesondere Fernstudierenden die Kontrolle des eigenen Lernerfolgs zu erleichtern und darüber hinaus durch aktive Hilfestellung den Lernerfolg der Studierenden noch steigern zu können, wurde ein vollautomatisch arbeitendes Online-Übungssystem realisiert, über welches derzeit interaktive Aufgaben zu Aussagen- und Prädikatenlogik angeboten werden. In dieser Arbeit stellen wir das System, die derzeit implementierten Aufgaben und das derzeitige Einsatzgebiet an der FernUniversität in Hagen vor.

1 Einleitung

Der sichere Umgang mit den elementaren Grundlagen der Logik gehört zu den wichtigen Lernzielen in der Informatikausbildung. Um diesen Umgang zu erlernen, können Aufgaben, mit denen die Studierenden ihren eigenen Lernfortschritt selbst kontrollieren können, ein wichtiges Hilfsmittel sein. Das nach der Idee solcher Selbstkontrollaufgaben benannte SKA-System, das wir in dieser Arbeit vorstellen, ist ein interaktives Online-Übungssystem, das Aufgaben zur Aussagen- und Prädikatenlogik bereitstellt, um so die Vermittlung der formalen Grundlagen der Informatik in diesem Bereich zu unterstützen.

SKA wurde als ein ergänzendes Angebot für den Studienbetrieb an der FernUniversität in Hagen entwickelt. Neben den eigentlichen *Kurstexten*, die den für das Selbststudium aufbereiteten Vorlesungsstoff enthalten, erhalten die Studierenden an der FernUniversität zwei unterschiedliche Typen von Aufgaben: *Selbstkontrollaufgaben*, die normalerweise in den Kurstext integriert sind und zu denen auch Musterlösungen im Anhang des jeweiligen Kurstextes geliefert werden, und die sog. *Einsendeaufgaben*.

Nach der Bearbeitung der im Kurstext enthaltenen Selbstkontrollaufgaben können die Studierenden ihre Lösung mit der Musterlösung vergleichen, erhalten aber keine Korrektur ihrer eigenen Lösung. Die Frage, ob die eigene Lösung korrekt ist, bleibt daher leicht offen, und auch die Möglichkeiten zur Diskussion der Lösung mit Kommilitonen oder zur Gruppenarbeit sind bei Fernstudenten gegenüber Studierenden an Präsenzuniversitäten eingeschränkt.

Die eingesandten Lösungen zu den Einsendeaufgaben dagegen werden korrigiert, wodurch

die Teilnehmer eine individuelle Rückmeldung über ihren Lernerfolg erhalten. Nachteilig ist dabei jedoch, dass zum einen die Korrektur der Einsendungen schon aus organisatorischen Gründen nur zeitversetzt erfolgen kann und dass zum anderen die Bearbeitung der Aufgabe schon mit der Einsendung abgeschlossen ist und in der Regel die Lösung nach der Rückmeldung nicht erneut verbessert und eingesandt werden kann.

Das an der FernUniversität im Einsatz befindliche WEBASSIGN-System [BHSV99] lindert beide genannten Nachteile, indem durch Online-Bearbeitung, -Einsendung und -Korrektur der Einsendeaufgaben insbesondere die Postlaufzeiten herkömmlicher Papiereinsendungen vermieden und die Verzögerungen im Wesentlichen auf die Korrekturzeit beschränkt werden, und indem der Student (zumindest bei ausgewählten Aufgaben) mit einer automatischen Vorkorrektur sofort bei der (vorläufigen) Einsendung ein erstes Feedback erhält und seine Lösung bis zum Abgabetermin daraufhin noch beliebig oft überarbeiten darf, bevor nach der Abgabe die endgültige Korrektur und Bewertung der Einsendung erfolgt.

Das hier beschriebene SKA-System soll dagegen — als Ergänzung zu gedruckten Selbstkontrollaufgaben mit Musterlösungen — interaktive, automatisch ausgewertete Selbstkontrollaufgaben ermöglichen, bei denen der Student ohne Wartezeit sofort Rückmeldungen und auf Wunsch sogar aktive Unterstützung bei der Lösung der Aufgabe erhält. Um eine möglichst breite Verfügbarkeit sicherzustellen und so wenig Anforderungen wie möglich an die Hard- und Softwareausstattung der privaten Computer der Studenten zu stellen, wurde das System als Webanwendung realisiert. Zur Nutzung genügt ein schmalbandiger Internetzugang, der heute ohnehin eine Mindestvoraussetzung zum Informatikstudium an der FernUniversität darstellt.

SKA wird derzeit an der FernUniversität Hagen im Rahmen des Erstsemester-Kurses „Formale Grundlagen der Informatik“ [BH03] eingesetzt, um interaktive Übungsaufgaben zu Aussagen- und Prädikatenlogik anzubieten, und zwar vornehmlich solche zur Bildung bestimmter Normalformen. Das Umformen einer gegebenen Formel in eine bestimmte Normalform durch eine Folge von Äquivalenzumformungen ist eine fehlerträchtige Aufgabe, die erfahrungsgemäß sehr vielen Studierenden am Anfang ihrer Informatikausbildung erhebliche Schwierigkeiten bereitet — wie viele Newsgruppen-Beiträge von Studierenden zeigen, die untereinander diskutieren, ob eine Umformung nun korrekt oder falsch ist. Ein einziger Umformungsfehler hat dabei gravierende Auswirkungen auf das Ergebnis. Unsere Übungsaufgaben werten daher jede einzelne vorgenommene Umformung des Studierenden sofort aus und lassen ihn Fehler sofort korrigieren, damit sich keine Folgefehler durch die Bearbeitung der Aufgabe ziehen. Auch steht das System dem Nutzer mit Rat und Tat zur Seite, wenn er unsicher ist, wie als nächstes fortzufahren ist.

Der folgende Abschnitt 2 soll einen ersten Eindruck von der Benutzung des SKA-Systems vermitteln, wozu beispielhaft eine konkrete Aufgabe vorgestellt wird. Abschnitt 3 gibt an, welche Aufgabentypen SKA derzeit unterstützt und wie neue Aufgaben erstellt werden können. Abschnitt 4 beschreibt die Verfahren zur automatischen Auswertung und zur Ermittlung von Tipps, während in Abschnitt 5 auf die Implementierung des SKA-Systems in Java eingegangen wird. Abschließend folgen eine Betrachtung verwandter Arbeiten in Abschnitt 6 sowie Zusammenfassung und Ausblick in Abschnitt 7.


2 Aufgabenbearbeitung im SKA-System

In diesem Abschnitt zeigen wir die Benutzung des Systems aus Studierendensicht am Beispiel einer bestimmten Selbstkontrollaufgabe: Der Student soll zu einer gegebenen aussagenlogischen Formel eine äquivalente Formel in konjunktiver Normalform (KNF, vgl. [Sch00]) ermitteln. Das Aufgabenformular zeigt Abbildung 1.

Äquivalenzumformungen zu KNF

Bringen Sie folgende Formel in konjunktive Normalform, indem Sie schrittweise Äquivalenzumformungen vornehmen und jede Umformung mit "Auswerten" überprüfen lassen:

$\neg(A \vee B \vee O) \vee (C \rightarrow O) \vee \neg\neg L$

 Hilfe zur Syntax

Liste aller bisherigen Umformungen anzeigen

Die folgende Formel ist nun weiter umzuformen:

$\neg(A \vee B \vee O) \vee (C \rightarrow O) \vee \neg\neg L$

Umgeformte Formel eingeben:

Letzte F. kopieren

Fertig, die Formel ist in KNF.

Abbildung 1: Aufgabenformular

Aus technischen Gründen beschränken wir uns bei der Darstellung von Formeln auf reinen ASCII-Text, weshalb für Junktoren und Quantoren folgende Ersatzdarstellungen festgelegt wurden:

Junktor / Quantor:	¬	∨	∧	→	↔	∀	∃
ASCII:	~		&&	->	<->	{A}	{E}

Eine Syntax-Hilfeseite ist über einen Hyperlink in der Aufgabenseite jederzeit erreichbar.

2.1 Bearbeitung als Folge von Einzelschritten

In Aufgaben dieses Typs wird nicht vom Studenten verlangt, die gesamte Umformung allein vorzunehmen und sein Ergebnis einzugeben, sondern er darf jederzeit einfache Umformungsschritte vornehmen und einzeln auswerten lassen. Nach jeder Umformung erhält er eine Rückmeldung, ob diese korrekt war. Im Falle einer syntaktisch nicht korrekten Eingabe oder der Eingabe einer nicht zur Ausgangsformel äquivalenten Formel wird er auf den Fehler hingewiesen (vgl. Abbildungen 2 und 3) und kann seine Eingabe korrigieren. Erst wenn die Umformung fehlerfrei war, kann mit weiteren Schritten fortgefahren

werden, was insbesondere Folgefehler vermeidet. In der Auswertung wird dann an Stelle einer Fehlermeldung eine Erfolgsmeldung angezeigt, die zuletzt eingegebene Formel wird zur neuen umzuformenden Formel und das Eingabefeld wird für eine neue Eingabe geleert.

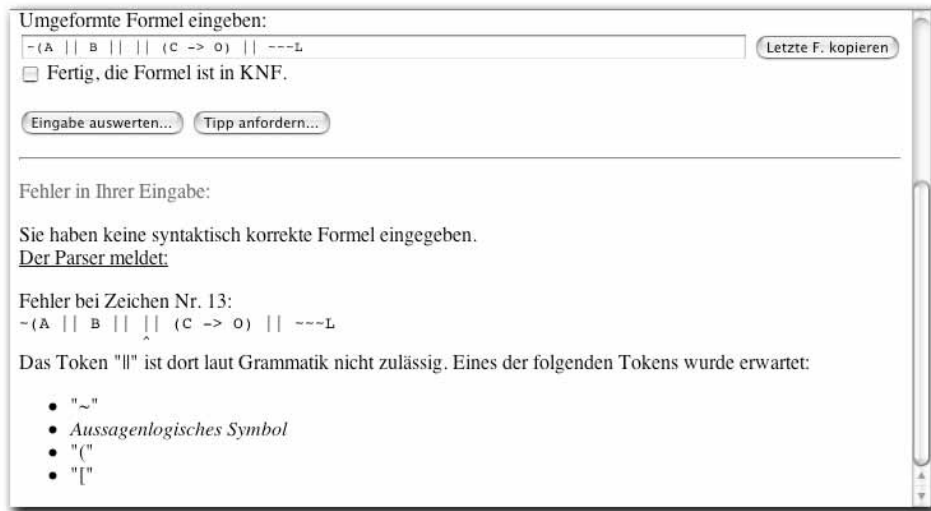


Abbildung 2: Syntaxfehler in einer Eingabe

Um die Aufgabenseite nicht unnötig mit Informationen zu überladen, werden die schon vorgenommenen Umformungen standardmäßig nicht angezeigt, sondern immer nur die Ausgangsformel in der Aufgabenstellung und die als nächstes umzuformende Formel, die der Nutzer im letzten Schritt eingegeben hat. Man kann aber jederzeit die Liste aller schon vorgenommenen Schritte ein- und wieder ausblenden lassen (vgl. obere Checkbox in Abb. 1).

Die Aufgabe ist gelöst, sobald die eingegebene Formel in der geforderten Normalform ist. Das soll der Nutzer jedoch selbst erkennen, weshalb vom System nicht automatisch darauf hingewiesen wird. Sobald der Nutzer der Meinung ist, dieses Ziel erreicht zu haben, soll er die unterhalb der Eingabezeile befindliche Checkbox „Fertig, die Formel ist in KNF“ ankreuzen und seine Eingabe erneut auswerten. Dann erhält er nicht nur eine Rückmeldung, ob seine Formel äquivalent zur Ausgangsformel ist, sondern auch, ob sie tatsächlich in der geforderten Normalform ist. Aber auch mit Erreichen der Normalform ist die Aufgabe nicht immer *optimal* gelöst: Es kann hier unter Umständen ein Hinweis ausgegeben werden, dass die eingegebene Formel zwar tatsächlich in der Normalform ist, sich jedoch noch vereinfachen lässt. Das System kennt mehrere Möglichkeiten, Formeln in KNF noch weiter zu vereinfachen — und gibt auf Anforderung auch entsprechende Tipps.

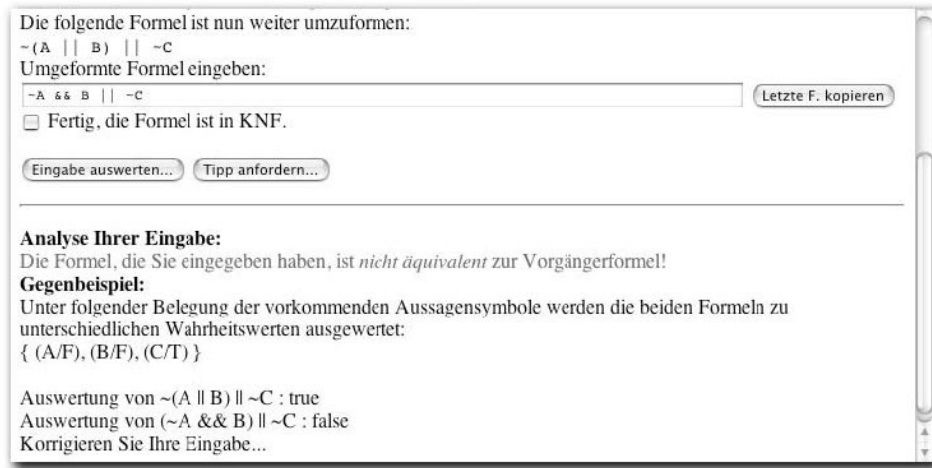


Abbildung 3: Fehlerhafte Umformung

2.2 Tipp-Funktion

In der Einleitung ist bereits angeklungen, dass das SKA-System neben der reinen Auswertung von Nutzer-Eingaben auch beratend tätig werden kann. Wann immer der Student unsicher ist, wie er weiter vorzugehen hat, kann er einen Tipp anfordern. Dieser Tipp ist zunächst rein verbal und wird unterhalb des Aufgabenformulars eingeblendet. Bei einer Normalformaufgabe wird der Tipp eine bestimmte Umformungsregel nennen und vorher deren zielgerichteten Einsatz motivieren, z.B. indem Eigenschaften der umzuformenden Formel aufgezeigt werden, welche die Normalform nicht mehr haben darf. Als weitere Hilfe wird meist zusätzlich im Aufgabenformular eine Markierung an der vom Tipp betroffenen Teilformel angebracht (durch farbliche Hinterlegung sowie Einfassung in eckige Klammern). Abbildung 4 zeigt ein Beispiel für einen solchen Tipp.

Nun kann der Benutzer versuchen, die im Tipp genannte Regel selbst anzuwenden. Alternativ kann er aber auch — zusätzlich zu dem verbalen Tipp — einen Umformungsvorschlag anfordern (vgl. Punkt 3. in Abbildung 4). Dann wird das System das Ergebnis der im Tipp vorgeschlagenen Umformung direkt in die Eingabezeile des Formulars eintragen. Der Benutzer kann diese automatisch generierte Formel daraufhin auswerten lassen, so als hätte er sie selbst eingegeben.

2.3 Lernmodell

Offenbar eignen sich diese Aufgaben nicht nur zur Selbstkontrolle, sondern auch zum aktiven, *explorativen Lernen*. So können Fragen der Art „Was passiert eigentlich, wenn ...?“

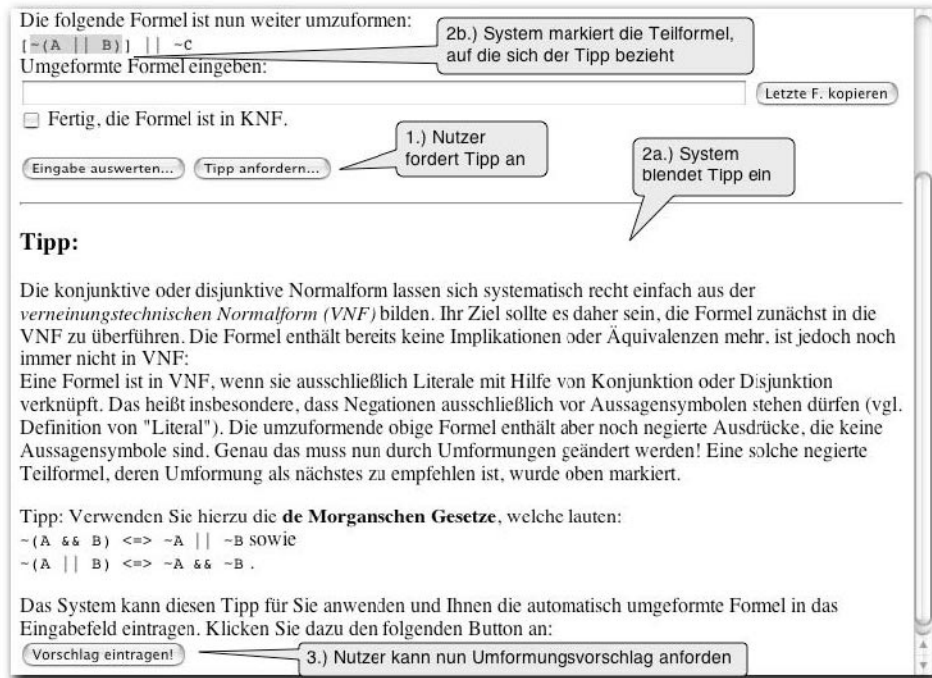


Abbildung 4: Ein Tipp wurde angefordert

oder „Darf ich eigentlich auch ...?“, wie sie häufig in Newsgruppen gestellt werden, durch einfaches Ausprobieren beantwortet werden. Der Student muss sich dabei nicht davor scheuen, Fehler zu machen, denn diese werden ihm von niemandem angelastet, sondern tragen vielmehr zum Lernerfolg bei. Der Nutzer kann sich (durch wiederholtes Anfordern von Tipps) die Lösung einer Aufgabe komplett vorführen lassen oder (am besten beginnend mit einfacheren Aufgaben) sich selbst an deren Lösung versuchen, wobei ihm das System hilfreich zur Seite steht, indem es auf Fehler aufmerksam macht oder Tipps gibt. In dieser Hinsicht bestehen Analogien zur Handwerkslehre und dem darauf aufbauenden *Cognitive Apprenticeship Model* [CBN89].

3 Unterstützte Aufgabentypen und Aufgabenerstellung

Das SKA-System bietet eine Reihe unterschiedlicher Aufgabentypen zur (klassischen) Logik an. In der Aussagenlogik kann der Umgang mit aussagenlogischen Belegungen eingeübt (Konstruktion von erfüllenden oder falsifizierenden Belegungen, Heuristiken für die Vorgehensweise) oder das Erstellen von Normalformen trainiert werden (schrittweise Überführung in KNF bzw. DNF (disjunktive Normalform), Umformung von DNF in

KNF, etc.) (vgl. [Sch00]).

Bei den Aufgaben zur Prädikatenlogik geht es nicht nur um die Erstellung von KNF und DNF einer Formel, wobei die letzten Schritte der Verfahren zur Bildung dieser Normalformen aus der Aussagenlogik übernommen werden können. Die für die Prädikatenlogik charakteristischen Schritte sind vielmehr das Bilden der bereinigten Form einer Formel und die Überführung in Pränexform (vgl. [Sch00, Wal03]). Hierzu bietet SKA ein Reihe von Aufgaben unterschiedlicher Komplexität an.

Um eine neue Aufgabe eines unterstützten Aufgabentyps zu erstellen, muss lediglich der gewünschte Aufgabentyp ausgewählt und die Ausgangsformel als Parameter angegeben werden. Als Aufgabentyp könnte beispielsweise die Bildung der Pränexform ausgewählt und als Ausgangsformel z.B. $(\exists u P(u)) \wedge (\exists x \forall y \neg \exists z R(x, y, z))$ angegeben werden. Alle weiteren Komponenten zur interaktiven Darstellung und Bearbeitung und zur automatischen Auswertung, Korrektur und Tippermittlung werden dann automatisch von SKA bereitgestellt. Lediglich für die Realisierung neuer Typen von Aufgaben muss das SKA-System um die Implementierung entsprechender Java-Klassen erweitert werden.

4 Automatische Auswertung / Korrektur und Tippermittlung

Prinzipiell lässt sich das System zur Umsetzung verschiedenartigster Aufgabentypen einsetzen (vgl. auch Abschnitt 5), so lange für das gewählte Problemfeld eine automatische Auswertung der Nutzereingaben umsetzbar ist. In diesem Abschnitt gehen wir auf die Realisierung der Auswertung in den vorgestellten Aufgabentypen zu Aussagen- und Prädikatenlogik ein.

Für die genannten Aufgaben der *Aussagenlogik* war zunächst die Möglichkeit zu schaffen, eine aussagenlogische Formel unter einer gegebenen Belegung ihrer Variablen auszuwerten, was kein großes Problem darstellt. Darauf aufbauend konnte eine Äquivalenzprüfung für beliebige aussagenlogische Formeln nach der sog. Wahrheitstafelmethode realisiert werden, wozu nur eine Auswertung mit allen möglichen Belegungen aus einer endlichen Menge nötig ist. Damit können die Nutzereingaben problemlos überprüft werden. Aufwendiger ist dagegen die Tippermittlung. Hierzu sind Algorithmen zur automatischen Umformung beliebiger aussagenlogischer Formeln in KNF bzw. DNF implementiert worden, und ein Tipp basiert auf genau dem Umformungsschritt, den der jeweilige Algorithmus als nächstes anwenden würde. Durch die automatische Anwendung dieses Schritts wird auch gleichzeitig der Umformungsvorschlag ermittelt, den der Student einblenden lassen kann.

Für die Aufgaben zur *Prädikatenlogik* wurden entsprechend Algorithmen zur automatischen Bildung der bereinigten Form bzw. Pränexform realisiert. Problematisch ist allerdings die Auswertung der Benutzereingaben, da für zwei beliebige prädikatenlogische Formeln nicht durch einen Algorithmus entschieden werden kann, ob sie semantisch äquivalent sind (Unentscheidbarkeit der Prädikatenlogik, vgl. [Sch00]) — weshalb bei Aufgaben, in denen prädikatenlogische Formeln in eine Normalform zu bringen sind, nicht immer eindeutig eine Antwort aus {„richtig“, „falsch“} gegeben werden kann, sondern als dritte Antwortmöglichkeit noch „nicht nachvollziehbar“ hinzukommt.

Da die Normalformbildung, wie sie üblicherweise in Lehrbüchern dargestellt wird, aufgrund von Umformungsschritten erfolgt, die in einer bestimmten Richtung (Auflösen von Implikationen, Negationen „nach innen“ etc.) ausgeführt werden, ist es wünschenswert, zumindest diese Umformungsschritte als korrekt zu erkennen. Wir haben daher einen Algorithmus zum Vergleich der Benutzereingabe mit der umzuformenden Formel entwickelt (in [SG04]), der diese Eigenschaft aufweist und, grob skizziert, nach folgendem Prinzip arbeitet:

- Bezeichne F die umzuformende Ausgangsformel und B die Benutzereingabe, z.B.:

$$F = (\exists u P(u)) \wedge (\exists x \forall y \neg \exists z R(x, y, z))$$

$$B = (\exists x \forall z \forall y \neg R(x, y, z)) \wedge \exists u P(u)$$

- Die Benutzereingabe B wird automatisch in Pränexform PB umgeformt:

$$PB = \exists x \forall z \forall y \exists u \neg R(x, y, z) \wedge P(u)$$

- Das Präfix von PB , also die Folge von Quantifizierungen¹, wird mit der umzuformenden Formel F abgeglichen, indem überprüft wird, ob keine Quantifizierung $Q1$, die in F im Wirkungsbereich einer andersartigen² Quantifizierung $Q2$ steht, in PB den Wirkungsbereich von $Q2$ verlassen hat (andernfalls wäre die Umformung möglicherweise falsch, Ergebnis: „nicht nachvollziehbar“). Zusätzlich wird noch geprüft, ob keine Quantifizierung fehlt und keine neu hinzugekommen ist und ob jede Quantifizierung auch den korrekten Quantor aufweist (Quantifizierungen, die in F im Wirkungsbereich ungerade vieler Negationen stehen, wechseln ihren Quantor bei der Pränexformbildung, die anderen behalten ihren Quantor).

Im obigen Beispiel stehen die beiden Quantifizierungen über y und z im Präfix von PB hinter $\exists x$, haben also im Zuge der Umformung den Wirkungsbereich von $\exists x$ nicht verlassen. Sie wurden zwar untereinander vertauscht, was jedoch aufgrund des (nach Auflösen der Negation in F) gleichen Quantors (\forall) erlaubt ist³. Das Präfix von PB ist also in Ordnung.

- Ist das Präfix von PB in Ordnung, so wird auch F automatisch in Pränexform PF gebracht, und die Matrizen von PB und PF (quantorenfreie Formeln) werden bijektiv auf aussagenlogische Formeln abgebildet und auf Äquivalenz überprüft. Hierzu wird der schon für die Aussagenlogik implementierte Algorithmus wieder verwendet, der an dieser Stelle sogar ein Entscheidungsverfahren liefert: Sind die Matrizen nicht äquivalent (aber war die vorangehende Präfixprüfung erfolgreich), so gilt $F \not\equiv B$, d.h. es erfolgt die Ausgabe „Fehlerhafte Umformung“. Für das obige Beispiel erhalten wir: $PF = \exists u \exists x \forall y \forall z P(u) \wedge \neg R(x, y, z)$

Die beiden Matrizen $P(u) \wedge \neg R(x, y, z)$ sowie $\neg R(x, y, z) \wedge P(u)$ von PF und PB sind äquivalent, was durch Abbilden auf die aussagenlogische Formel $AF := A \wedge \neg C \leftrightarrow \neg C \wedge A^4$ und Nachweis, dass AF tautologisch ist, gezeigt wird. Somit gilt $F \equiv B$ und die Benutzereingabe war korrekt.

¹Quantifizierung: Paar aus einem Quantor und der durch diesen gebundenen Variablen

²Mit andersartig ist gemeint, dass beide Quantifizierungen (nach Auflösen aller Negationen vor Quantoren) verschiedene Quantoren aufweisen.

³Es gilt $\forall x \forall y F(x, y) \equiv \forall y \forall x F(x, y)$ bzw. $\exists x \exists y F(x, y) \equiv \exists y \exists x F(x, y)$

⁴Die Wahl der Symbole A und C ist willkürlich. Wichtig ist lediglich, dass die in der prädikatenlogischen Formel vorkommenden Atome bijektiv auf Aussagensymbole abgebildet werden.

Dieser Algorithmus ist allerdings nicht in jedem Fall anwendbar, denn er setzt voraus, dass F und B in bereinigter Form sind und die Mengen der in PF bzw. PB vorkommenden Atome identisch sind. Die einzige zur Normalformbildung benötigte Umformung, die zur Veränderung von Atomen führt, ist das Bilden der bereinigten Form durch gebundene Umbenennung von Variablen. Wir haben daher zwei Teilaufgaben implementiert: In der ersten ist die gegebene Formel in bereinigte Form zu bringen. Es gibt ein Verfahren, mit dem entschieden werden kann, ob die Nutzereingabe B aus der Ausgangsformel F durch (ggf. mehrfaches) gebundenes Umbenennen hergeleitet werden kann. Damit werden die Benutzereingaben überprüft. Ist die Formel in bereinigter Form, folgt die zweite Teilaufgabe, in der der Nutzer eine in bereinigter Form vorliegende Formel in Pränexform überführen soll, ohne Atome zu modifizieren.

Die bereinigte Form ist Voraussetzung für die Anwendbarkeit bestimmter benötigter Umformungsschritte des Algorithmus zur Pränexform-Bildung. Leider ergibt sich durch das Vorziehen der Bereinigung an den Beginn der Normalformbildung das Problem, dass es unter den weiteren Umformungsschritten einen gibt (das Auflösen eines Äquivalenzjunktors), der die bereinigte Form unter Umständen nicht aufrecht erhält. Betrachten wir z.B. die beiden folgenden semantisch äquivalenten Formeln:

$$R(x) \leftrightarrow (\forall y P(x, y)) \equiv [R(x) \rightarrow (\forall y P(x, y))] \wedge [(\forall y P(x, y)) \rightarrow R(x)]$$

Während die linke Formel noch in bereinigter Form ist, wird in der rechten die Variable y durch zwei Quantoren gebunden. Daher musste obiger Algorithmus um eine Sonderfallbehandlung für die Kontrolle von nicht bereinigten Benutzereingaben erweitert werden. Nach einem solchen Umformungsschritt ist außerdem die Formel neu zu bereinigen, bevor mit anderen Umformungen fortgefahren werden kann. Da die bereinigte Form aufgrund freier Wahl eines neu einzuführenden Variablenbezeichners nicht eindeutig bestimmt ist, der Algorithmus zur Prüfung der Benutzereingabe jedoch aus zwei verschiedenen Formeln jeweils automatisch eine Pränexform herleitet und identische Mengen vorkommender Atome in beiden Formeln voraussetzt, muss für die dabei ggf. auszuführenden Neubereinigungen in beiden Formeln eine identische Bezeichnerwahl bei den Variablenumbenennungen sichergestellt werden.

Der Großteil der Implementierungen zu Aussagen- und Prädikatenlogik ist nicht aufgabenspezifisch. Für die hier vorgestellten Aufgaben stellen wir zwei Pakete „al“ (Aussagenlogik) und „pl“ (Prädikatenlogik) bereit. Darin sind die Datenstrukturen zur Darstellung und Verarbeitung entsprechender Formeln, Routinen zur automatischen Umformung (unter anderem für die Tipp-Berechnung), zu Äquivalenzprüfungen, Normalformprüfungen und nicht zuletzt Parser zur Konvertierung eines Formel-Strings in eine solche Datenstruktur implementiert. Die Aufgabenklassen — auch noch neu zu erstellende — können auf diese Pakete zur Umsetzung ihrer Korrektur- und Tipp-Logik zurückgreifen (vgl. auch Punkt 5. in Abbildung 5).

5 Implementierung

Das vollständig in Java implementierte SKA-System umfasst ein Framework, mit dessen Hilfe sich Aufgaben erstellen und ausführen lassen. Es besteht zum einen aus einem Servlet, das als Front Controller fungiert und allein die Kommunikation mit dem Client-Browser abwickelt, und zum anderen aus einer abstrakten Klasse *AUFGABE*, von der konkrete Aufgabenklassen abzuleiten sind, sowie ein paar Bausteinen (z.B. Eingabezeile, Checkbox, Button) zum Erstellen von Aufgabenformularen.

Eine damit erstellte *Aufgabenklasse* definiert einen generischen *Aufgabentyp*, aus welchem sich durch Übergabe weiterer Parameter (in der vorgestellten Beispielaufgabe die Ausgangsformel) zur Laufzeit eine konkrete *Aufgabe* ableiten lässt. Jede Aufgabenklasse definiert (ggf. in Abhängigkeit von den Parametern) einen Titel, einen Aufgabentext, ein Aufgabenformular sowie Funktionalität zur Auswertung von Benutzereingaben und ggf. zur Ermittlung von Tipps⁵.

Im Folgenden beschreiben wir die Ausführung einer Aufgabe:

Bis auf wenige Ausnahmen⁶ löst jede Aktion des Benutzers — wie das Starten einer Aufgabe, das Anfordern der Auswertung einer Benutzereingabe oder das Anfordern eines Tipps — einen Request an die Webapplikation aus, der vom *SKASERVLET* verarbeitet wird. Das Servlet wertet die Request-Parameter aus und bestimmt damit zunächst die konkrete Aufgabenklasse, auf die sich der Request bezieht. Diese wird daraufhin instanziiert, wobei die Initialisierungs-Parameter aus dem Request (wie z.B. die Ausgangsformel) an den Konstruktor übergeben werden. Die Eingaben des Nutzers — sofern vorhanden — werden in das Aufgabenformular der Aufgabenklasse (genauer: der erzeugten Instanz) eingetragen und stehen dieser so zur Verfügung, woraufhin im Regelfall⁷ die Routine der Aufgabenklasse zur Auswertung der Eingaben bzw. zur Tippermittlung aufgerufen wird. Deren Ausgabe, also Erfolgs- oder Fehlermeldungen bzw. der Tipp-Text, werden an das *SKASERVLET* zurückgegeben und von diesem gepuffert. Zu guter Letzt erzeugt das Servlet die Response, also die neue an den Browser zu übertragende HTML-Seite, wozu es zunächst von der Aufgabenklasse die Aufgabenstellung (bestehend aus Überschrift, Aufgabentext und Formular) generieren lässt und diese zusammen mit der gepufferten Rückmeldung der Auswertungsroutine ausgibt. Abbildung 5 fasst den oben beschriebenen Ablauf eines solchen Requests zusammen.

Die Aufgabenstellung wird erst nach Auswertung der Nutzereingaben bzw. Tippermittlung neu erzeugt, weil auf diese Weise die Korrektur-/Tipp-Logik noch Änderungen einbringen kann. Beispiele für solche Veränderungen am Aufgabenformular in oben beschriebener Beispielaufgabe sind das Anbringen der Markierung bei Anforderung eines Tipps oder

⁵Die Tipp-Funktionalität ist optional, es können auch Aufgaben nur mit reiner Auswertungslogik erstellt werden.

⁶Einfache Aktionen — wie z.B. das Eintragen des Umformungsvorschlags in die Eingabezeile — können clientlokal über JavaScript ausgeführt werden. Das Framework bietet insbesondere einer Aufgabenklasse die Möglichkeit, zusätzlich noch eigenen JavaScript-Code zu definieren.

⁷Ausnahmefälle sind der Start der Aufgabe oder ein per Request angeforderter Neu-Aufbau der Aufgabenseite, wie ihn unsere Normalformaufgaben z.B. beim Ein- und Ausblenden der Liste vorgenommener Umformungen nutzen.

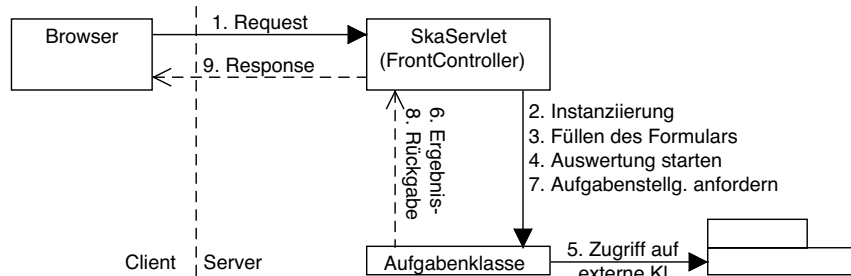


Abbildung 5: Architektur / Request-Ablauf

das Ersetzen der als nächstes umzuformenden Formel durch die letzte Eingabe und das Leeren des Eingabefeldes im Falle der Auswertung einer korrekten Umformung.

6 Verwandte Arbeiten

Ein im Web verfügbares Übungssystem aus dem Bereich der Aussagenlogik ist unter [DVT] zu finden. Es handelt sich dabei um einen so genannten Aussagenlogik-Taschenrechner, der jedoch eine ganz andere Struktur und Zielsetzung hat: Es wird keine spezielle Aufgabe gestellt, sondern der Nutzer kann vielmehr selbst eine beliebige aussagenlogische Formel eingeben und beliebig umformen. Dazu kann er auch eine Teilformel markieren und dann im Menü eine Regel auswählen, welche anschließend auf die markierte Teilformel angewandt wird — sofern möglich. Bei [Slo00] dagegen handelt es sich um ein interaktives Übungssystem zum Resolutionsbeweis der Aussagenlogik, wobei insbesondere eine gegebene aussagenlogische Formel zunächst in KNF zu überführen ist. Das System kann dabei wahlweise die komplette Umformung der Startformel in KNF vorführen, oder der Benutzer kann die Umformung in beliebig vielen Schritten selbst vornehmen und dann seine Eingabe am Ende auswerten lassen. Beides sind Java-Applets, die speziell für ihre spezifische Aufgabe entwickelt wurden und nicht um neue Aufgaben ergänzt werden können.

Mit WEBASSIGN [BHSV99] existiert auch an der FernUniversität schon ein System für Online-Aufgaben. Es ist in erster Linie, wie in der Einleitung schon genauer beschrieben, für Einsendeaufgaben vorgesehen. Die Einsendungen werden gespeichert und dann (manuell oder auch automatisch) bewertet, weshalb jeder Nutzer einen Account besitzen muss. Die Zielsetzung des SKA-Systems ist jedoch im Kontext vom Selbstkontrollaufgaben viel eher auf die interaktive und tutorielle Unterstützung des Studenten ausgerichtet.

7 Zusammenfassung und Ausblick

Unser Ziel war es, interaktive Übungsaufgaben für Fernstudierende anzubieten, die im Zweifel mit Tipps bei der Lösung der Aufgaben behilflich sind und damit das Bearbeiten bestimmter Aufgabentypen trainieren, welche den Studierenden im ersten Semester oft Probleme bereiten. Um zu den ersten umgesetzten Aufgaben jederzeit weitere hinzufügen zu können, wurde ein Framework zur Erstellung und Ausführung solcher Aufgaben realisiert. Das Gesamtsystem mit den implementierten Aufgaben ist bereits im Einsatz und läuft stabil und ohne nennenswerten Administrationsaufwand.

Für die Zukunft ist insbesondere das Erstellen weiterer Aufgabentypen geplant. So kann sich an die Bildung der Pränexform einer prädikatenlogischen Formel direkt eine Aufgabe anschließen, die Formel nun in Skolemische Normalform zu bringen; darauf aufbauend können Aufgaben zum Resolutionskalkül (vgl. [Sch00, Wal03]) angeboten werden.

Das Framework selbst kann bei Bedarf ebenfalls erweitert werden, z.B. um neue Formulalemente wie mehrzeilige Textfelder. Falls neben der Webapplikation auch eine Desktop-Applikation angeboten werden soll, wäre es denkbar, eine Alternative zum SKASERVLET zu entwickeln, z.B. eine Java-Applikation, welche die Aufgaben ebenfalls ausführen kann.

Literatur

- [BH03] C. Beierle und H. Helbig. *Kurs 01601: Formale Grundlagen der Informatik*. FernUniversität in Hagen, 2003.
- [BHSV99] J. Brunsmann, A. Homrighausen, H.-W. Six und J. Voss. Assignments in a Virtual University - The WebAssign System. In *Proceedings of the 19th World Conference on Open Learning and Distance Education*, Wien, Juni 1999.
- [CBN89] A. Collins, J. S. Brown und S. E. Newman. Cognitive Apprenticeship. Teaching the Crafts of Reading, Writing, and Mathematics. In *Knowing, learning, and instruction*, Seiten 453–494. Lawrence Erlbaum Associates, 1989.
- [DVT] Lehrgebiet Datenverarbeitungstechnik, FernUniversität in Hagen. Aussagenlogik-Taschenrechner.
<http://www.fernuni-hagen.de/DVT/Java/LogikRechner/LogikRechner.html>.
- [Sch00] U. Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, 5. Auflage, 2000.
- [SG04] I. Schulz-Gerlach. Ein interaktives WWW-basiertes Übungssystem mit Aufgaben zu Aussagen- und Prädikatenlogik. Diplomarbeit, FernUniversität in Hagen, September 2004.
- [Slo00] O. Sloboda. Interaktive Einführung in den Resolutionsbeweis der Aussagenlogik. Diplomarbeit, Fachhochschule Dortmund, 2000.
<http://kik.informatik.fh-dortmund.de/Diplomarbeiten/DA%20Sloboda/HTML-%20Kurs/Start.htm>.
- [Wal03] C. Walther. Automatisches Beweisen. In G. Görz, C.-R. Rollinger und J. Schneeberger, Hrsg., *Handbuch der Künstlichen Intelligenz*. Oldenbourg, 4., korr. Auflage, 2003.