

Skills und Konzepte als Grundlage für Benutzermodellierung in einem *Prolog*-ITS

Christoph Peylo und Tobias Thelen

Institut für Semantische Informationsverarbeitung

Universität Osnabrück

D-49069 Osnabrück

{christoph.peylo|tobias.thelen}@uni-osnabrueck.de

Zusammenfassung

Zunächst wird das an der Universität Osnabrück im Rahmen des Projekts „Virtueller Campus“ entwickelte Internet-basierte *Prolog-Tutor-System* vorgestellt, das im Rahmen von Lehrveranstaltungen zum logischen Programmieren eingesetzt wird. Dabei wird sowohl auf die technische Realisierung des Systems, als auch auf das Einsatzgebiet (*Prolog*) näher eingegangen. Danach wird zunächst allgemein auf Benutzermodellierung im Bereich von Lehr- und Lernsystemen eingegangen und die dabei verwendeten Technologien diskutiert. Dabei wird argumentiert, dass auf Ontologien und Wissensstrukturen basierende Ansätze besser zur Modellierung von Benutzerwissen geeignet sind als Bayessche Netze. Schließlich wird das Vorgehen bei der Benutzermodellierung im *VC-Prolog-Tutor* vorgestellt und an Beispielen illustriert. Im letzten Abschnitt wird kurz auf die gegenwärtig noch zu leistende Arbeit eingegangen.

Keywords

ITS, Adaptivität, User Modelling, Ontologie, Prolog

1 Das Projekt „Virtueller Campus“

Der Projektverbund „Virtueller Campus Hannover-Hildesheim-Osnabrück“ verfolgt das Ziel, in einem praktischen Anwendungskontext Modelle für die Integration virtueller/multimedial unterstützter Lehrformen an Hochschulen zu entwickeln. Zu den didaktischen Modellen für Lehr-Lern-Formen werden Lösungen erprobt, die den durch i Multimedia-Innovationen in der Hochschullehre ausgelösten organisatorischen Herausforderungen Rechnung tragen. Beteiligte Projektpartner sind das Institut für Technische Informatik, Abteilung Rechnergestützte Wissensverarbeitung der Universität Hannover, das Institut für Angewandte Sprachwissenschaft und das Zentrum für Fernstudium und Weiterbildung der Universität Hildesheim sowie das Institut für Semantische Informationsverarbeitung der Universität Osnabrück.

Im Osnabrücker Teilprojekt wird ein Internet-basiertes, interaktives *Prolog*-Tutorsystem entwickelt, das im Rahmen von Lehrveranstaltungen zum logischen Programmieren in *Prolog* eingesetzt wird.

2 Der VC-Prolog-Tutor

2.1 Prolog und Beschreibung der Prolog-Kurse am ISIV

Prolog ist eine relativ junge Programmiersprache, die Anfang der 80er Jahre entwickelt wurde, um Probleme der maschinellen Verarbeitung natürlicher Sprache theoretisch fundiert behandeln zu können. Dazu wurde auf ein Verfahren aus dem Bereich des *automatischen Beweisens* zurückgegriffen, die SLD-Resolution über Horn-Klauseln. Im Kern ist ein *Prolog*-Interpreter also ein automatischer Beweiser für eine Teilmenge der Prädikatenlogik erster Stufe. Dadurch unterscheidet *Prolog* sich grundsätzlich von anderen, prozeduralen, funktionalen oder objektorientierten Programmiersprachen: Im Vordergrund steht die deklarative Beschreibung eines Problems im Gegensatz zur expliziten Spezifikation des Lösungsweges.

Im Rahmen der Studiengänge *Computerlinguistik und Künstliche Intelligenz* sowie *Cognitive Science* ist ein regelmäßig durchgeführter *Prolog*-Kurs zentraler Bestandteil des Grundstudiums. Dabei soll nicht lediglich die Fähigkeit vermittelt werden, syntaktisch korrekte *Prolog*-Programme zu verfassen, sondern ebenfalls eine Reihe von Techniken, die notwendig sind, um die Struktur von Problemen abzubilden.

2.2 Die technische Seite des Systems

Der *VC-Prolog-Tutor* ist als Client-Server-System konzipiert, das clientseitig lediglich einen Java-fähigen Browser voraussetzt. Der Benutzer des Systems ist nicht gezwungen, zusätzliche Software, die zudem für unterschiedliche Plattformen zur Verfügung gestellt werden müsste, auf seinem Rechner zu installieren. Vorteilhaft ist dieses Vorgehen, wenn der Benutzer, in der Regel Studienanfänger, mit seinem System noch nicht so vertraut ist, dass die fehlerfreie Installation von Zusatzsoftware erwartet werden könnte; außerdem in Umgebungen, die zentral verwaltet werden und in denen einzelne Benutzer nicht die Möglichkeit haben, Software zu installieren und schließlich wird Interessierten ein möglichst leichter Zugang zu Testzwecken ermöglicht.

Der bislang entwickelte und im Einsatz befindliche Prototyp soll als Rahmen für Internet-Lernumgebungen dienen, der auch für weiterführende Veranstaltungen (im Rahmen des Kurses *Methoden der KI* sind Übungsaufgaben mit Hilfe von *Prolog* zu lösen), zur Vermittlung anderer Programmiersprachen (eine experimentelle Anbindung zu *Lisp* besteht bereits) oder in weiter entfernten Kontexten, wie der Entwicklung von Grammatiken zur Verarbeitung natürlicher Sprache, Verwendung finden kann.

Auf Server-Seite werden in einer Datenbank sämtliche Benutzerdaten verwaltet, wie Kurszugehörigkeit und Zugriffsrechte, Bereiche für abgelegte Dateien, aktuelle Übungsaufgaben, abgegebene Lösungen dazu etc. Das System bietet serverseitig eine einheitliche Schnittstelle für Kursteilnehmer und Tutoren, die die Benutzerverwaltung übernehmen, Übungsaufgaben und Musterlösungen ablegen, sowie die abgegebenen Lösungen abrufen, automatisiert testen lassen und kommentieren können. Im Server soll soviel der Funktionalität des Systems wie möglich untergebracht sein, insbesondere wird der verwendete Interpreter, Compiler o.ä. vom Server gestartet und angesprochen, d.h. clientseitig ist kein über das allgemeine Client-Server-Protokoll hinausgehendes Wissen über die vom Benutzer erstellten und an den Server übermittelten Inhalte nötig.

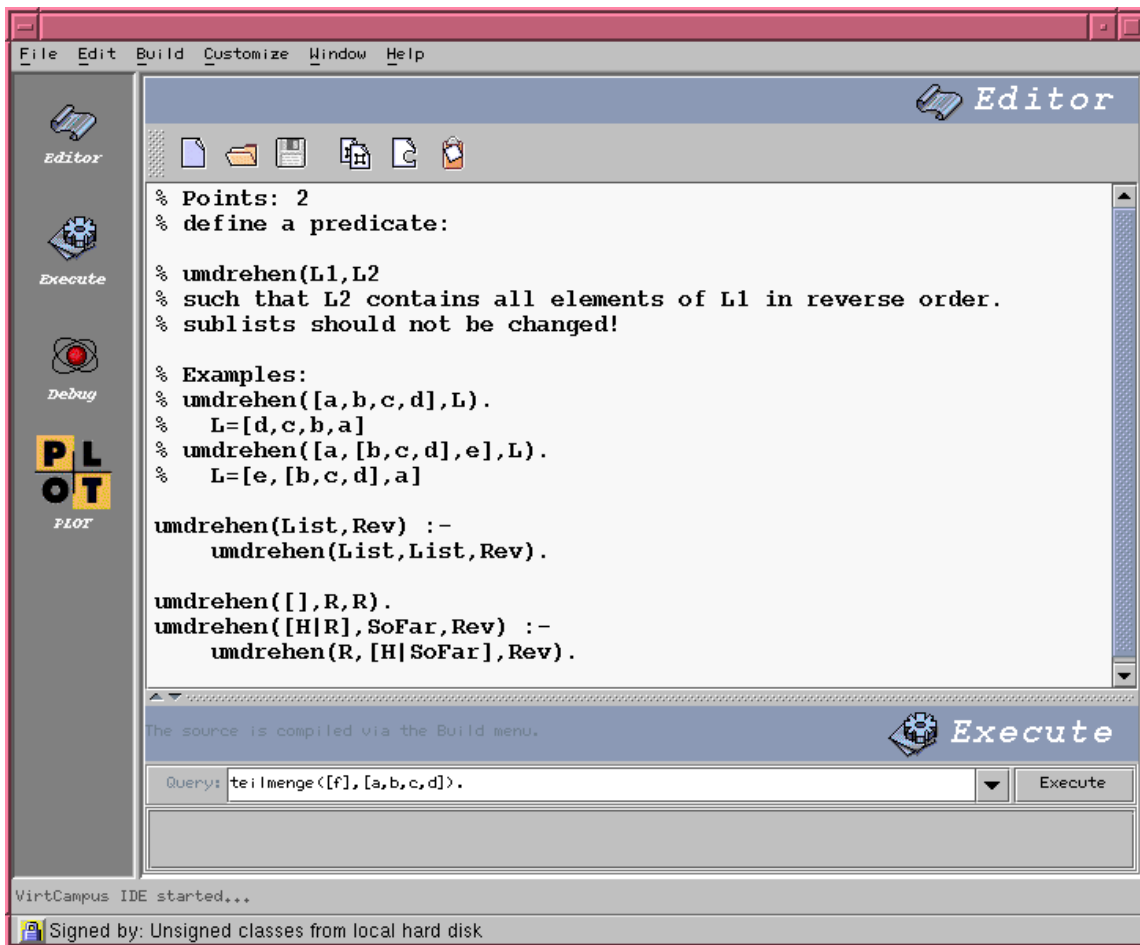


Abbildung 1: Screenshot des VC-Prolog-Tutor Applets

Clientseitig kommt ein Java-Applet zum Einsatz, das von seiner Erscheinung und seinem Funktionsumfang her als Integrierte Entwicklungsumgebung (IDE) gestaltet ist (s. Abbildung 1). Der Benutzer kann Dateien und Übungstemplates aufrufen, editieren und ablegen und Bereiche seines Dateibaums anderen Mitgliedern seiner Arbeitsgruppe zugänglich machen. Die Bearbeitung und Abgabe von Übungsaufgaben erfolgt vollständig über den VC-Prolog-Tutor, nach der Korrektur durch einen menschlichen Tutor können Kommentare und Bewertungen auf die gleiche Weise abgerufen werden. Der Prolog-spezifische Teil der Entwicklungsumgebung ermöglicht es dem Benutzer, das erstellte Programm an den serverseitig laufenden Prolog-Interpreter zu schicken, Anfragen zu stellen und so das Programm zu testen. Weitergehende Unterstützung erfährt er einerseits durch einen Debug-Modus, die *Visual Trace*, die im Rahmen des Studienprojektes PLOT (Prolog lernen mit dem Online-Tutor) (Bönnen et al., 1999) entwickelt wurde, andererseits durch ein Fehleranalysemodul, das weit über die im Prolog-Interpreter vorhandenen Analysemöglichkeiten hinausgeht. Das Fehleranalysemodul ist ebenfalls im Rahmen von PLOT entstanden.

Der VC-Prolog-Tutor bietet eine enge Anbindung zu weiteren Materialien, wie einem allgemeinen oder kurspezifischen Skript, einem technischen Manual zur verwendeten Programmiersprache, weiteren Hinweisen zu den Übungsaufgaben etc. Da die Entwicklungsumgebung als Java-Applet aus den Web-Seiten des betreffenden Kurses heraus gestartet wird, ergibt sich die Inte-

gration von *Prolog*-Tutor und HTML-Materialien aus Benutzersicht auf eine natürliche Weise.

2.3 Fehleranalyse

Die Fehleranalysekomponente *PLOT* ist in der Lage, eine Reihe von typischen Fehlern, gerade auch solchen, die Anfängern häufig unterlaufen, zu erkennen und dem Benutzer eine für ihn nachvollziehbare Erklärung zu liefern. Dabei werden einige Einschränkungen vorgenommen, sowohl hinsichtlich des verarbeitbaren *Prolog*-Fragmentes als auch hinsichtlich der Vorannahmen über die Semantik des Programms, d.h. für tiefergehende Teile der Analyse ist eine Zuordnung zu einer Aufgabe bzw. einer Menge von Musterlösungen notwendig.

Die Analyse wird von drei Komponenten vorgenommen, die sukzessive auf vorherige Ergebnisse zugreifen:

Syntaxanalyse: Die Syntaxanalyse kann nur ein *Prolog*-Fragment verarbeiten, das keine Cuts („!“), Oder-Verknüpfungen („;“), selbstdefinierte Operatoren oder Funktoren an Argumentposition enthält. Diese Einschränkungen sind für den anvisierten Zweck, typische Anfängerfehler abzufangen, hinnehmbar. Vorteil dieser relativ strengen Annahmen über die verwendeten Sprachkonstrukte ist die Möglichkeit, Syntaxfehler auch stillschweigend korrigieren und nach weiteren Fehlern suchen zu können. Ein sehr häufiger Fehler sind vergessene Klammern, für die der *Prolog*-Interpreter zwar Fehlermeldungen generiert, aber nicht in der Lage ist, die eigentliche Fehlerposition nachvollziehbar zu bestimmen. Falls möglich, fügt die Syntaxanalyse z.B. vergessene Klammern mit Hilfe von Fehlerregeln ein.

Semantikcheck: Im allgemeinen Teil des Semantikchecks wird überprüft, ob jedes verwendete Goal auch tatsächlich definiert ist, im speziellen Teil wird anhand von Beispielanfragen überprüft, ob der Beweis eine heuristisch bestimmte Rekursionstiefe übersteigt, also mit einiger Wahrscheinlichkeit auf eine Endlosschleife geschlossen werden kann, und ob für diese Beispielanfragen das Prädikat die richtigen Ergebnisse liefert. Dafür muss bekannt sein, was das Prädikat im Erfolgsfalle tun soll, d.h. zu welcher dem System bekannten Aufgabe es eine Lösung darstellen soll. Für alle Aufgaben erstellt ein menschlicher Tutor Musteranfragen, die samt Ergebnissen in einer Datenbank abgelegt werden.

Strukturcheck: Schließlich wird das zu überprüfende Programm mit einer Reihe eingetragener Musterlösungen strukturell verglichen und auf Inkonsistenzen im Layout überprüft. Der strukturelle Vergleich mit Musterlösungen kann unterschiedliche Variablenbenennungen, andere Argument- und Klauselreihenfolgen etc. erkennen und „Abstände“ zu Musterlösungen berechnen. Jede Musterlösung kann mit einer Bewertung versehen werden, wie „optimale Lösung“ oder „umständliche Lösung, da Konzept X statt Y verwendet“. Diese Bewertung wird dem Benutzer zusätzlich zu anderen gefundenen Fehlern mitgeteilt.

3 Benutzermodellierung

3.1 Wozu Benutzermodellierung?

Im Rahmen von netzbasierten Lehr- und Lernsystemen ist Benutzermodellierung vor allem im Hinblick auf zwei Aspekte interessant: Problemlöseunterstützung und Adaptivität (Self, 1974), (Self, 1994). Dabei kann Adaptivität in adaptiver Navigationsunterstützung oder adaptiver Präsentation (oder aus Elementen von beiden) bestehen (Brusilovsky, 1999), (Kay und Kummerfeld, 1997), (Bra und Calvi, 1999).

Studenten kann aufgrund eines Benutzermodells detailliertere Hilfe bei der Problembewältigung gegeben werden. So können durch individuelle Leselisten (*sequencing*) konzeptuelle Defizite beseitigt oder für den Kenntnisstand der Lernenden geeignete Übungsaufgaben ausgewählt werden, um ein optimales Fortschreiten im Kurs zu erleichtern (McCalla, 1994).

Für den Tutor eines Kurses ergeben sich durch den Vergleich verschiedener Benutzermodelle eines Kurses wertvolle Rückschlüsse auf das Leistungsniveau eines Kurses und von einzelnen Teilnehmern Kurses (*class monitoring*). Die dabei verwendete Technologie kann auch für die Zusammenstellung von Arbeitsgruppen (Hoppe, 1995), (Ikeda et al., 1997) oder zum Auffinden eines geeigneten Ansprechpartners für eine Problemlösung (McCalla et al., 1997) eingesetzt werden.

Obwohl die Benutzermodellierung gerade vom Standpunkt des Tutors interessant und wünschenswert ist, ist dabei zu bedenken, dass es sich dabei um vertrauliche und persönliche Daten über Individuen handelt, die einen sensiblen Umgang erfordern. Benutzermodellierung sollte daher vorwiegend im Dienst der Benutzerunterstützung eingesetzt werden, so dass der Nutzen für den Lernenden größer als der Eingriff in die Persönlichkeitssphäre ist.

3.2 Technologien zur Benutzermodellierung

Im Kontext von Lehr- und Lernumgebungen soll ein Lernermodell in der Lage sein, das aktuelle Wissen des Lernenden zu jedem Zeitpunkt des Programmdurchlaufs definieren zu können (Winkels und Breuker, 1992), (Schulmeister, 1997). Dabei wird das Wissen des Lernenden entweder als Teilmenge des Expertenmodells abgebildet (*subset* oder *overlay modell*) oder als Abweichung vom Expertenwissen ermittelt (*deviation modell*). Die diagnostischen Fähigkeiten beider Systeme sind allerdings rein inhaltsorientiert und nicht psychologisch, können also nicht das Lernverhalten erklären (Schulmeister, 1997, S. 184).

3.3 Bayessche Netze

Häufig wird die zu erlernende Domäne durch ein Bayessches Netz repräsentiert, wobei die Knoten Konzepte bzw. Lehrinhalte und die Kanten Verbindungen zwischen diesen darstellen. Verbindungen können einen thematischen Zusammenhang bedeuten oder curriculare Abhängigkeiten repräsentieren. Für die einzelnen Verbindungen gibt es somit meist unterschiedliche semantische Interpretationen. Die Übergänge zwischen den Kanten sind mit Wahrscheinlichkeiten gewichtet. Damit wird die Unsicherheit, wie die Kenntnis von Konzepten bzw. Lehreinheiten

„wirklich“ zusammenhängen durch Wahrscheinlichkeiten ausgedrückt. Bei Kenntnis der abgearbeiteten Lehreinheiten kann die Wahrscheinlichkeit angegeben werden, welche Konzepte der Benutzer kennt, bzw. was er als nächstes bearbeiten können sollte. Dieses Vorgehen hat den Vorteil, dass es relativ unaufwendig ist, da das Wissen des Benutzers direkt auf das Domänenmodell abgebildet wird.

Trotz dieser Vorteile ist es vom Konzeptionellen her in mehrfacher Hinsicht zu hinterfragen:

- Fachliche Domänen sind (in aller Regel) strukturiert (Edelmann, 1996) und die Kohärenz der Terminologie bzw. der Konzepte einer Domäne ist durch Konventionen (Kuhn, 1991), (Putnam, 1990) und nicht durch Wahrscheinlichkeiten bestimmt. Somit erscheint eine Repräsentation der Domäne durch ein Bayessches Netz als fragwürdig.
- Der Unsicherheitsfaktor in den Übergängen zwischen den Knoten des Konzeptgraphen, besteht im wesentlichen darin, dass in den Systemen meist kein explizites Kriterium für das Verständnis des Gelesenen existiert. Eine weitere Schwierigkeit resultiert daraus, dass Unterricht letztendlich auf eine praktische Umsetzung hinzielt. Der Umsetzungsprozess von theoretischem Wissen in den meisten Systemen aber nicht explizit modelliert wird. Eine Vorlesung über eine Programmiersprache wird sich zwar auf die Vermittlung der theoretischen Grundlagen und Konzepte konzentrieren, intendiert ist aber die praktische Umsetzung des erworbenen theoretischen Wissens.
- Noch problematischer wird es, wenn wir *offene* Lernumgebungen im Internet berücksichtigen. Wenn auch von vielen Autoren nur intendiert ist, den eigenen Kurs für möglichst viele Benutzer zu öffnen, so machen internetbasierte Lernumgebungen tatsächlich nur dann Sinn, wenn sie offen sind, d.h. es Studenten zumindest ermöglichen, nach thematisch verwandten Ressourcen im Netz zu suchen und ihr extern erworbenes Wissen im Kontext des Kurses anwenden können. In einem solchen Fall läßt sich die Lernaktivität des Studenten nicht ohne weiteres im eigenen System (z.B. durch die Auswertung von Logdateien) verfolgen, d.h. hier müssen andere Wege gefunden werden, um die externe, eigenständige Lernaktivität nachweisen und in das Benutzermodell integrieren zu können.

Um ein Benutzermodell im Sinne von Adaptivität des Systems an die Erfordernisse des Benutzers erreichen zu können, sollte unseres Erachtens primär der Versuch unternommen werden, durch genauere Modellierung des Lern- und Verstehensprozesses (vgl. Self (1992)) die Unsicherheiten zu beseitigen, als diese in das Modell einzubetten.

3.4 Wissensstrukturen

Eine strukturierte Organisation des Domänenwissens ist nicht nur aus didaktischen Erwägungen her sinnvoll, um die Beziehung der Konzepte zu veranschaulichen (Edelmann, 1996), sondern es ist eine notwendige Voraussetzung um den Kenntnisstandes des Benutzers auf inhaltlicher Ebene modellieren zu können und die Kommunizierbarkeit von Wissen zwischen System und Benutzer möglich zu machen (vgl. Andriessen und Sandberg (1999)). Hinsichtlich der Strukturierung des Wissens im Bereich von Lehr- und Lernsystemen bietet der Ansatz der *Knowledge Spaces* (Falmagne et al., 1990), (Dütsch und Gediga, 1995) interessante Möglichkeiten. Danach lassen sich, basierend auf der Kenntnis, welche Aufgaben ein Lerner zu lösen im Stande war, Wissensstände und Wissensstrukturen modellieren:

Definition 1 (Knowledge State) Der Wissensstand (knowledge state) einer Person wird durch die Probleme bestimmt, die sie lösen kann:

$$K_S := \{p \in P : \langle s, p \rangle \in R\}$$

Wobei U eine Menge von Personen (mit $s \in U$), P eine Menge von Problemen ist und $R \subseteq U \times P$, so dass $\langle s, p \rangle \in R$, gdw. s Problem p lösen kann.

Definition 2 (Knowledge Structure) Eine Wissensstruktur, z.B. einer Klasse oder eines Kurses, ist eine Sammlung von Kenntnisständen:

$$K := \{K_S : S \in U\} \cup \{\emptyset, P\}$$

Wobei U eine Menge von Personen (mit $S \in U$) und P eine Menge von Problemen bzw. Aufgaben ist.

Vielversprechend erscheint es, diese Ansätze mit den gegenwärtigen Versuchen, Ontologien für den Bereich intelligenter Lehr- und Lernsysteme zu nutzen (Ikeda et al., 1999), (Mizoguchi und Bourdeau, 1999), (Peylo et al., 2000) zusammenzuführen.

4 Benutzermodellierung im VC-Prolog-Tutor

Gegenwärtig können im VC-Prolog-Tutor zwei Quellen zur Beurteilung eines Benutzers ausgewertet werden: die Ergebnisse der Fehlerkorrektur (PLOT) und die serverseitigen Logdateien. Beim einloggen eines Benutzers in das System können über die Vergabe von cookies Aktivitäten im System (Konsultieren von Skript und Manualseiten etc.) einem konkreten Benutzer zugeordnet werden. Basierend auf den Arbeiten von Falmagne et al. (1990), Doignon und Falmagne (1999) und Düntsch und Gediga (1995) lassen sich damit Wissensstände und Wissensstrukturen modellieren (siehe Tabelle 1 und Abbildung 2).

Student	Aufgabe1	Aufgabe2	Aufgabe3	Aufgabe4	Aufgabe5	Aufgabe6	Aufgabe7
Student1	ja	ja	ja	ja	ja	ja	ja
Student2	ja	ja	ja	ja	ja	nein	nein
Student3	ja	ja	nein	ja	nein	ja	nein
Student4	ja	nein	ja	ja	nein	ja	ja
Student5	ja	ja	ja	nein	ja	nein	nein
Student6	ja	ja	nein	ja	ja	nein	nein
Student7	ja	nein	ja	nein	ja	ja	ja

Tabelle 1: Beziehung zwischen Lernenden und Übungsaufgaben

Die in Tabelle 1 enthaltenen Relationen zwischen Studenten und gelösten Aufgaben, die durch die Beispielfragen im Semantikcheck (siehe 2.3 auf Seite 4) ermittelt werden, lassen sich in einem Verband darstellen, der als Wissensstruktur betrachtet werden kann. Dabei lassen sich

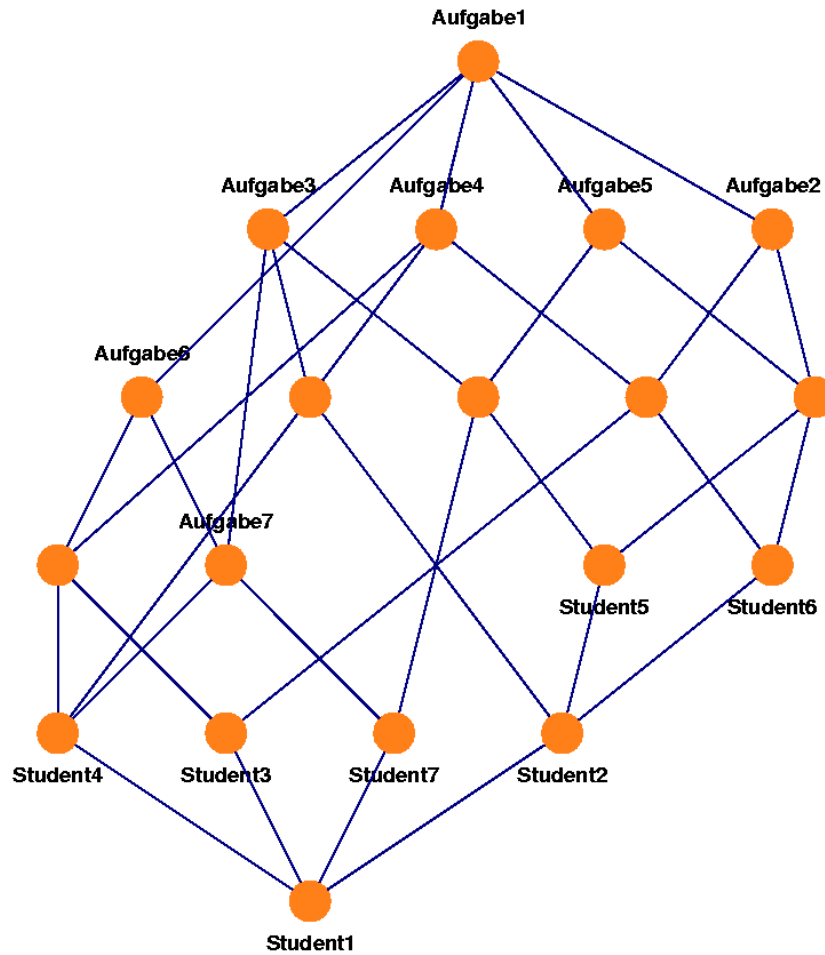


Abbildung 2: Obwohl der Verband nur auf der Relation zwischen Aufgaben und Studenten basiert, lassen sich unterschiedliche semantische Interpretationen an die Kanten anlegen: absteigende Kanten zwischen Aufgaben können als „schwerer-als“ interpretiert werden, absteigende Kanten zwischen Lernern als „besser-als“ und absteigende Kanten zwischen Aufgaben und Lernern bedeuten, dass der Lernende die entsprechende Aufgabe lösen konnte.

unterschiedliche semantische Interpretationen auf die Kanten im Graphen anwenden, je nachdem ob sie Aufgaben, Lerner oder Lerner und Aufgaben verbinden (vgl. Abbildung 2). Dieses Vorgehen erlaubt es, den Wissensstand von Benutzern zu vergleichen und das Leistungsniveau eines Lernenden zu erkennen. Um einem Lernenden detailliertere Hilfestellung bei der Lösung von Problemen zu geben und Fehlkonzeptionen entgegenzuwirken, ist diese Granularitätsebene jedoch noch zu grob.

4.1 Die Verbindung von Ontologien und Wissensstrukturen

Um einem Studenten detailliertere Unterstützung zukommen lassen zu können, muss die inhärente Struktur der Domäne dem System bekannt sein, wie bereits dargelegt wurde. Die Bereitstellung des relevanten Wissens eines Bereichs erfolgt daher im *VC-Prolog-Tutor* durch eine umfassende Modellierung der involvierten Konzepte und Domänen in einer als Begriffsverband repräsentierten Ontologie (vgl. (Klose et al., 1992), Pirlein (1995), Guarino (1998) Sowa (1999)), in der sowohl theoretische Konzepte, als auch Fähigkeiten repräsentiert werden. Dabei müssen alle Konzepte, die für die Domäne – in diesem Fall *Prolog* – relevant sind, in der Ontologie modelliert sein. So identifizierten du Boulay und Sothcott (1987) drei Felder, die für das Erlernen einer Programmiersprache relevant sind:

Syntax und Semantik Kenntnis der Syntax bezieht sich auf die Regeln, die korrekten Programmcode festlegen. Obwohl dies für Anfänger die erste Hürde ist, liegt hierin nicht der zentrale Aspekt im Lernen einer Programmiersprache. Wissen über die Semantik einer Programmiersprache beinhaltet Wissen über die Datenstrukturen und -typen der jeweiligen Sprache und Kontrollstrukturen sowie die Art und Weise, wie der Programmcode verarbeitet wird. Dieses Wissen ermöglicht die Vorhersage, welche Resultate ein Codefragment haben wird. Dies wird als die prozeduralen und deklarativen Aspekte der Semantik von Programmiersprachen bezeichnet (du Boulay und Sothcott, 1987).

Taktik und Strategie Dieser Aspekt umfaßt Wissen wie man ein Problem in Teilprobleme zerlegen und mit den gegebenen Sprachkonstrukten analysieren und bearbeiten kann. Dieses Wissen schlägt sich im Programmdesign nieder und manifestiert sich in der Auswahl der von der Sprache zur Verfügung gestellten Daten- und Kontrollstrukturen, um ein gegebenes Problem möglichst effizient zu lösen.

Pragmatisches Wissen Dieses umfasst Wissen über den Umgang mit der Programmierumgebung, über Programm testen und (gegebenenfalls) *debugging*, den Umgang mit Editor und Compiler etc.

Diese Bereiche umfassen also nicht nur die Kenntnis theoretischer Konzepte, sondern auch ihre praktische Umsetzung in so genannten *skills*.

Definition 3 (Skills) *S* sei eine Menge von Fähigkeiten, *P* eine Menge von Problemen. Eine SKILL-Funktion $f(p) \rightarrow 2^{2^{|S|}}$, mit $p \in P$ ist eine Abbildung in die Menge der SKILLS, die zur Lösung von p benötigt werden. Hierdurch läßt sich auch eine Abhängigkeit von Aufgaben definieren. Wenn $f(p) \subseteq f(q)$, mit $p, q \in P$, bedeutet dies, dass q schwieriger als p ist.

Eine Ontologie in diesem Sinn geht also deutlich über eine reine Hierarchie theoretischer Konzepte hinaus.

4.2 Repräsentation von Handlungen

Um unser Vorgehen zu illustrieren, ist in Abbildung 3 der Zusammenhang verschiedener Elemente taktischen Wissens bei der Lösung von *Prolog*-Programmieraufgaben dargestellt.¹ Die



Abbildung 3: Zusammenhang von taktischem Wissen, das zur Lösung von *Prolog*-Programmieraufgaben benötigt wird. Die durchgezogenen Pfeile repräsentieren die *has-part*-Relation (als inverse Relation zu *part-of*). Die gestrichelten Pfeile symbolisieren die *is-a*-Relation.

Abhängigkeit von Handlungen zueinander kann formal durch Grammatiken repräsentiert werden. Da die Regeln einer Grammatik eine bestimmte Reihenfolge der Terminalsymbole vorgeben, können auf diese Weise implizit sequentielle bzw. temporale Aspekte modelliert werden, ohne eine explizite Zeit-Ontologie erforderlich zu machen. Auch läßt sich dadurch die Abhängigkeit von Handlungselementen zueinander darstellen und eine gewisse Hierarchie von Handlungen erstellen. Die theoretische Kenntnis einer Handlungsregel wird durch die nicht-terminalen Symbole der Ersetzungsregel repräsentiert. Dadurch wird die Kenntnis der Abläufe einer Handlung für das theoretische Konzept als konstitutiv bestimmt. Für die praktische Umsetzung kommt zusätzlich die Kenntnis der Terminalsymbole hinzu, d.h. das Vermögen die Regel praktisch umzusetzen. Auf diese Weise kann auch auf formaler Ebene zwischen theoretischem Wissen und der Fähigkeit, dieses umzusetzen, unterschieden werden. Eine Unterscheidung, die in den wenigsten Systemen explizit repräsentiert werden kann. In Abbildung 4 wird dies durch den Ausschnitt einer *XML*-Grammatik illustriert.

Ein Beispiel soll dies verdeutlichen: Aus anderen Programmiersprachen bekannte Konstruktionen wie Verzweigungen, Speichern von Werten in Variablen oder Programm-Schleifen werden in *Prolog* nicht durch spezielle Syntax ausgedrückt, sondern durch *Muster*, deren Anwendung und Erkennung besonders geübt werden muss. An dieser Stelle wird auch schnell detaillierteres Wissen über die Vorgehensweise eines *Prolog*-Interpreters notwendig: Ist die Reihenfolge

¹Unter taktischem Wissen verstehen wir die Umsetzung von Programmier-techniken und -strategien bei der Bearbeitung einer konkreten Aufgabe. Strategie ist demgegenüber eher abstrakt und beschreibt Vorgehen auf einer Ebene, die über konkreten Aufgaben abstrahiert.

```

...
<!ELEMENT RECSKILL (RULE*,BASECASE,RECCASE) >
  <!ATTLIST RECSKILL id ID #IMPLIED>

  <!ELEMENT BASECASE (FACT+,RULE*) >

  <!ELEMENT RECCASE (AKKUMULAT|RULE) >

  <!ELEMENT RULE (HEAD,BODY) >

  <!ELEMENT HEAD (ARGS) >

  <!ELEMENT BODY (SUBGOAL+) >

  <!ELEMENT SUBGOAL (ARGS) >
  <!ATTLIST SUBGOAL
    id ID #IMPLIED
    ident CDATA #REQUIRED
    skill CDATA #IMPLIED
    arity CDATA #IMPLIED>

  <!ELEMENT FACT (HEAD) >
  <!ATTLIST FACT
    id ID #IMPLIED
    ident CDATA #REQUIRED
    arity CDATA #REQUIRED>

  <!ELEMENT ARGS (LIST|VAR|CONST|NUMBER|OPERAT) * >
  <!ATTLIST ARGS skill CDATA #REQUIRED>
...

```

Abbildung 4: Das Beispiel zeigt die taktischen Kenntnisse, die zur Programmierung eines einfachen rekursiven Prädikats benötigt werden.

der Fakten und Regeln im Sinne einer logischen, deklarativen Spezifikation eines Programms irrelevant, so spielt sie für ein konkretes *Prolog*-Programm eine entscheidende Rolle.

Iterative Konstrukte anderer Programmiersprachen werden in *Prolog* meistens durch *Rekursion* ersetzt. In Abbildung 3 ist ein Taktik-Knoten „rekursion“ vorhanden, der mit der in Abbildung 4 angegebenen Grammatik korrespondiert.

Aus der Analyse der korrekten Lösung für das *append*-Problem (s. Abbildung 5) wird die Information gewonnen, dass zur Lösung des Problems auf *diese* Art das Konzept „Rekursion“ verwendet werden muss, dessen Umsetzung in Handlungen durch `recskill` beschrieben ist. Eine Analyse des fehlerhaften Programms in Abbildung 6 mit dieser Grammatik kommt zu dem Ergebnis, dass eine als *base_case* analysierbare Klausel nicht vorhanden ist. Damit ist klar, dass die richtige Taktik gewählt, bei der Umsetzung aber ein notwendiger Teil ausgelassen wurde. Neben die Vermittlung und Analyse der Fähigkeit, syntaktisch wohlgeformte *Prolog*-Programme zu schreiben, muss die Vermittlung und Analyse der Beherrschung von *Mustern* treten, die nicht auf der Ebene von Schlüsselwörtern, sondern von Konzepten und deren Umsetzung in Programmstrukturen liegen.

Unseres Erachtens ist für ein Benutzermodell, das auf Adaptivität und Problemlöseunterstützung des Systems auf inhaltlicher Ebene abzielt, die Repräsentation des Domänenwissens und der Aktionen, die der Benutzer über der Domäne ausführen können soll, die Schlüsseltechnologie. Ähnliche Ansätze wurden für Teilprobleme von *Prolog* auch von Gregg-Harrison (1991) und

```
append ( [], L, L) .
```

```
append ( [H|Rest] , L, [H|RestAppend] ) :-  
    append (Rest , L, RestAppend) .
```

Abbildung 5: Musterlösung: append

```
append ( [H|Rest] , L, [H|RestAppend] ) :-  
    append (Rest , L, RestAppend) .
```

Abbildung 6: Fehlerhaftes Beispielprogramm: Kein Rekursionsabbruch

Hong (1997) verfolgt. Allerdings wurde in den genannten Ansätzen nicht versucht, die identifizierten Handlungsschemata und Grammatiken in eine Domänenontologie zu integrieren.

5 Weitere Arbeiten

Die Modellierung der *Prolog*-Domäne durch eine Ontologie, die die für das Erlernen der Programmiersprache relevanten Aspekte umfaßt, soll in den nächsten Monaten abgeschlossen sein. Bis zum Ende des Jahres sollen die Benutzermodellierungskomponente und Problemlöseunterstützung des Systems erweitert sein, damit sie auf der dann verfügbaren Granularitätsebene den Benutzer entsprechend unterstützen können.

Literatur

- ANDRIESSEN, J. UND SANDBERG, J. (1999): "Where is Education Heading and How About AI?" *International Journal of Artificial Intelligence in Education* 10: S. 130 – 150.
- BÖNNEN, C.; GERSTENHÖFER, J.; HARTMANN, N.; HOLZWARTH, A.; LENSCHOW, K. KOBBER N. UND MEISSNER, B. (1999): "Plot: ProLog Online Tutor". Technischer Bericht, Institut für Semantische Informationsverarbeitung.
- BRA, P. DE UND CALVI, L. (1999): "Aha! An open Adaptive Hypermedia Architecture". *The new review of Hypermedia and Multimedia* 4: S. 115 – 139.
- BRUSILOVSKY, P. (1999): "Adaptive and Intelligent Technologies for Web-based education". *Künstliche Intelligenz* 4: S. 19 – 26.
- DOIGNON, J.P. UND FALMAGNE, J.C. (1999): *Knowledge Spaces*. Berlin, Heidelberg, New York: Springer.
- DU BOULAY, B. UND SOTHCOTT, C. (1987): "Computers Teaching Programming: An Introductory survey of the field". In: *Artificial Intelligence and Education*, herausgegeben von Lawler, R.W. und Yazdani, M., Ablex, Band 1, S. 345 – 372.

- DÜNTSCH, I. UND GEDIGA, G. (1995): "Skills and knowledge structures". *British Journal for Mathematical and Statistical Psychology* 48: S. 9 – 27.
- EDELMANN, W. (1996): *Lernpsychologie*. Weinheim: Psychologie Verlags Union, 5. Auflage.
- FALMAGNE, J.C.; DOIGNON, J.P.; KOPPEN, M.; VILLANO, M. UND JOHANNESSEN, L. (1990): "Introduction to Knowledge Spaces: How to Build, Test and search them". *Psychological Review* 97 (2).
- GREGG-HARRISON, T.S. (1991): "Learning Prolog in a schema-based environment". *Instructional Science* 20: S. 173 – 192.
- GUARINO, N. (1998): "Formal Ontology and Information Systems". In: *Formal Ontology in Information Systems. Proceedings of the First International Conference, June 6-8, Trento, Italy*, herausgegeben von Guarino, N. Amsterdam, Berlin, Oxford, Tokyo, Washington: IOS Press, S. 3–19.
- HONG, J. (1997): "An Intelligent Tutoring and Assessing System for Prolog Programming". In: *Artificial Intelligence in Education*, herausgegeben von du Boulay, B. und Mizoguchi, R. Amsterdam, Berlin, Oxford etc.: IOS Press, S. 586–588.
- HOPPE, H.U. (1995): "The Use of Multiple Student Modeling to Parameterize group learning". In: *Proceedings of the AI-ED 95, International Conference on Artificial Intelligence in Education*.
- IKEDA, M.; GO, S. UND MIZOGUCHI, R. (1997): "Opportunistic Group Formation". In: *Artificial Intelligence in Education*, herausgegeben von du Boulay, B. und Mizoguchi, R. Amsterdam, Berlin, Oxford etc.: IOS Press.
- IKEDA, M.; HAYASHI, Y.; LAI, J.; CHEN, W.; BOURDEAU, J.; SETA, K. UND MIZOGUCHI, R. (1999): "An Ontology - more than a shared vocabulary". In: *AI-ED 99 Workshop on Ontologies for Educational Systems*.
- KAY, J. UND KUMMERFELD, B. (1997): "User models for customized hypertext". In: *Intelligent Hypertext: Advanced Techniques for the World Wide Web*, herausgegeben von Nicolas, C. und Mayfield, J., Berlin, Heidelberg, New York: Springer, LNCS.
- KLOSE, G.; LANG, E. UND PIRLEIN, T. (1992): *Ontologie und Axiomatik der Wissensbasis von LILOG*. Berlin, Heidelberg, New York: Springer.
- KUHN, T.S. (1991): *Die Struktur wissenschaftlicher Revolutionen*. Suhrkamp, 11. Auflage.
- MCCALLA, G. (1994): "The Central Importance of Student Modelling to Intelligent tutoring". In: *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, herausgegeben von McCalla, G. und Geer, J., Berlin, Heidelberg, New York: Springer.
- MCCALLA, G.; GREER, J.; KUMAR, V.S.; MEAGHER, P.; COLLINS, J.; TKATCH, R. UND PARKINSON, B. (1997): "A peer help System for workplace training". In: *Artificial Intelligence in Education*, herausgegeben von du Boulay, B. und Mizoguchi, R. Amsterdam, Berlin, Oxford etc.: IOS Press, S. 183 – 190.

- MIZOGUCHI, R. UND BOURDEAU, J. (1999): "Using Ontological Engineering to Overcome Common ai-ed problems". *International Journal of Artificial Intelligence in Education* 11.
- PEYLO, C.; TEIKEN, W.; ROLLINGER, C. UND GUST, H. (2000): "An Ontology as domain model in a web-based educational system for prolog". In: *Proceedings of the 13th Flairs Conference*. AAAI Press. To appear.
- PIRLEIN, T. (1995): *Wiederverwendung von Commonsense Ontologien im Knowledge Engineering*. Sankt Augustin: Infix.
- PUTNAM, H. (1990): *Vernunft, Wahrheit und Geschichte*. Frankfurt: Suhrkamp Taschenbuch Wissenschaft.
- SCHULMEISTER, R. (1997): *Grundlagen hypermedialer Lernsysteme*. München, Wien: Oldenbourg, 2. Auflage.
- SELF, J. (1992): "Computational Mathematics: The Missing Link in Intelligent Tutoring System Research". In: *New Directions for Intelligent Tutoring Systems*, herausgegeben von Costa, E., Berlin, Heidelberg, New York: Springer, S. 38 – 56.
- SELF, J.A. (1974): "Student models in computer aided instruction". *International Journal of Man-Machine Studies* 6: S. 261 – 276.
- SELF, J.A. (1994): "Formal Approaches to Student Modelling". In: *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, herausgegeben von Geer, J.E. und McCalla, G.I., Berlin, Heidelberg, New York: Springer, S. 295 – 355.
- SOWA, J.F. (1999): *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, Albany, Belmont etc.: Brooks/Cole.
- WINKELS, R. UND BREUKER, J. (1992): "What's in a ITS? a Functional Decomposition". In: *New Directions for Intelligent Tutoring Systems*, herausgegeben von Costa, E., Berlin, Heidelberg, New York: Springer, S. 57 – 68.