Zur Modellierung des hilfegeleiteten Wissenserwerbs beim Problemlösen

Olaf Schröder, Claus Möbus

Universität Oldenburg, Fachbereich Informatik, Abt. Lehr- Lernsysteme

Zusammenfassung

Dieser Beitrag beschreibt einen Ansatz zur Modellierung hilfegeleiteter Wissenserwerbsprozesse beim Problemlösen. Ausgangspunkt ist eine theoretische Konzeption, die motivationale und volitionale Aspekte der Handlungssteuerung mit Aspekten des Lernens verbindet. Wissenserwerb wird dabei als Wechselspiel des Erwerbs neuen Wissens mit Hilfen nach Stocksituationen sowie der Wissensoptimierung durch Komposition aufgefaßt. Für diese theoretische Konzeption werden für die Domäne des Programmierens in der graphischen funktionalen Sprache ABSYNT zwei unterschiedlich abstrakte lauffähige Realisationen vorgestellt: ein internes Modell der sukzessiven Wissensstadien eines Lernenden, und ein externes Modell der Wissenserwerbsprozesse, die die Übergänge zwischen den Stadien ermöglichen. Das interne Modell ist integrierter Teil eines Hilfesystems zum Erwerb von Programmierwissen für ABSYNT. Es ist in der Lage, unter Nutzung zeitprotokollierter Problemlösedaten online Lösungsentwürfe zu untersuchen und möglichst wissensstandsangepaßte Hilfen zu generieren. Es beschreibt die Wissensentwicklung als Fortschreiten auf einem Kontinuum von Mikroregeln über Lösungsschemata bis hin zu spezifischen Lösungsbeispielen für Programmieraufgaben und für Teilprobleme aus diesen. Das externe Prozeßmodell enthält darüber hinaus die hypothetischen Gründe (in Form von Wissenserwerbs- und Problemlöseprozessen) für das Fortschreiten über die Wissensstadien. Es ist als kognitives Wissenserwerbsmodell zur vollständigen Modellierung kontinuierlicher Wissenserwerbsprozesse und der dabei ablaufenden Handlungs- und Verbalisationssequenzen konzipiert. Internes Modell, externes Modell und die zugrundeliegende theoretische Konzeption beschränken sich gegenseitig, und es wird maximale Konsistenz zwischen ihnen angestrebt.

Einleitung

Die computerisierte Modellierung von Vorgängen der Informationsverarbeitung und speziell von Wissenserwerbsprozessen hat sich zu einem wichtigen Forschungsfeld entwickelt. Modelle dieser Art ermöglichen detaillierte Hypothesen zur Repräsentation und zur Veränderung von Wissen, und entsprechend detaillierte und manchmal überraschende empirische Vorhersagen. Sie tragen somit zur kognitionspsychologischen Theorienbildung und zur Grundlagenforschung bei. Zum anderen wird gerade durch die Modellierung von Wissenserwerbsprozessen der Weg gebahnt für die Entwicklung wissensstandsangepaßter Instruktionen und Hilfen. Dies ist nicht nur eine wichtige Voraussetzung für die individualisierte und damit effizientere Informationsvermittlung, und damit für die Effektivierung von Lehr-/ Lernprozessen, sondern auch für die Gestaltung benutzerorientierter Informatiksysteme.

Bereits die Diagnose von Wissensständen ohne Berücksichtigung von Wissensveränderungen ist jedoch ein schwieriges Problem (Self, 1990). Noch schwieriger ist die Modellierung von Wissenserwerbsprozessen, die diese hypothetischen

Wissensstadien erst erzeugen. Zwei notwendige Voraussetzungen, sich diesem Ziel zu nähern, sind:

• eine *theoretische Konzeption*, die der Breite der Phänomene beim Wissenserwerb im Zuge des Problemlösens gerecht wird. Um Fähigkeits- und Wissenserwerbsprozesse in verschiedenen instruktionalen Settings (wie Wissenserwerb mit Regeln, aus Beispielen, mit Hilfen, mit vorgegebenen Problemlösungen oder mit selbständiger Setzung von Problemzielen) modellieren zu können, ist ein theoretischer Rahmen erforderlich, der die Setzung von Problemzielen seitens des Problemlösers, die Handlungssteuerung, das Erleben von Stocksituationen, das Aufsuchen, Finden und Verarbeiten neuer Information, die Überprüfung und Korrektur von Lösungsentwürfen, den Erwerb und die Modifikation von Wissen umfaßt. Darüber hinaus muß sich diese theoretische Konzeption auf einem hohen Auflösungsniveau bewegen, auf dem spezifische Handlungs- und Verbalisationssequenzen beschrieben werden können. Schließlich muß die theoretische Konzeption auf verschiedene Problemlösedomänen anwendbar, also domänenunabhängig sein.

• eine entsprechend *detaillierte Erhebung der Daten*, die als Indikatoren für Wissensbestände, Stocksituationen, Problemlöseheuristiken, Prozesse der Wissensveränderung usw. dienen. Dies setzt im weiteren voraus, daß dem Problemlöser hinsichtlich der Wahl und Sequenzierung einzelner Handlungsschritte genügend Spielraum gegeben wird. Anderenfalls erfährt man durch die Modellierung mehr über den Gegenstandsbereich als über die kognitiven Prozesse des Problemlösers. Nur die vom Problemlöser frei getroffenen Problemlöseentscheidungen sind für die Modellbildung potentiell bedeutsame Problemlösedaten.

Sind diese beiden Voraussetzungen erfüllt, so kann mit der Entwicklung eines lauffähigen Modells eine Brücke zwischen der theoretischen Konzeption und den empirischen Problemlösedaten geschlagen werden. Das Modell dient dabei einerseits der Realisation und Konkretisierung der theoretischen Konzeption und andererseits der Reproduktion und ggf. Vorhersage der empirischen Daten.

Die von uns angestrebte *theoretische Konzeption* ist die *ISPDL-Architektur* (Impasse - Success - Problem Solving - Driven Learning), an der wir gegenwärtig arbeiten. In ihr werden motivations- und volitionstheoretische Aspekte (insbesondere Heckhausen´s (1989) Rubikonmodell) und lerntheoretische Aspekte miteinander verbunden. Die lerntheoretischen Aspekte sind dabei als Wechselspiel zweier Teilprozesse konzipiert:

• Induktiver *Erwerb neuen Wissens* in Stocksituationen ("Impasse Driven Learning", van Lehn, 1988; 1990; 1991). Unter *Stocksituationen* verstehen wir Problemsituationen, in denen der Problemlöser zunächst nicht weiter weiß ("stecken bleibt"), da ihm das nötige Wissen bzw. Handlungsrepertoire fehlt. Der Problemlöser setzt dann schwache (domänenunabhängige) Heuristiken zur Überwindung der Stocksituation ein, wie Fragen zu stellen, in Büchern oder Anleitungen nachzuschlagen usw., also allgemein die Nutzung von *Hilfen*. Mit schwachen Heuristiken können zum einen Stocksituationen überwunden werden und zum anderen neues Wissen erworben werden, so daß vergleichbare Stocksituationen in Zukunft vermieden werden.

• Deduktive *Wissensoptimierung* nach erfolgreicher Wissensnutzung ("Success Driven Learning"): Anderson´s (1983a) Komposition bzw. Newell´s (Rosenbloom & Newell, 1987) Chunking.

Die *Wissensdomäne*, in der wir lauffähige Modelle dieser theoretischen Konzeption realisieren, ist der Erwerb von Programmierwissen in der graphischen funktionalen

Programmiersprache ABSYNT. In ABSYNT sind die Erstellung, Benennung und Verknüpfung von Programmobjekten (wie Operatoren, Parameter etc.) unterschiedliche und darüber hinaus vom Problemlöser frei sequenzierbare Handlungen. Außerdem besteht in ABSYNT die Möglichkeit zur Formulierung von Prüfhypothesen (s.u.). Zum anderen liegen schwierige Probleme wie die Nichtmonotonie von Lösungsschritten (Korf, 1987) in der Domäne ABSYNT nicht vor. Damit erfüllt ABSYNT die Forderung nach einer detaillierten Datenerhebung im oben beschriebenen Sinne.

Die Brücke zwischen der theoretischen Konzeption (ISPDL-Theorie) und den Daten (Problemlöseprotokolle des Programmierens in ABSYNT) besteht bei uns aus zwei verschiedenen Modellen, die unterschiedlich detaillierte Realisationen der ISPDL-Theorie darstellen und dementsprechend unterschiedlich viele Problemlösedaten erklären:

• Das "*Stadienmodell*" oder "*interne Modell*" repräsentiert das Domänenwissen des Lernenden zu verschiedenen Zeitpunkten im Wissenserwerbsprozeß. Es wird im Kontext eines Problemlösemonitors entwickelt, der den Lernenden beim Entwurf von ABSYNT-Programmen mit Hilfen unterstützt. Das Stadienmodell ermöglicht die online-Diagnose der Lösungsentwürfe des Lernenden und die online-Generierung von Hilfen. Es ist also Bestandteil des Problemlösemonitors (deshalb "internes Modell"), beschreibt den *domänenbezogenen Wissensstand* und beruht auf vom Rechner abgreifbaren Daten: Programmierhandlungen und die dabei benötigte Zeit.

• Das "*Prozeßmodell*" oder "*externe Modell*" dagegen ist nicht integrierter Bestandteil des Problemlösemonitors (also "extern"). Es enthält hypothetische Wissenserwerbs- und Wissensveränderungsprozesse, Stocksituationen und schwache Heuristiken und geht damit über das Stadienmodell hinaus: Es liefert die (hypothetischen) *Gründe* für die Änderungen des Domänenwissens gemäß dem Stadienmodell. Es dient der Reproduktion und Vorhersage *auch* solcher empirischer Daten, die nicht rechnerseitig erfaßbar sind, wie etwa Verbalisationen.

Abb. 1 veranschaulicht die Beziehung zwischen dem internen und dem externen Modell. Das Prozeßmodell (externes Modell) ist als kognitives Wissenserwerbsmodell konzipiert. Das Stadienmodell (internes Modell) dagegen enthält auf der Theorieebene, der Modellebene und der Datenebene nur *Teilaspekte* des externen Modells: die Planung und den Einsatz von Handlungsoperatoren, die online-Diagnose und Hilfengenerierung und die Verarbeitung der vom Rechner erfaßbaren Daten. (Die Beziehung beider Modelle zur ISPDL-Theorie wird im nächsten Abschnitt erläutert werden.) Das interne Modell repräsentiert eine konkrete Anwendung der ISPDL-Theorie im Rahmen des Problemlösemonitors. Es dient der Überprüfung von Vorhersagen und der Bildung neuer Hypothesen, welche jedoch wiederum in die Weiterentwicklung des externen Modells und der ISPDL-Theorie einfließen. Es wird also größtmögliche Konsistenz von internem Modell, externem Modell und der ISPDL-Theorie angestrebt.

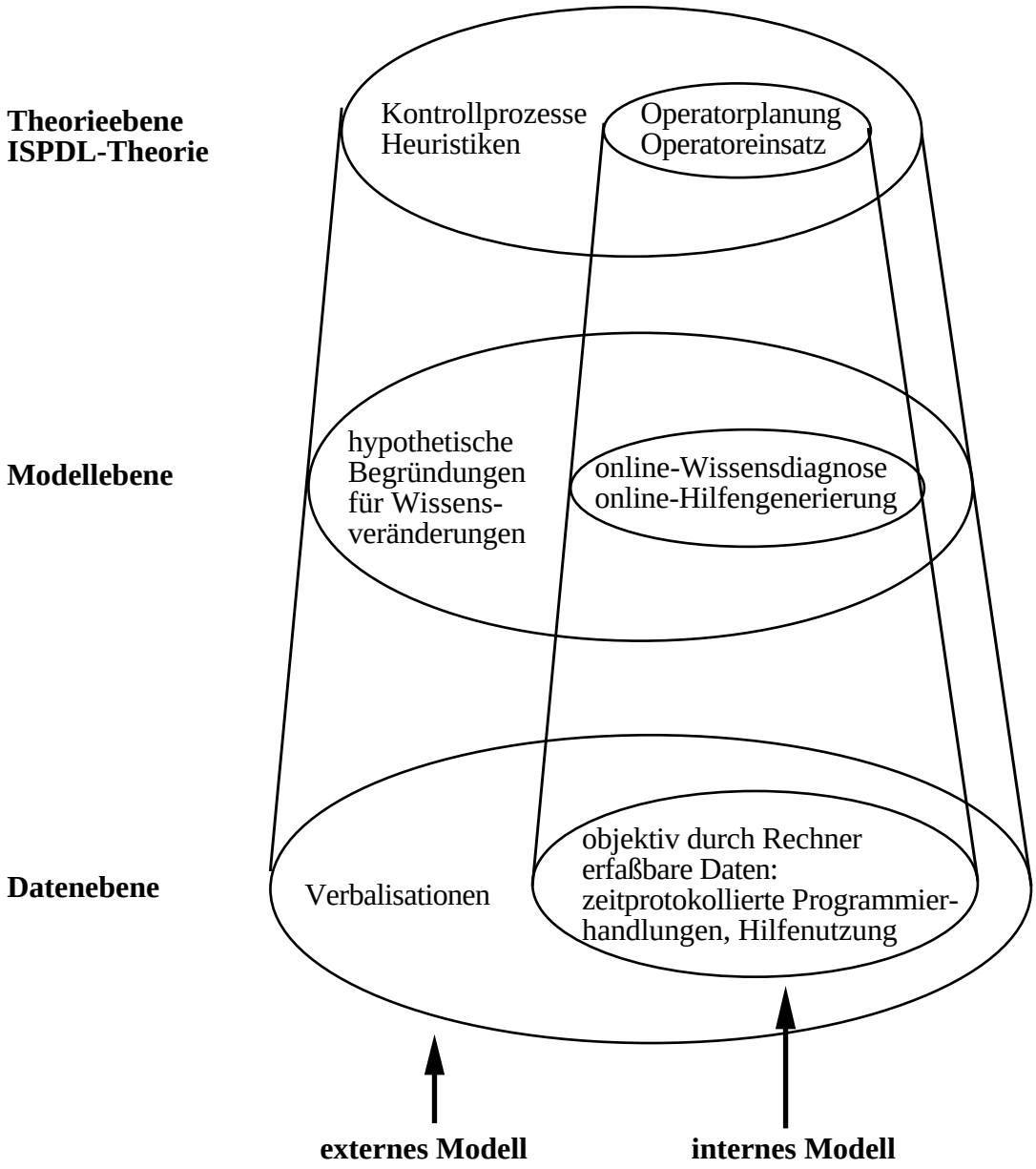


Abb. 1: Zur Relation zwischen internem Modell (Stadienmodell) und externem Modell (Prozeßmodell)

Dieser Beitrag enthält die Darstellung des aktuellen Standes unserer Arbeiten zur Modellierung hilfegeleiteter Wissenserwerbsprozesse beim Problemlösen. Zunächst wird unsere theoretische Konzeption, die ISPDL-Architektur, dargestellt. Dann wird ein kurzer Überblick über unsere Wissensdomäne ABSYNT gegeben. Anschließend wird das interne Modell sukzessiver Wissensstadien Lernender beschrieben und illustriert. Der letzte Abschnitt referiert den Stand des externen Modells, das mit hypothetischen Problemlöse- und Wissenserwerbsprozessen die Wissensentwicklung über die Stadien beschreiben soll. Abschließend wird ein kurzer Ausblick auf die weitere Arbeitsplanung gegeben.

Problemlösendes Handeln und Wissenserwerb: Die ISPDL-Architektur

Die ISPDL-Architektur (Impasse - Success - Problem solving - Driven - Learning) ist der Ausgangspunkt für eine theoretische Konzeption (Möbus, 1991b), von der wir glauben, daß sie den skizzierten Anforderungen hinsichtlich Phänomenbreite, Auflösungsgrad und Domänenunabhängigkeit gerecht werden kann. Sie verbindet drei inhaltliche Aspekte miteinander:

• Die *motivations- und volitionstheoretische Strukturierung* des Problemlöseprozesses von der Zielfindung über die Handlungsplanung und -ausführung bis zur Bewertung und ggf. Modifikation einer Lösung im Sinne des Rubikonmodells von Heckhausen (1987; 1989) und Gollwitzer (1990). Im Rubikonmodell werden vier Handlungsphasen unterschieden. In der Phase des *Abwägens* untersucht der Problemlöser verschiedene mögliche Zielintentionen im Hinblick auf ihre Machbarkeit und Wünschbarkeit. Hat er sich für ein Ziel entschieden, so beginnt die Phase des *Planens*, also der Vorbereitung der Zielrealisierung. Hierzu gehört u.a. die Bildung und Sequenzierung von Subzielen. Der Plan wird *ausgeführt*, und das erhaltene Ergebnis wird z.B. im Hinblick auf die eigenen Wünsche, auf Oberziele usw. *bewertet*. Gollwitzer (1990) postuliert darüber hinaus für jede Handlungsphase spezifische Einstellungen des Problemlösers.

• *Wissenserwerb* durch den Einsatz schwacher Heuristiken, wie Informationssuche, Analogiebildung, Problemlöseheuristiken etc., nach Stocksituationen (Erwerb neuen Wissens durch "*Impasse Driven Learning*", van Lehn, 1988; 1990; 1991). Ist die Anwendung der schwachen Heuristik erfolgreich in dem Sinne, daß die Stocksituation überwunden werden kann, so wird neues Wissen erworben: nämlich die Information, die zu der Überwindung der Stocksituation beigetragen hat, bzw. deren vorheriges Fehlen erst zu der Stocksituation geführt hat (Möbus, 1990; Schröder, 1990a, 1990b). Dieses Wissen steht dann in späteren Situationen zur Verfügung und hilft so, ähnlich begründete Stocksituationen zu vermeiden. Kann die Stocksituation hingegen mit der aufgesuchten Information nicht überwunden werden, so kann ähnlich wie schon im General Problem Solver (Ernst & Newell, 1969) ein neues Problem (sekundäre Stocksituation, Brown & van Lehn, 1980) entstehen. Eine konsequente Realisierung der Theorie des Impasse Driven Learning ist die allgemeine Problemlösearchitektur SOAR (Laird, Rosenbloom & Newell. 1986; 1987; Young, 1991).
Nach der Theorie des Impasse Driven Learning sucht der Lernende neue Information nur dann auf bzw. ist für sie empfänglich, wenn er sich in einer Stocksituation befindet. Stocksituationen sind also der Motor des Wissenserwerbs, da sie Problemlösen, Anforderung von Hilfen etc. anregen (van Lehn, 1988). Neue Informationen werden nur dann aufgenommen, wenn sie wirklich benötigt werden (ein Prinzip, das auch außerhalb der Theorie des Impasse Driven Learning als wichtig angesehen wird, z.B. Winkels & Breuker, 1990). Ohne Stocksituation hingegen werden neue Informationen als lästig empfunden ("unerbetene Ratschläge"). Das *aktive Aufsuchen* von Information durch den Lernenden gibt also Hinweise auf Wissenslücken und damit auf seine Wissensstrukturen sowie auf die ablaufenden Wissenserwerbsprozesse, und ist damit ein wichtiges empirisches Datum (vgl. auch Self, 1990).

• *Optimierung* von schon vorhandenem Wissen *ohne* Stocksituationen durch erfolgreiche Wissensanwendung ("*Success Driven Learning*"). Dieser Aspekt wird von einer Reihe von Ansätzen akzentuiert, die van Lehn (1990) als "Memorize-and-Compile"-Theorien bezeichnet: Wissen wird durch "Kompilation" automatisiert, so daß es effizienter angewendet werden kann. Beispiele für Wissensoptimierungsprozesse im Sinne des Success Driven Learning sind Chunking (Elio, 1986; Iba, 1989; Rosenbloom & Newell, 1986; 1987; Wolff, 1987), Komposition (Anderson, 1983a; b; 1986; 1989; Lewis, 1987; Neves & Anderson, 1981; Vere, 1977), Prozeduralisierung (Anderson, 1983a; 1986; 1989), und die Bildung rekursiver Makrooperatoren (Cheng & Carbonell, 1986). Wir beschreiben Wissensoptimierung durch Komposition, weil sie ein klares Konzept darstellt und eine Reihe prüfbarer Vorhersagen erlaubt (s.u.).

Abb. 2 bis 5 zeigen den gegenwärtigen Stand der Überlegungen zu der ISPDL-Architektur in der Form eines höheren hierarchischen Petrinetzes. Mit Petrinetzen lassen sich *verteilte* zeitdiskrete Prozesse beschreiben und modellieren. Die Semantik von Petrinetzen ist u.a. in Reisig (1982; 1985) und die der höheren hierarchischen Netze in Huber et al. (1990) dargestellt. Wir wollen es hier bei einer natürlichsprachlichen Beschreibung bewenden lassen:

Die ISPDL-Architektur gliedert sich in vier Subaspekte: Problemverarbeitung, Zielverarbeitung, operationale und nichtoperationale Zielverarbeitung. Jeder dieser Subaspekte wird durch ein Netz bzw. durch eine Seite (Abb. 2 - 5) dargestellt. Stellen (Ellipsen) in den Netzen repräsentieren Zustände bzw. Datenbehälter. Transitionen (Rechtecke) repräsentieren Ereignisse bzw. Verarbeitungsschritte. Die Stellen können Marken tragen, die für mentale Objekte (Ziele, Gedächtnisspuren, Heuristiken) oder reale Objekte (Problemlösung, Verhaltensprotokoll) stehen. Stellen können zusätzliche Bezeichner (tags) tragen. So bedeutet ein "B" in einem Quadrat, daß diese Seite über diese Border-Stelle betreten (IN) oder verlassen (OUT/EXIT) wird. Stellen mit dem Bezeichner "FG" gehören einer globalen Fusionsmenge an. Diese Stellen stehen in allen Netzen bzw. Seiten mit gleichem Inhalt zur Verfügung.

Die Informationsverarbeitung wird durch das Schalten von Transitionen modelliert. Eine Transition kann schalten, wenn sie von allen Stellen im Vorbereich (d.h. Stellen, die durch einen Pfeil *auf* die Transition mit dieser verbunden sind) eine Marke (Ziel, Regel etc.) abzieht und allen Stellen im Nachbereich (d.h. Stellen, die durch einen Pfeil *von* der Transition mit dieser verbunden sind) eine Marke zuliefert. Bei einem Doppelpfeil befindet sich die Stelle im Vor- und Nachbereich der Transition.

Der Prozeß beginnt mit dem Netz "Problem Processing" (Abb. 2) mit "Goals". (Auf die Schattierungen in einigen Stellen und Transitionen der Netze wird weiter unten eingegangen.) Zunächst wägt der Problemlöser zwischen den verschiedenen alternativen Zielen unter Einbeziehung seines Wissens ("Knowledge base") ab (1. Phase des Rubikonmodells). Diese "Deliberate"-Phase führt zur Auswahl eines bestimmten Ziels ("Goal"). Das Ziel wird bearbeitet ("Goal Processing"), dieser Prozeß ist im folgenden Subnetz (Abb. 3) enthalten. Es resultiert eine Lösung ("Solution"), und das verwendete Wissen der Wissensbasis wird über deduktive Prozesse optimiert ("Success Driven Learning"). Sind alle Ziele auf diese Weise bearbeitet ("Solutions"), so ist der Prozeß beendet.

Die Transition "Goal Processing" in Abb. 2 trägt den tag "HI" (hierarchical invocation transition). Damit wird der Fortgang des kognitiven Prozesses in einer neu erzeugten Instanz des Netzes "Goal Processing" angezeigt. Zusätzlich wird durch die beiden

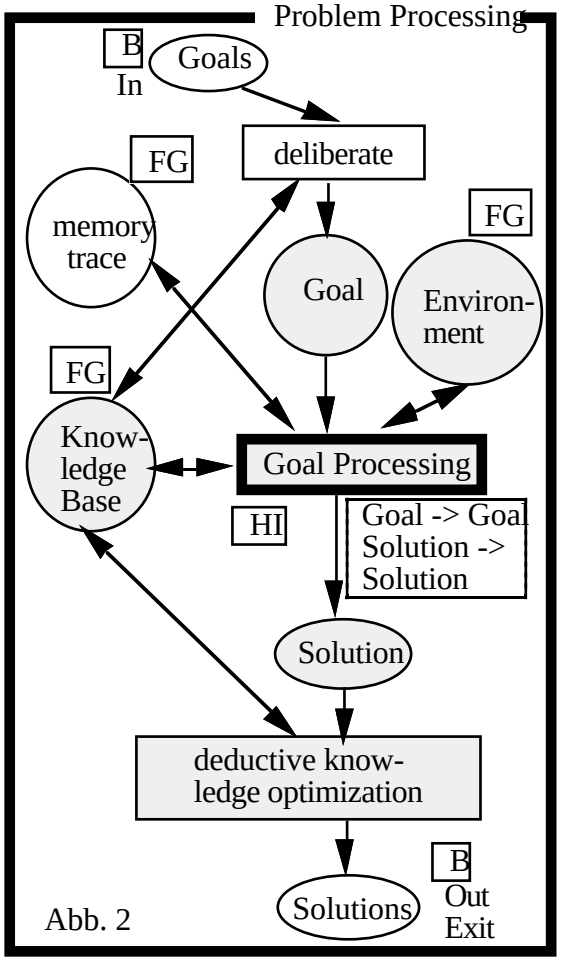


Abb. 2

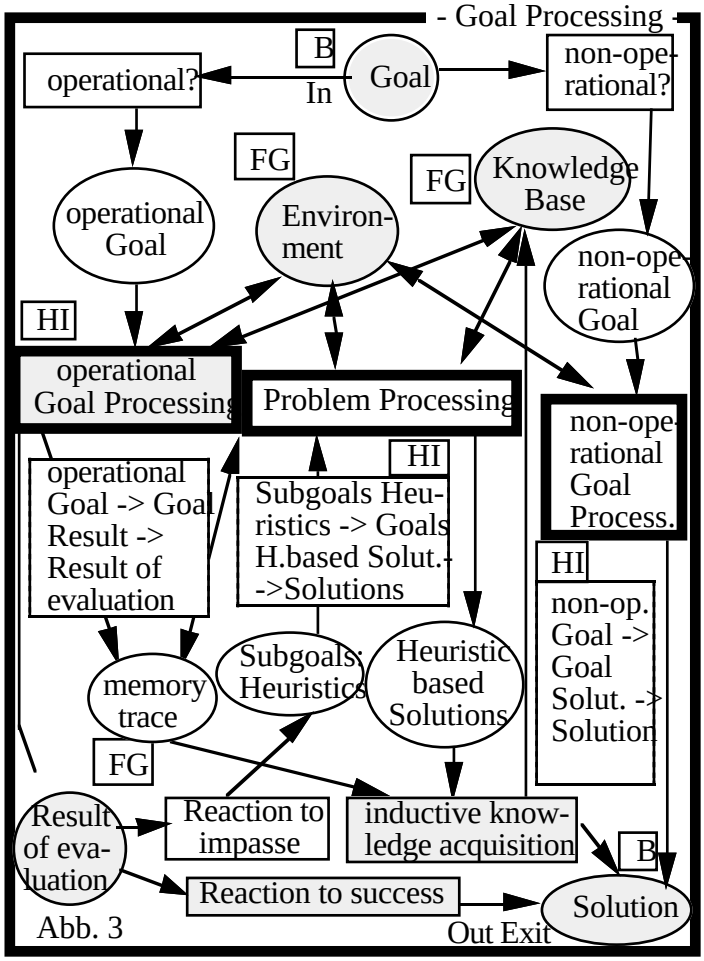


Abb. 3

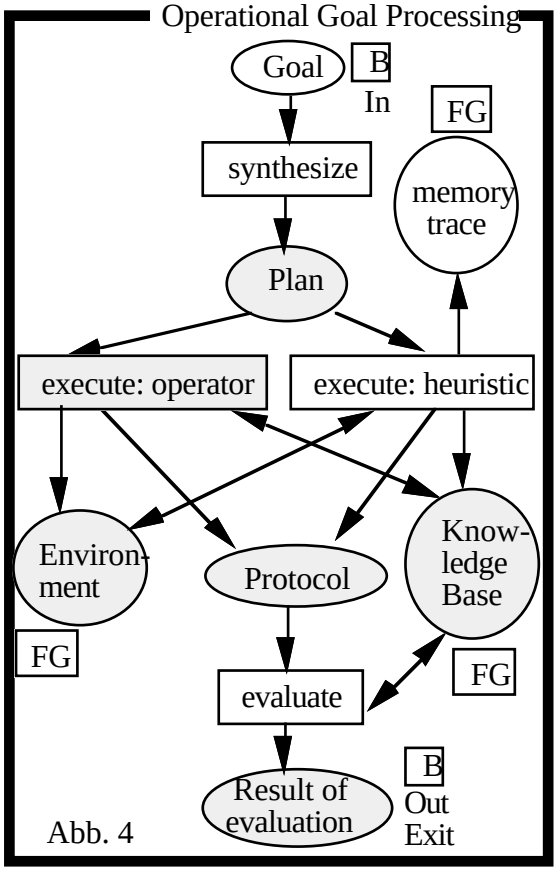


Abb. 4

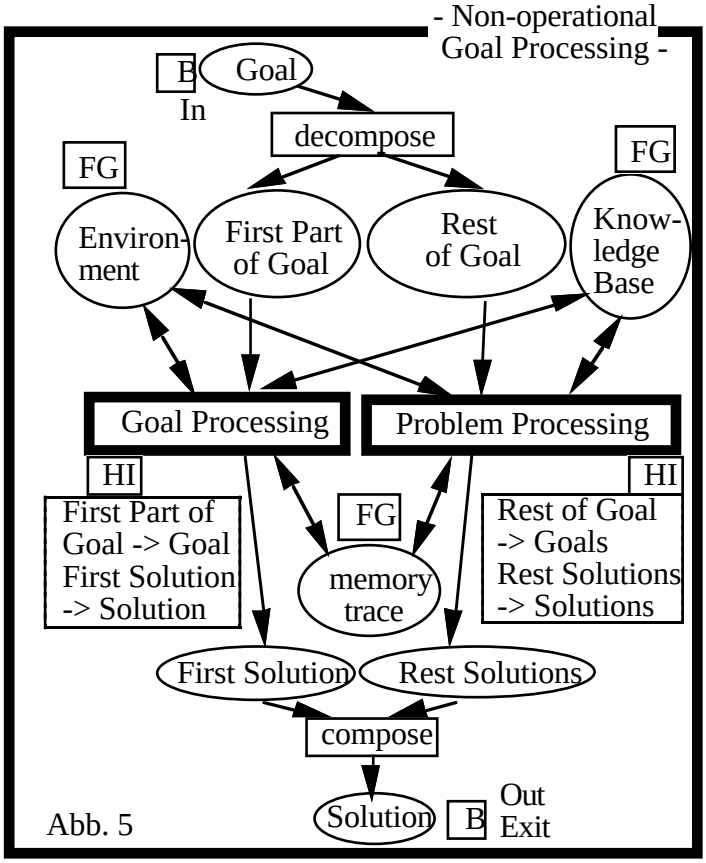


Abb. 5

Zuordnungen "Goal -> Goal" und "Solution -> Solution" (in Abb. 2 in dem Rechteck mit durchbrochener Linie) gefordert, daß die Inhalte der Stellen "Goal" bzw. "Solution" im Netz "Problem Processing" in den Stellen "Goal" bzw. "Solution" des Subnetzes "Goal Processing" auftauchen. Im Subnetz "Goal Processing" (Abb. 3) wird zunächst geprüft, ob das Ziel ("Goal") operational ist oder in Teilziele zerlegt werden muß. Ist es operational, so wird es bearbeitet (Subnetz "Operational Goal Processing", Abb. 4). Ein Lösungsplan wird synthetisiert und ausgeführt. Er kann sich auf Domänenoperatoren ("execute: operator") oder auf Heuristiken ("execute: heuristic") beziehen. Die entstandene Lösung ("Protocol") wird bewertet, und das Netz "Operational Goal Processing" wird wieder verlassen. Die Teilprozesse "synthesize", "execute: operator", "execute: heuristic" und "evaluate" in Abb. 4 korrespondieren zu entsprechenden Phasen des Rubikonmodells.

Zurück zu Abb. 3: Nach positiver Bewertung ("reaction to success") wird das Netz "Goal Processing" über "Solution" verlassen. Es kann aber auch eine Stocksituation vorliegen ("reaction to impasse"). In diesem Fall kann der Problemlöser verschiedene Heuristiken in Betracht ziehen. Es findet erneut "Problem processing" statt, nun jedoch bezogen auf die Auswahl und den Einsatz von Heuristiken, wie z.B. die Nutzung von Hilfen. Dadurch wird induktiv neues Wissen erworben ("Impasse Driven Learning").

Im Falle eines nichtoperationalen Ziels (Abb. 5) schließlich wird ein operationales Teilziel abgespalten und über "Goal processing" verarbeitet. Mit dem Rest des nichtoperationalen Ziels wird erneut "Problem processing" durchgeführt. Anschließend werden die erhaltenen Teillösungen zur Gesamtlösung des nichtoperationalen Ziels zusammengesetzt.

Wie schon einleitend erwähnt, sind das interne Modell (Stadienmodell) und das externe Modell (Prozeßmodell) als unterschiedlich abstrakte Realisationen der ISPDL-Architektur konzipiert. Das externe Modell soll (möglichst) alle Komponenten der ISPDL-Architektur umfassen. (Im gegenwärtigen Implementationsstand des externen Modells ist dies noch nicht erreicht). Das interne Modell umfaßt dagegen nur die Komponenten der ISPDL-Architektur, die in den Abb. 2 bis 4 schattiert dargestellt sind; z.B. in Abb. 2 die Stellen "Goal" und "Solution" und die Transition "Goal Processing".

Erwerb von Programmierwissen im ABSYNT-Problemlösemonitor

Der ABSYNT-Problemlösemonitor stellt eine visuelle Lernumgebung dar (*iconic environment* im Sinne von Glinert, 1990) und unterstützt Novizen beim Erwerb von Wissen über funktionale Programmierkonzepte bis hin zu rekursiven Systemen. Der ABSYNT-Problemlösemonitor kann Hilfen und Lösungsvorschläge bereitstellen, enthält jedoch keine curriculare Komponente. Seine wichtigsten Bestandteile sind ein visueller

Programmeditor, ein visueller Trace und eine Hilfekomponente. Sie werden hier kurz dargestellt. Eine ausführliche Beschreibung der Sprache sowie ihre Begründung und Analyse mit kognitiven Designprinzipien befinden sich in Möbus & Thole, 1989.

Im *Editor* (Abb. 6) können ABSYNT-Programme konstruiert werden. Der Problemlöser kann sie außerdem syntaktisch überprüfen lassen. Ein ABSYNT-Programm befindet sich in einem Rahmen und besteht aus einem Kopf und einem Körper. Der Startbaum dient dem Programmaufruf. Die Knoten der Bäume sind Konstanten, Parameter, primitive und selbstdefinierte Operatorknoten. Knoten werden mit der Maus aus der Menüleiste am linken Rand eines Start- bzw. Rahmenfensters entnommen und miteinander verbunden. Die einzelnen Programmierhandlungen sind vom Benutzer z.B. im Sinne von "top down"-, "bottom-up"- oder "middle out"-Programmierung frei sequenzierbar (dies ist für die Modellierung wichtig, s.u.). So können z.B. Verbindungslinien bereits gezogen werden, wenn die zu verbindenden Knoten noch nicht programmiert wurden, diese werden dann durch Schatten repräsentiert (Abb. 6). Abb. 7 illustriert die freie Sequenzierbarkeit der Programmierhandlungen einer Person anhand einer zeitprotokollierten Handlungssequenz.

Abb. 6: Schnappschuß des visuellen ABSYNT-Programmeditors

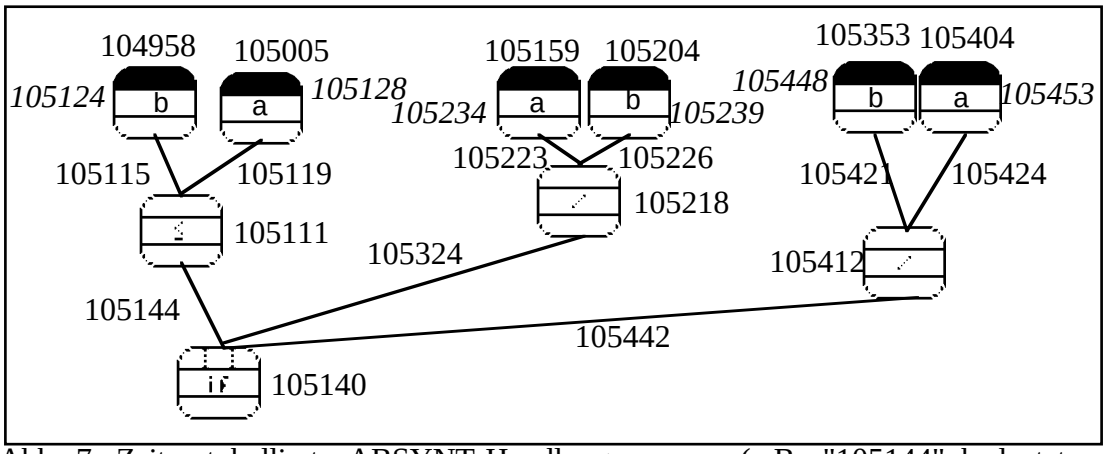


Abb. 7: Zeitprotokollierte ABSYNT-Handlungssequenz (z.B. "105144" bedeutet: um 10:51:44 Uhr wurde die Verbindung vom if-Knoten zum ≤ Knoten gezogen; kursiv geschriebene Zeitindices bezeichnen die Aktion der Knotenbeschriftung)

Der visuelle *Trace* ist ein Ergebnis der Entwicklung lauffähiger Spezifikationen des ABSYNT-Interpreters. In dem Trace wird jeder Berechnungsschritt des Interpreters visualisiert. Der Lernende kann die Interpreterschritte auch selbst mit Maus und Tastatur unter Nutzung visueller Repräsentationen der Interpreterspezifikationen als Hilfen vorhersagen (Möbus & Schröder, 1989; 1990).

Die *Hilfekomponente* besteht aus einer Hypothesentestumgebung und einer Menge von Planungshilfen. In der Hypothesentestumgebung kann der Lernende Hypothesen über die Korrektheit seiner Lösungsentwürfe (oder auch ihrer Teile) formulieren (Möbus, 1990; 1991a; Möbus & Thole, 1990). Abb. 8 zeigt einen Lösungsentwurf zu der "Gerade"-Aufgabe ("Gesucht ist ein Programm, das prüft, ob eine natürliche Zahl gerade ist"). Der Lernende hat einen Teil seines Entwurfs markiert (fette Linien und fett umrandete Knoten). Dieser markierte Entwurfsausschnitt stellt die aktuelle Hypothese des Lernenden dar: "Ich behaupte, daß sich dieser Ausschitt zu einer korrekten Lösung für die 'Gerade'-Aufgabe vervollständigen läßt!" Der Lernende hat die Hypothese anschließend dem System über ein pop-up-Menü zur Analyse übergeben. Die Antwort des Systems lautet jedoch: "Nein, dieser Entwurfsausschnitt kann nicht zu einer bekannten Lösung ergänzt werden." (unteres Fenster der Abb. 8). Daraufhin hat der Lernende einen kleineren Entwurfsausschnitt als Hypothese ausgewählt und dem System übergeben (Abb. 9). Diesen Entwurfsausschnitt erkennt das System als zu einer korrekten Lösung ergänzbar und meldet dies mit der Kopie der Hypothese zurück (unteres Fenster der Abb. 9). Der Lernende kann sich diese Kopie nun von dem System schrittweise ergänzen lassen (Abb. 10, unteres Fenster), denn das System hat durch Ergänzung der Hypothese eine komplette Lösung generiert. Diese komplette Lösung kann sich der Lernende ebenfalls zeigen lassen. Der Hypothesentestansatz wurde aus folgenden drei Gründen realisiert:

Abb. 8: Lösungsentwurf für die "Gerade"-Aufgabe (oberes Fenster) mit Hypothese (fett) und negativer Systemrückmeldung (unteres Fenster)

Abb. 9: Lösungsentwurf für die "Gerade"-Aufgabe (oberes Fenster) mit Hypothese (fett) und positiver Systemrückmeldung (unteres Fenster)

Abb. 10: Systemrückmeldung von Abb. 9, erweitert um einen Ergänzungsvorschlag (unteres Fenster, fett)

1. In funktionalen Programmen können Fehler meistens *nicht eindeutig* lokalisiert werden. Meist gibt es mehrere Möglichkeiten, einen fehlerhaften Entwurf zu korrigieren. So kann z.B. der in Abb. 10 (oberes Fenster) dargestellte Entwurf durch Änderung des ELSE-Zweiges des if-then-else-Operators korrigiert werden, wie durch den Ergänzungsvorschlag (untere Hälfte in Abb. 10) nahegelegt. Derselbe Entwurf kann aber unter Beibehaltung des ELSE-Zweiges auch durch Abänderung von IF-Zweig und THEN-Zweig korrigiert werden, wie Abb. 11 illustriert. Mit dem Hypothesentestansatz wird die Entscheidung über die beizubehaltenden und die zu ändernden Entwurfsausschnitte dem *Problemlöser* überlassen.

2. Nach der Theorie des Impasse Driven Learning, die Teil unserer ISPDL-Architektur ist, greift der Problemlöser in Stocksituationen *von sich aus* auf Hilfen zu: Ohne Stocksituation kein Informationsbedarf. Das oft ohne explizite Theorie postulierte didaktische Prinzip, daß ein Hilfesystem den Lernenden nicht leichtfertig unterbrechen sollte (z.B. Winkels & Breuker, 1990), folgt somit aus der Theorie des Impasse Driven Learning. Im ABSYNT-Problemlösemonitor werden daher dem Problemlöser Informationen (Fehlerrückmeldungen, Ergänzungsvorschläge) nicht vorgegeben, sondern mit der Hypothesentestmöglichkeit *angeboten*.

Abb. 11: Derselbe Lösungsentwurf wie in Abb. 8 bis 10, aber mit anderer Hypothese sowie mit positiver Systemrückmeldung und Ergänzungsvorschlag

3. Da der Problemlöser selbst Zeitpunkt und Inhalt von Prüfhypothesen bestimmt, stellt der Hypothesentestansatz eine wichtige *Datenquelle* über das aktuelle Domänenwissen des Problemlösers und die ablaufenden Problemlöseprozesse dar. Wenn der Problemlöser die Gelegenheit nutzt, Rückmeldung und Ergänzungsvorschläge zu erhalten, dann erlaubt dies Hypothesen über Stocksituationen und Wissensdefizite.

Die Analyse der Lösungsentwürfe und Entwurfsausschnitte sowie die Synthese von Lösungsentwürfen und von Ergänzungsvorschlägen geschehen mit einer *Ziel-Mittel-Relation* (goals-means-relation "gmr"; Möbus, 1990; 1991a; Möbus & Thole, 1990), die einen UND-ODER-Graph mit parametrisierten Knoten bildet. Die Ziel-Mittel-Relation zerlegt das Aufgabenziel in Subziele, welche weiter ausdifferenziert werden usw. bis auf die Ebene der funktionalen Sprachkonstrukte. Gegenwärtig besteht die Ziel-Mittel-Relation aus 622 Regeln ("GMR-Regeln"). Diese Menge von GMR-Regeln kann als "ungeübtes (reines) Expertenwissen" aufgefaßt werden und wird daher im folgenden auch mit EXPERT bezeichnet. Die Regeln in EXPERT können mehrere Millionen Lösungsentwürfe für insgesamt 40 Programmieraufgaben analysieren und synthetisieren.

Die Regeln der Ziel-Mittel-Relation können als visuelle Planungsregeln repräsentiert und dem Lernenden als *Planungshilfen* angeboten werden. Neben der Rückmeldung und

Bereitstellung von Ergänzungsvorschlägen nach Hypothesen liegen damit auch Hilfen auf *Zielebene* vor. (Abb. 12 und 13 im folgenden Abschnitt zeigen Beispiele.) Explorationen mit den Planungshilfen zeigten, daß sie von Novizen zur Konstruktion von Lösungsentwürfen erfolgreich verwendet werden konnten. Gegenwärtig arbeiten wir daran, die Zielsymbole in den Planungshilfen durch prädikative Beschreibungen zu ersetzen.

Diese Vielfalt der Möglichkeiten zur Analyse und Synthese von Lösungen ist notwendig, um auch "ungewöhnliche" Entwürfe erkennen zu können. Gerade bei Novizen ist im Sinne der Theorie des Impasse Driven Learning zu erwarten, daß Stocksituationen mit möglichst einfachen Heuristiken zu überwinden versucht werden (van Lehn, 1990). Diese aber führen eher zu "lokalen Reparaturen" (Brown & van Lehn, 1980) und somit zu komplex aussehenden Entwürfen, als zu grundätzlichen Umplanungen des Entwurfs. Andererseits macht die Vielfalt der Möglichkeiten zur Entwurfsanalyse und -synthese, zur Bereitstellung von Ergänzungsvorschlägen und von Planungshilfen Selektionskriterien notwendig. In der Steuerung der Entwurfsanalyse und der Auswahl von Hilfen sehen wir die wesentlichen Aufgaben eines *Lernermodells*: Es ermöglicht zum einen eine Effektivierung der Entwurfsanalyse und zum anderen eine Anpassung der jeweiligen Hilfeangebote an den aktuellen Wissensstand des Lernenden. Dies sind die beiden Funktionen des internen Modells im ABSYNT-Problemlösemonitor.

Das interne Modell (Stadienmodell): Der Übergang vom Novizen zum Experten

Das interne Modell (Möbus, Schröder & Thole, 1991a; b) besteht aus einer Menge von Regeln, die den aktuellen hypothetischen Wissensstand des Lernenden repräsentieren. Es dient

- der effizenten online-Analyse und Synthese von Lösungsentwürfen und der online-Bereitstellung wissensstandsangepaßter Ergänzungsvorschläge und Planungshilfen

- der Ableitung empirischer Vorhersagen auf der Verbalisierungs- und auf der Handlungsebene aus dem jeweiligen hypothetischen Wissensstand des Lernenden. Diese Vorhersagen dienen der Validierung des Stadienmodells, aber auch des Prozeßmodells (s.u.), denn dieses muß die hypothetischen Wissensstände des Stadienmodells ebenfalls enthalten und zu denselben Vorhersagen (und darüber hinaus noch zu weiteren) führen.

Das interne Modell wird hier in seinen Grundzügen beschrieben (zu einer ausführlichen Beschreibung siehe Möbus, Schröder & Thole, 1991b). Zunächst werden seine "Bausteine", die Regeln, dargestellt. Dann wird der Ablauf des internen Modells beschrieben, also die Veränderung seiner Regelmenge während des Problemlösens. Es folgt eine kurze Darstellung der empirischen Vorhersagen und Konsequenzen des Modells und der Implikationen für die Gestaltung von Hilfen.

Gemäß der ISPDL-Architektur wird neues Wissen nach Stocksituationen induktiv erworben. Mit diesem Wissen korrespondiert in dem internen Modell *erworbenes* (nicht optimiertes) Wissen. Es besteht aus "*einfachen Regeln*", die eine Teilmenge der GMR-Regeln sind. Weiterhin wird gemäß der ISPDL-Architektur vorhandenes Wissen deduktiv optimiert. Mit diesem Wissen korrespondiert in dem internen Modell *optimiertes* Wissen. Es besteht aus *Komposita*, die aus einfachen Regeln und ggf. Komposita gebildet werden (s.u.), und die die Aktionen des Problemlösers simulieren können.

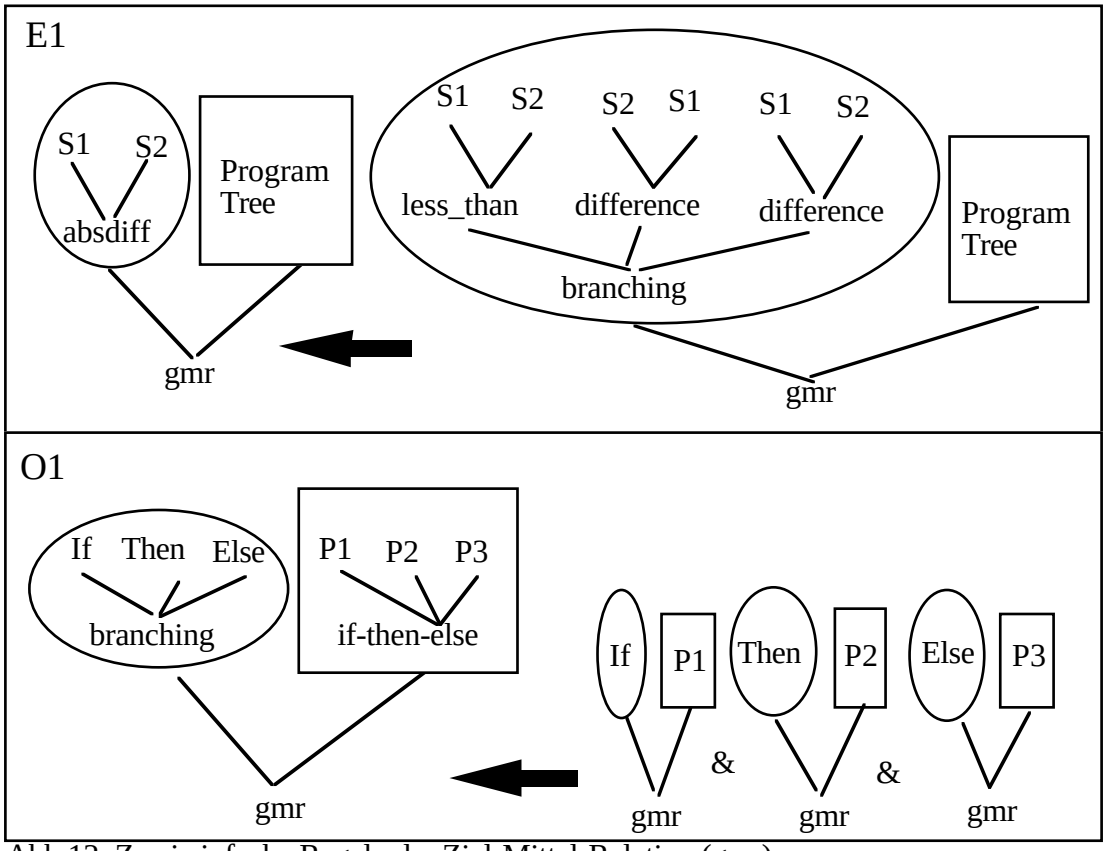


Abb.12: Zwei einfache Regeln der Ziel-Mittel-Relation (gmr)

Abb 12 zeigt zwei einfache Regeln. Jede Regel hat einen Regelkopf links von dem Pfeil, und einen Regelkörper rechts davon. Der Kopf enthält ein Ziel-Implementations-Paar. Das Ziel befindet sich in der Ellipse, seine Implementation in dem Rechteck. Der Regelkörper enthält ein oder eine Konjunktion mehrerer Ziel-Implementations-Paare. E1 aus Abb. 12 ist eine Zielelaborationsregel, da sie das Aufgabenziel "absdiff (S1, S2)" (= Programm zur Berechnung der absoluten Differenz zweier Zahlen) in die Zielstruktur "branching (less_than (...) ...)" überführt. O1 ist eine Implementationsregel, da sie die Implementation des Ziels "branching (...)" durch den if-then-else-Knoten beschreibt. Die einfachen Regeln aus Abb. 12 sind folgendermaßen zu lesen:

> E1: (*Regelkopf)*: Wenn das Ziel "absdiff" mit zwei beliebigen Subzielen S1 und S2 ist,
> dann lasse Platz für einen noch zu programmierenden Baum ("Program Tree")
> und (*Regelkörper*):
> wenn Du im nächsten Planungsschritt das Ziel "branching" mit den Subzielen "less_than (S1, S2)", "difference (S2, S1)" und "difference (S1, S2)" bildest,

dann ist der Programmbaum für dieses Ziel ("Program Tree") auch die Lösung für das Ziel im Regelkopf.

O1: (*Regelkopf*): Wenn das Ziel "branching" mit drei beliebigen Subzielen "If", "Then" und "Else" ist,
dann programmiere den "if-then-else"-Knoten mit drei Verbindungslinien und lasse darüber Platz für drei noch zu programmierende Teilbäume (P1, P2, P3),
und (*Regelkörper*):
wenn Du im nächsten Planungsschritt das Ziel "If" verfolgst,
dann ist seine Lösung P1 auch der Teilbaum P1 im Regelkopf, und
wenn Du im nächsten Planungsschritt das Ziel "Then" verfolgst,
dann ist seine Lösung P2 auch der Teilbaum P2 im Regelkopf, und
wenn Du im nächsten Planungsschritt das Ziel "Else" verfolgst,
dann ist seine Lösung P3 auch der Teilbaum P3 im Regelkopf.

Wenn die Regeln (einfache Regeln und Komposita) als Horn-Klauseln (Kowalski, 1979) betrachtet werden, dann kann die Komposition zweier Regeln RI und RJ zu dem Kompositum RIJ durch die Resolutionsregel beschrieben werden (Hofbauer & Kutsche, 1989, S. 42):

$$\frac{RI: (F \leftarrow P \,\&\, C) \qquad RJ: (P' \leftarrow A)}{RIJ: (F\pi \leftarrow A \,\&\, C\pi)\sigma}$$

A und C sind Konjunktionen atomischer Formeln. P, P' und F sind atomische Formeln. π ist eine Umbenennung, so daß $(F \leftarrow P \,\&\, C)\pi$ und $(P' \leftarrow A)$ variablendisjunkt sind. σ ist der "most general unifier" (mgu) von {P, P'}. Z.B. kann das Kompositum C7 in Abb. 13 durch sukzessive Komposition gemäß der o.g. Inferenzregel mit der Menge einfacher Regeln {O1, O5, L1, L2} als Ausgangspunkt erzeugt werden:

```
O1: gmr (branching (If, Then, Else), ite-pop (P1, P2, P3)) :-
        gmr (If, P1), gmr (Then, P2), gmr (Else, P3). (siehe auch Abb. 12)
O5: gmr (equal (S1, S2), eq-pop (P1, P2)) :- gmr (S1, P1), gmr (S2, P2).
L1: gmr (parm (P), P-pl) :- is_parm (P).
L2: gmr (const (C), C-cl) :- is_pconst (C).
C7: gmr (branching (equal (parm (Y), const (C)), parm (X), Else),
                ite-pop (eq-pop (Y-pl, C-cl), X-pl, P)) :-
        is_parm (Y), is_const (C), is_parm (X), gmr (Else, P). (s. Abb. 13)
```

mit
- ite-pop: primitiver ABSYNT-Operator "if-then-else"
- eq-pop: primitiver ABSYNT-Operator "="
- P-pl: unbenannter ABSYNT-Parameterknoten
- C-cl: leerer ABSYNT-Konstantenknoten

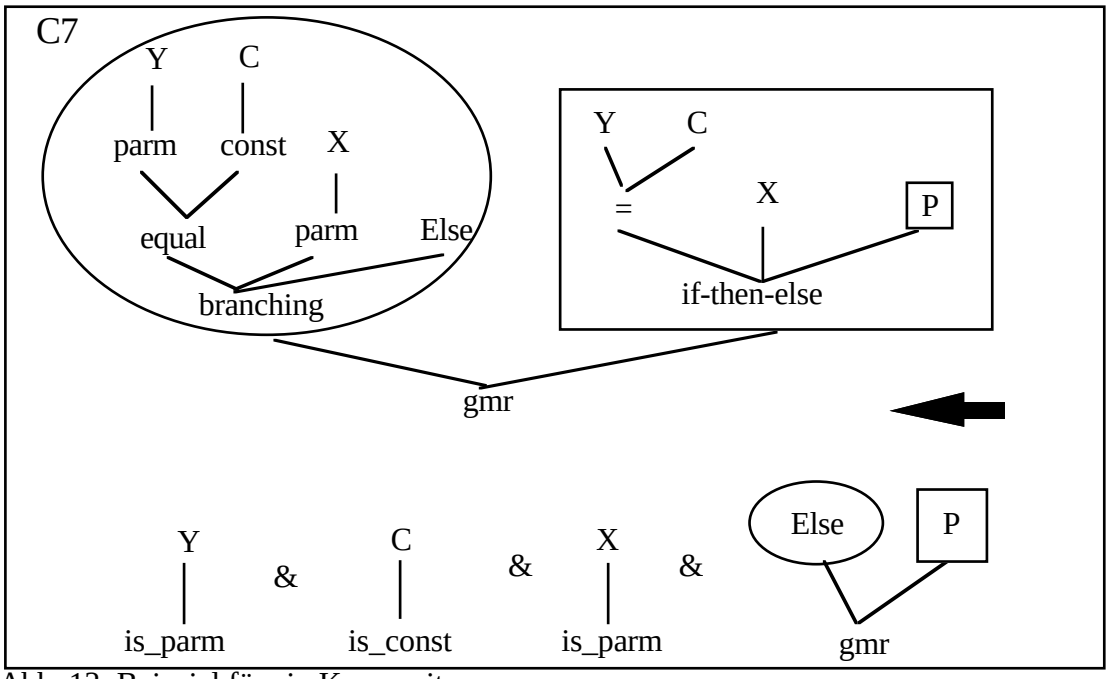


Abb. 13: Beispiel für ein Kompositum

C7 entsteht aus der Menge {O1, O2, L1, L2} auf folgende Weise: Die Komposition zweier Regeln RI und RJ zu RIJ kann abgekürzt dargestellt werden als

$$RIJ = RI_k \bullet RJ.$$

k bezeichnet die k-te Variable im Zielbaum des Regelkopfes von RI (z.B. $O1_3$ = "Else"). "•" bedeutet: Die k-te Variable im Zielbaum des Regelkopfes von RI wird durch den Zielbaum im Regelkopf von RJ ersetzt. Dann wird der Term im Regelkörper von RI, der die k-te Variable enthält und mit dem Regelkopf von RJ unifiziert werden kann, durch den Regelkörper von RJ ersetzt. Auf diesen resultierenden Term wird schließlich der mgu σ angewendet, was zum Kompositum RIJ führt. So ist z.B. $O1_2 \bullet L1$ =

```
C1: gmr (branching (If, parm (P), Else), ite-pop (P1, P, P3)) :-
        gmr (If, P1), is_parm (P), gmr (Else, P3).
```

Es gibt 16 Möglichkeiten, C7 aus {O1, O2, L1, L2} zu gewinnen. Ein Beispiel ist:
$C7 = (O1_2 \bullet L1)_1 \bullet ((O5_2 \bullet L2)_1 \bullet L1)$.

Abb. 14 zeigt den Aufbau des internen Modells (IM). Es enthält zwei Mengen von Regeln. Die Menge IM repräsentiert das jeweils aktuelle Domänenwissen des Lernenden. Die Menge POSS enthält mögliche Kandidaten für Komposita, die nach bestimmten Kriterien (s.u.) im IM aufgenommen werden. Nun zu dem Ablauf:

START: Zu Beginn der Bearbeitung von Programmieraufgaben sind IM und POSS leer.

i := 1: Der Lernende löst die 1. Aufgabe

1. TEST: Da IM und POSS leer sind, geschieht nichts.

1. PARSEN: Die Lösung zur 1. Aufgabe wird mit EXPERT-Regeln geparsed.

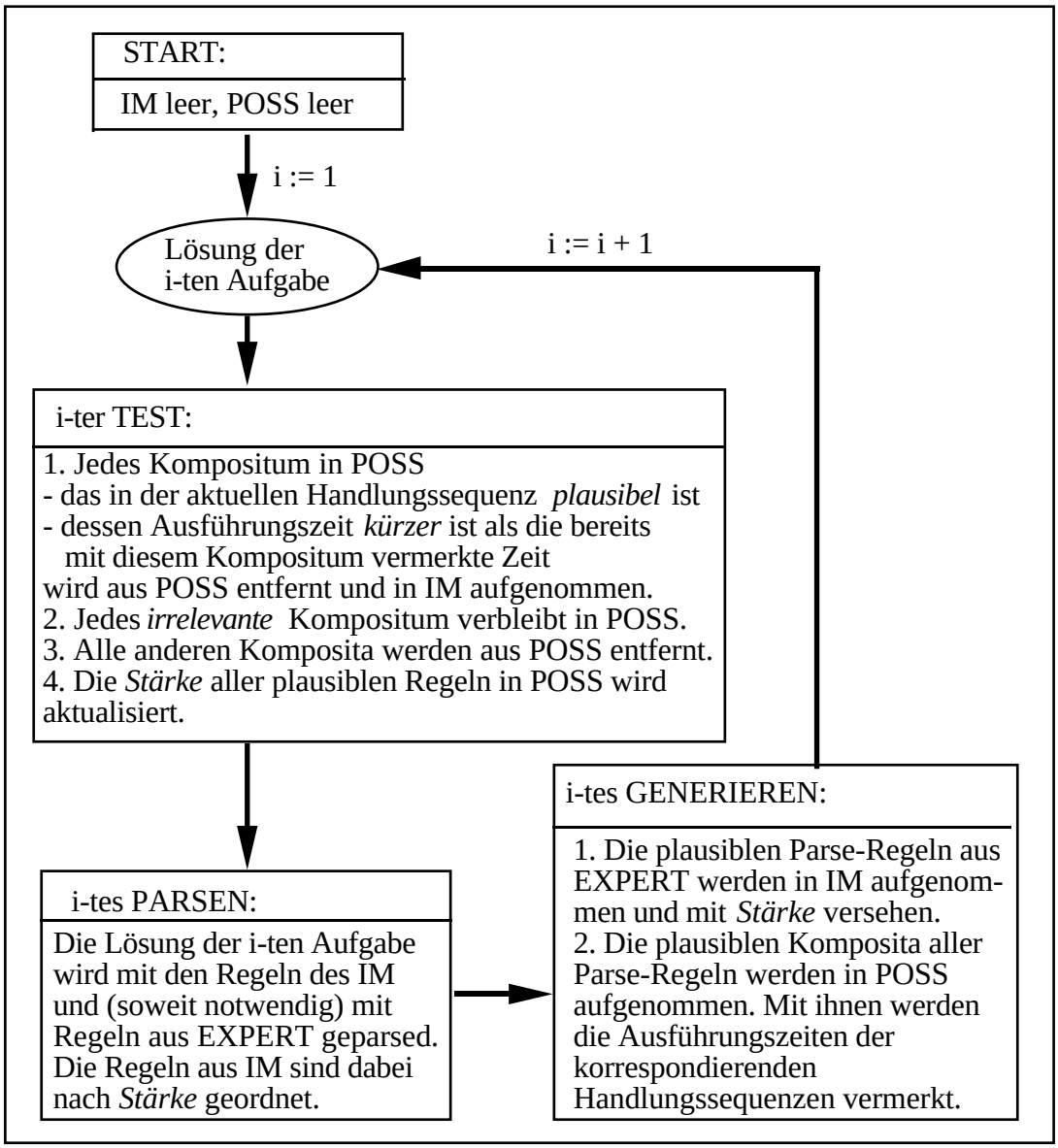


Abb. 14: Aufbau des internen Modells

1. GENERIEREN: Zunächst wird die Plausiblilität der gerade benutzten EXPERT-Regeln ermittelt. Eine Regel ist *plausibel*, wenn das Programmfragment in ihrem Regelkopf vom Lernenden in einer ununterbrochenen Sequenz programmiert wurde. (Diese steht als Handlungsprotokoll in einem Logfile zur Verfügung, das vom Rechner während der Aktionen des Lernenden online angelegt wird. Zielelaborationsregeln, wie E1 in Abb. 12, werden im internen Modell gegenwärtig nicht berücksichtigt.) Z.B. besteht das Programmfragment im Regelkopf von O1 (Abb. 12) aus einem "if-then-else"-Knoten und drei Verbindungslinien. Diese Regel ist plausibel, wenn der Lernende die entsprechenden vier Programmierhandlungen in ununterbrochener Sequenz ausgeführt hat. Hinsichtlich der Handlungssequenz in Abb. 15 ist O1 jedoch *unplausibel*: Die entsprechenden Programmierhandlungen wurden zwar ausgeführt, aber nicht in ununterbrochener Reihenfolge. Sie wurden in Abb. 15 zu den Zeitpunkten 11:15:52 (Positionierung des "if-then-else"-Knoens), 11:15:58, 11:16:46 und 11:16:55 (ziehen der drei Verbindungslinien vom "if-then-else"-Knoten aus) ausgeführt, also unterbrochen bei 11:16:42 und 11:16:50. Dagegen ist das oben genannte Kompositum C1 hinsichtlich dieser Handlungssequenz *plausibel*, da sein Regelkopf den if-then-else-Knoten mit einem Parameter an der zweiten Verbindungslinie enthält. Die entsprechende Handlungssequenz läuft in Abb. 15 von 11:15:52 bis 11:16:55 ab.
Die plausiblen EXPERT-Parse-Regeln werden in das IM übernommen und bekommen einen Stärkewert: das Produkt aus der Häufigkeit, mit der eine Regel bei einem Lernenden schon plausibel war, und der Anzahl der durch die Regel erklärten

Programmierhandlungen. (Z.B. ist diese Anzahl für O1 gleich 4.) Damit ist die Stärke der Regel eine monotone Funktion ihrer bisherigen empirischen Bewährung.
Schließlich werden die Komposita aller gerade zum Parsen benutzten Regeln gebildet. Für jedes Kompositum wird geprüft, ob es plausibel ist. Wenn ja, dann wird es in POSS aufgenommen, und die vom Lernenden benötigte Ausführungszeit für die entsprechende Handlungssequenz wird mit diesem Kompositum vermerkt. (Für C1 würden bei der Sequenz in Abb. 15 die Zeit von 11:15:43 bis 11:16:55 vermerkt werden, also 72 Sek.)

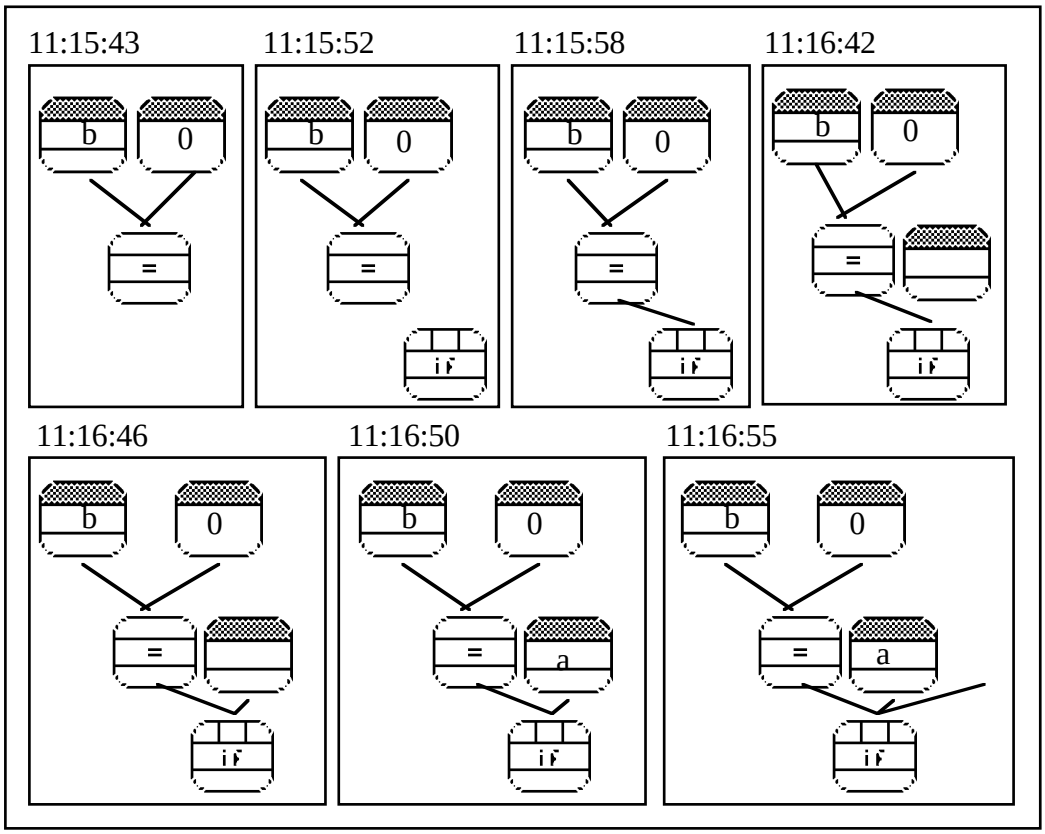


Abb. 15: Ausschnitt aus der Handlungssequenz einer Versuchsteilnehmerin

i := 2 (i := i+1): Der Lernende löst die zweite Aufgabe.

2. TEST: Jedes Kompositum in POSS wird daraufhin überprüft, ob
- es im Hinblick auf die Handlungssequenz des Lernenden zur 2. Aufgabe plausibel ist
- die ihm entsprechende Handlungssequenz in dem Protokoll schneller ausgeführt wurde als die mit diesem Kompositum in POSS vermerkte Zeit.

Die Komposita, die diese Voraussetzungen erfüllen, werden in das IM aufgenommen. Für die anderen Komposita in POSS gilt:
Ein Kompositum, dessen Regelkopf ein Programmfragment enthält, das in der aktuellen Lösung nicht enthalten (einbettbar) ist, ist im Hinblick auf die aktuelle Lösung irrelevant. Es verbleibt in POSS. Alle anderen Komposita entweder unplausibel (einbettbar, aber nicht in ununterbrochener Sequenz programmiert), oder der Lernende benötigte für die entsprechende Handlungssequenz mehr Zeit als die dem Kompositum zugeordnete Zeit. Diese Komposita werden aus POSS entfernt.
Schließlich wird für alle Regeln (einfache Regeln und Komposita) im IM, die im Hinblick auf die aktuelle Handlungssequenz plausibel sind, der Stärkewert aktualisert, indem er um die Anzahl der durch die Regel erklärten Programmierhandlungen erhöht wird.

2. PARSEN: Die Lösung der 2. Aufgabe wird nun mit den Regeln im IM geparsed, wobei diese Regeln nach Stärke geordnet sind. Auf EXPERT wird nur zugegriffen, wenn die Regeln im IM für den Parsevorgang nicht ausreichen.

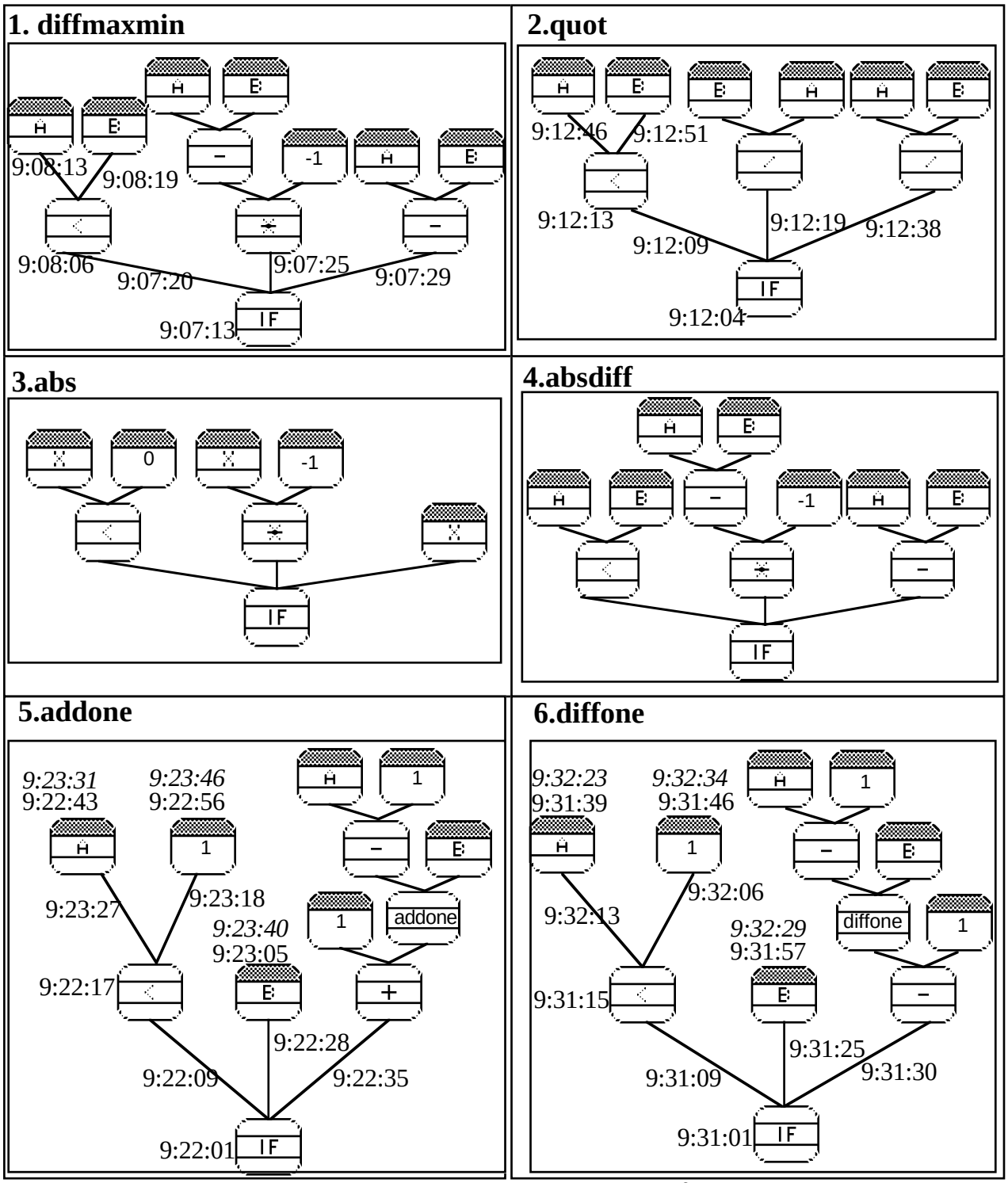


Abb. 16: Künstlich erstellte, z.T. zeitprotokollierte Lösungsentwürfe (Programmkörper) zu 6 aufeinanderfolgenden ABSYNT-Programmieraufgaben

Der Prozeß setzt sich nun fort wie oben beschrieben. Wir wollen das interne Modell an einem Beispiel illustrieren. Abb. 16 zeigt eine künstliche Sequenz von Programmierhandlungen. Es sind die Körperbäume der Lösungen zu 6 aufeinanderfolgenden Programmieraufgaben dargestellt: "diffmaxmin" (Differenz des Maximums und Minimums zweier Zahlen), "quot" (Division der größeren durch die kleinere von zwei positiven Zahlen), "abs" (Absolutbetrag einer Zahl), "absdiff" (wie "diffmaxmin": Absolutbetrag der Differenz zweier Zahlen), "addone" (Addition durch "+1") und "diffone" (Subtraktion durch "-1"). Teile der Programmbäume in Abb. 16 sind mit dem Zeitpunkt ihrer Herstellung auf dem Bildschirm versehen. (So wurde um 9:32:13 Uhr die Verbindung vom "<"-Knoten zum Parameter A in der Lösung zu "diffone" hergestellt. Die kursiv gedruckten Zeiten stehen wieder für Knotenbeschriftungen.)

Das IM enthält einfache Regeln ("Mikroregeln") und Komposita. Innerhalb der Komposita unterscheiden wir *Problemlöseschemata* und *Fälle*. Problemlöseschemata sind Komposita, die mindestens eine Variable enthalten, die an einen Teilbaum gebunden werden kann. (Im Körper eines solchen Kompositums befindet sich daher noch mindestens ein "gmr"-Aufruf.) C7 (Abb. 13) ist ein Beispiel für ein Problemlöseschema, da die Variable P an einen beliebigen Teilbaum gebunden werden kann. Ein *Fall* ist dagegen ein "vollständig auskomponiertes" Kompositum. Es enthält nur noch Variablen für Parameter und Konstanten.

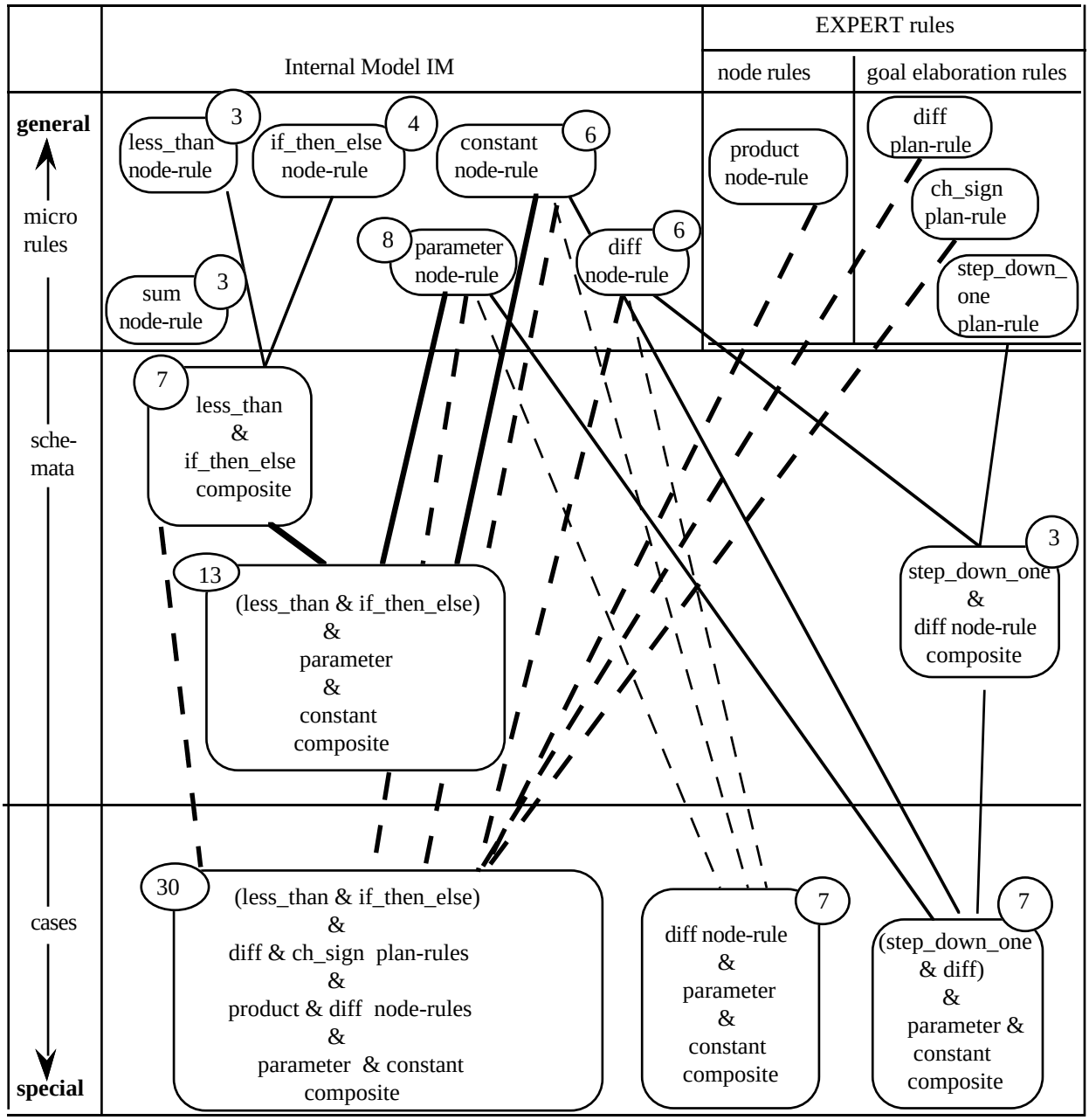


Abb. 17: Partielle Ordnung (Spezialisierungsgraph) der Mikroregeln, Schemata und Fälle im IM nach Erstellung der 6 Lösungen in Abb. 16

Die Mikroregeln, Schemata und Fälle im IM bilden einen Spezialisierungsgraphen. Abb. 17 stellt diesen Graphen nach der Lösung der letzten Aufgabe ("diffone") in der Aufgabensequenz von Abb. 16 dar. Die Ziffern in Kreisen geben die aktuellen Stärkewerte der Regeln an. Jedes Kompositum in Abb. 17 ist mit den Regeln verbunden, aus denen es gebildet wurde. So ist z.B. das "(less_than & if_then_else) & parameter & constant"-Kompositum in Abb. 17:

gmr (branching (less_than (parm (Y), const (C)), parm (X), Else),

```
        ite-pop (lt-pop (Y-pl, C-cl), X-pl, P)) :-
is_parm (Y), is_const (C), is_parm (X), gmr (Else, P).
```

("lt-pop" bezeichtet den primitiven ABSYNT-Operator "<")

Dieses Kompositum resultiert aus der Komposition des "less_than & if_then_else"-Kompositums

```
gmr (branching (less_than (S1, S2), Then, Else),
        ite-pop (lt-pop (P1, P2), P3, P4)) :-
gmr (S1, P1), gmr (S2, P2), gmr (Then, P3), gmr (Else, P4).
```

mit den weiter oben dargestellten Regeln L1 (parameter node rule) und L2 (constant node rule) für Parameter und Konstanten:

"(less_than & if_then_else) & parameter & constant"-Kompositum =
$(("less\_than\ \&\ if\_then\_else"\text{-}Kompositum_1 \bullet L1)_1 \bullet L2)_1 \bullet L1_1$.

Vor der Bearbeitung der ersten Aufgabe ("diffmaxmin") der Aufgabensequenz in Abb. 16 ist das IM leer. Die Lösung zu "diffmaxmin" wird daher mit EXPERT-GMR-Regeln geparsed. Die "if_then_else-node"-Regel (Implementationsregel O1, s. Abb. 12) und die "less_than-node"-Regel sind unter den Regeln, die die Lösung zur ersten Aufgabe (Abb. 16 oben links) parsen. Die in die "diffmaxmin"-Lösung eingetragenen Zeiten zeigen, daß diese Regeln plausibel sind: Der if_then_else-Knoten und die drei Verbindungen zu seinen Eingängen wurden in ununterbrochener Abfolge programmiert (9:07:13; 9:07:20; 9:07:25; 9:07:29). Entsprechendes gilt für den "<"-Knoten und die beiden Verbindungslinien zu seinen Eingängen (9:08:06; 9:08:13; 9:08:19). Daher wird die "if_then_else-node"-Regel mit der Stärke 4 und die "less_than-node"-Regel mit der Stärke 3 in das IM aufgenommen. Ein aus den Parseregeln gebildetes Schema ist das "less_than & if_then_else"-Kompositum. Es erklärt die zusammenhängende Sequenz der 7 Programmierhandlungen von 9:07:13 bis 9:08:19 und wird daher mit dem Vermerk "66 Sekunden" in POSS aufgenommen.
Nach der Lösung zu "quot" ist das "less_than & if_then_else"-Kompositum wiederum plausibel. Dieses Mal dauert die korrespondierende Handlungssequenz nur 47 Sekunden (von 9:12:04 bis 9:12:51). Daher kommt dieses Kompositum mit einer Stärke von 7 in das IM.
Als weiteres Beispiel werden die dem "(less_than & if_then_else) & parameter & constant"-Kompositum entsprechenden 13 Programmierhandlungen zusammenhängend ausgeführt (Aufgabe "addone", von 9:22:01 bis 9:23:46, = 105 Sekunden). Daher kommt dieses Schema in POSS. Bei der Aufgabe "diffone" ist es wieder plausibel, und die korrespondierende Handlungssequenz ist schneller (von 9:31:01 bis 9:32:34, = 93 Sekunden). Daher kommt dieses Schema mit der Stärke 13 in das IM.
Abb. 17 zeigt auch, daß Komposita im IM enthalten sein können, aber nicht alle Mikroregeln, aus denen sie gebildet wurden. Z.B. die "product-node"-Regel nicht im IM.

Abschließend sollen einige Aspekte des internen Modells betrachtet werden:

• *Regel-Schema-Fall-Kontinuum.* Komposita stellen im Vergleich zur entsprechenden Kette einfacher Regeln insofern eine Optimierung dar, weil sie weniger Zwischenziele enthalten und weniger Kontrollentscheidungen (Regelauswahlprozesse) notwendig machen. Aber nicht nur einfache Regeln, sondern auch Komposita können "komponiert" werden, so daß eine Hierarchie von Komposita entsteht. Komposita reichen von Problemlöseschemata bis hin zu spezifischen Lösungsbeispielen. Statt zwischen "Regel" einerseits und "Beispiel" andererseits zu dichotomisieren (z.B. Slade, 1991), spannen einfache Regeln und Komposita also ein *Kontinuum* von Mikroregeln über Problemlöseschemata zu Beispielen für Teillösungen und komplette Lösungen (Fallbeispiele) auf. Struktur- bzw. Rekursionsschemata, wie sie von Vorberg und Mitarbeitern vorgeschlagen wurden (Haack, Hahn & Wagner, 1988; 1989; Vorberg,

1989; Vorberg & Göbel, 1989), sind ebenfalls auf der Ebene der Problemlöseschemata anzusiedeln.
Das Wissen des Novizen besteht in dieser Konzeption zunächst aus *Mikroregeln*, die nur jeweils eine oder sehr wenige Programmierhandlung(en) erzeugen (Möbus, Schröder & Thole, 1991b). Ihre Anwendung erfordert viele Zwischenziele und Kontrollentscheidungen. Diese werden mit zunehmender Expertise aufgrund von Schema- und Fallbildung nach und nach überflüssig. Diese Sichtweise des Übergangs vom Novizen zum Experten entspricht weitgehend der von z.B. Chase & Simon (1973), Simon & Simon (1978), im Bereich des Programmierens Adelson (1981), Gugerty & Olson (1986). Auch Haack, Hahn & Wagner (1989) interpretieren den Übergang vom Novizen zum Experten mit dem Erwerb bzw. mit der Verfügbarkeit von Strukturschemata. Die Annahme, daß sich das Wissen der Novizen angemessener durch elementare Programmierregeln ("Mikroregeln") als durch Schemata und Fälle beschreiben läßt, wird auch durch eigene empirische Beobachtungen gestützt: Wenn die gleiche Aufgabe nacheinander in unterschiedlicher Formulierung dargeboten wurde, dann entwickelten unsere Versuchsteilnehmer völlig unterschiedliche Lösungen. Dieses Ergebnis ist mit der Annahme der Verfügbarkeit weniger Schemata oder gar "Fälle" schwer vereinbar.
Dieser Aspekt läßt sich näher untersuchen, indem das IM zur Generierung von Hilfen (Hypothesenergänzungen oder visuelle Planungshilfen) herangezogen wird. Wenn das IM vorwiegend Mikroregeln enthält, repräsentiert es das hypothetische Domänenwissen eines Novizen. Wenn dieser Person nun "Fälle" als Hilfen vorgegeben werden, so erhält sie vorgefertigte (Teil-)Lösungen ohne Zielinformation. Wir erwarten dann, daß der Problemlöser die dargebotene Information entweder "ohne Nachdenken" übernimmt, oder daß er durch "Selbsterklärung" (Chi et al., 1989) versucht, sie zu verstehen.

• *Empirische Vorhersagen*. Auf der Basis der Regeln des internen Modells lassen sich spezifische Handlungs- und Verbalisationssequenzen sowie auch Stocksituationen vorhersagen:

1. Ein Kompositum enthält weniger Ziele als eine entsprechende Kette elementarerer Regeln. Bei der Anwendung des Kompositums wird daher weniger verbalisiert als bei der Anwendung der Regelkette.

2. Die Anwendung des Kompositums erfordert keine Kontrollentscheidungen. Dagegen ist jede Regel einer entsprechenden Regelkette mit einer Kontrollentscheidung verbunden, da jede dieser Regeln aus einer Menge von Alternativen zunächst ausgewählt werden muß. Die Anwendung des Kompositums ist daher schneller als die Anwendung der Regelkette. (Dieser Aspekt wird mit den Ausführungszeiten von Handlungssequenzen in dem internen Modell berücksichtigt.)

3. Wenn der Lernende eine Regel anwendet, dann - so unsere Hypothese - wird das im Regelkopf enthaltene Programmfragment in einer zusammenhängenden, ununterbrochenen Sequenz programmiert. Zusätzlich können die in dem Regelkopf enthaltenen Ziele verbalisiert werden. *Innerhalb* einer Regel ist jedoch für diese Zielverbalisationen und Programmierhandlungen keine Reihenfolge determiniert und damit nicht vorhersagbar. Die Reihenfolge der Verbalisationen und Handlungen, die eine Regel erklärt, ist nicht vorhersagbar. Es ist nur die *Menge* dieser Ereignisse vorhersagbar: *Nichtdeterminismus der Regelereignisse*. Dagegen ist für eine sukzessive Kette von Regeln eine *Sequenz* von Ereignismengen vorhersagbar. Für den Vergleich eines Kompositums mit einer entsprechenden Regelkette folgt daraus, daß die Reihenfolge der Ereignisse für die Kette in höherem Maße vorhersagbar ist. (Zu einer genauen Beschreibung siehe Möbus, Schröder & Thole, 1991b). Allgemein bedeutet dies: *Mit zunehmender Expertise des Lernenden wird die Reihenfolge seiner Handlungen immer schlechter vorhersagbar*.

4. *Stocksituationen* sind im Problemlöseprozeß immer dann zu erwarten, wenn für das Parsen der Lösung des Lernenden zusätzliche Regeln aus EXPERT benötigt wurden. Sie repräsentieren in diesem Fall das neu erworbene Wissen. Diese Hypothese kann weiter präzisiert werden: Stocksituationen sollten unmittelbar vor den Handlungssequenzen

aufgetreten sein, die mit den Programmfragmenten in den Regelköpfen dieser neu aufgenommenen EXPERT-Regeln korrespondieren. Indikatoren für Stocksituationen sind z.B. Sprechpausen und negative Äußerungen (van Lehn, 1991).

• *Bezug des internen Modells zur ISPDL-Theorie*. Die vom internen Modell realisierten Aspekte der ISPDL-Theorie sind in Abb. 2 bis 5 schattiert dargestellt. Das Wissen im IM (einfache Regeln und Komposita) ist in der "Knowledge Base" der Abb. 2 bis 4 enthalten.
Zu Abb. 2: Mit den Regeln im IM überführt der Lernende das Aufgabenziel ("Goal") über "Goal Processing" in eine Lösung ("Solution"). Neu in das IM aufgenommene Komposita sind Resultat der "deductive knowledge optimization".
Zu Abb. 3: Das Ziel wird ausgeführt. Das interne Modell berücksichtigt nur "operationale Ziele". Die Reaktion auf einen Impasse ("inductive knowledge acquisition") besteht in der Aufnahme neuer Regeln aus EXPERT im IM, wenn diese für den Parseprozeß und damit für die Erklärung des Problemlöseprozesses notwendig sind.
Zu Abb. 4: Die operationale Zielverarbeitung besteht in der Ausführung von Domänenregeln im IM ("execute: operator"). Der dieser Ausführung zugrundeliegende Lösungsplan ("Plan") wird im internen Modell in Form des Parsebaums rekonstruiert. Das "Protocol" entspricht der aus der Ausführung resultierenden zeitprotokollierten Handlungssequenz des Problemlösers, die im internen Modell zur Ermittlung plausibler Regeln herangezogen wird.

• *Ableitung von Hilfen*. Das interne Modell ermöglicht die Bereitstellung wissensstandsangepaßter Ergänzungsvorschläge und Planungshilfen. So kann ein Ergänzungsvorschlag auf der Grundlage der Regel im IM mit höchstem Stärkewert generiert werden, die die Hypothese in Richtung auf eine korrekte Lösung ergänzt. Auf der Ebene von Planungshilfen kann die entsprechende visuelle Regel angeboten oder empfohlen werden. Da in diesen Fällen das erforderliche Domänenwissen im IM bereits enthalten ist, führen wir die aktuelle Stocksituation des Problemlösers auf fehlendes oder falsch eingesetztes Kontrollwissen zurück. Dies zu präzisieren, ist Aufgabe des externen Modells.
Es ist jedoch möglich, daß ein Ergänzungsvorschlag oder eine Planungshilfe auf der Grundlage des IM nicht gegeben werden kann, weil eine entsprechende, in Richtung auf eine Lösung führende Regel noch nicht im IM enthalten ist. In diesem Fall basiert die Stocksituation des Problemlösers auf fehlendem Domänenwissen. Für die Generierung eines Ergänzungsvorschlags oder einer Planungshilfe muß dann auf eine Regel in EXPERT zurückgegriffen werden.
In beiden Fällen ist es möglich, auf der Basis des IM Hilfeinformationen im Hinblick auf Körnigkeit (Kompositionsgrad) und Informationsmenge gezielt zu variieren und spezifische Hypothesen zur differentiellen Wirksamkeit von Information zu untersuchen, die in unterschiedlichem Maße wissensstandsbezogen bzw. wissensdefizitbezogen ist (siehe hierzu auch weiter unten).

• *Bezug zu anderen Arbeiten.* Ein Beispiel für einen Ansatz mit z.T. ähnlicher Zielsetzung ist die kognitive Diagnose und das episodische Lernermodell im ELM-LISP-Tutor (Bögelsack & Weber, 1989; Müller & Weber, 1991; Weber, 1989; Weber & Bögelsack, 1988), der als Intelligentes Tutorielles System zum Erlernen der Programmiersprache LISP konzipiert ist. Es werden Schülerlösungen zu vorgegebenen Aufgaben analysiert, und es werden wissensstandsabhängige, abgestufte Hilfen und Rückmeldungen gegeben. Den Kern des ELM-LISP-Tutors bilden ein Diagnosesystem und ein episodisches Lernermodell. Das Diagnosesystem besteht aus einer Reihe von Frames, die verschiedene Programmierkonzepte repräsentieren, welche im Bereich LISP von Bedeutung sind. Jedem Konzept ist eine Menge von Regeln zugeordnet, die verschiedene Realisierungen dieses Konzepts darstellen. Die Regeln beschreiben die Realisierung des Konzepts entweder durch Subkonzepte, welche durch weitere Frames spezifiziert sind, oder durch primitive LISP-Funktionen. Jede Regel ist mit der Bewertung "gut", "schlecht" oder

"falsch" versehen. Die Konzept-Frames enthalten also auch Fehlerregeln. Neben den Regeln enthält jeder Konzept-Frame zusätzlich mögliche Transformationen des Konzepts. Bei der Diagnose einer Schülerlösung wird diese mit den Konzept-Frames sowie ggf. mit episodischen Frames (s.u.) geparst. Als Ergebnis des Parsens wird die Hierarchie der verwendeten Konzepte, Regeln und ggf. Transformationen als Ableitungsbaum ausgegeben. Jede Konzept-Frame-Instanz in diesem Ableitungsbaum wird als episodischer Frame festgehalten, der u.a. besagt, mit welcher Regel das betreffende Konzept realisiert wurde, welchem Programmfragment in der aktuell vorliegenden Schülerlösung es entspricht, und ob es sich um eine gute, schlechte oder falsche Realisation handelt. Die Menge der aktuell vorliegenden episodischen Frames bildet das episodische Lernermodell (ELM). Da das ELM bei dem Parsevorgang mit herangezogen wird, zieht die Diagnose bereits früher von dem Schüler beschrittene Lösungswege auch bevorzugt in Erwägung. Da jeder Konzeptframe mehrere Regeln enthält, kann die Diagnose verschiedene Ableitungsbäume für denselben Lösungsentwurf generieren und so zu alternativen Erklärungen einer Lösung (bzw. eines Fehlers) gelangen.
Die Konzept-Frames im ELM-LISP-Tutor weisen mit den ABSYNT-Regeln in EXPERT einige Gemeinsamkeiten auf: Ähnlich wie im ELM-LISP-Tutor jedes Konzept durch verschiedene Regeln realisiert werden kann, so enthält die Ziel-Mittel-Relation in ABSYNT für jedes Programmierziel alternative Realisationen. Analog zur Konzepthierarchie bildet die Ziel-Mittel-Relation eine Ziel-Subziel-Hierarchie. Den Transformationsregeln in den Konzeptframes entsprechen Regeln in der Ziel-Mitttel-Relation. Ferner entspricht dem Ableitungsbaum der ELM-LISP-Diagnose ein Parsebaum in der Ziel-Mittel-Relation.
Das ELM und das interne Modell in ABSYNT weisen demgegenüber größere Unterschiede auf. Während im ELM Frame-Instanzen abgelegt sind, enthält das Stadienmodell uninstantiierte Regeln sowie Komposita als Resultate von Wissensoptimierungsprozessen. Komposita sind im ELM nicht enthalten. Ein weiterer Unterschied betrifft die Fehlerregeln. Diese sind in den Konzeptframes als "typische" LISP-Fehlkonzepte fest vorgegeben und gehen in das ELM ein, soweit sie zur Erklärung von Programmierfehlern herangezogen wurden. Dagegen enthält das interne Modell zunächst keine malrules. Malrules können aber im Zuge des Hypothesentestens aus fehlerhaften Lösungsentwürfen online erzeugt werden (Schröder, Möbus & Thole, 1991a).
Der ELM-LISP-Tutor sieht ferner eine vierstufige Hilfe vor. Der Lernende kann zwischen diesen Stufen selbst wählen. Abgestufte Hilfen sind in ABSYNT ebenfalls möglich. Allerdings werden Fehler in ABSYNT nicht vom System lokalisiert: Wie schon ausgeführt, wird die Entscheidung darüber, welche Programmteile als korrekt beibehalten werden sollen, dem Lernenden überlassen. Neben Ergänzungsvorschlägen kann sich der Lernende wie im ELM-LISP-Tutor auch eine komplette Lösung zeigen lassen.

Das externe Modell (Prozeßmodell)

Der gegenwärtige Stand des externen Modells besteht aus einer Menge von Strukturen und Prozessen, die folgende Arten von Wissen bzw. von Informationen repräsentieren:

• aufgabenübergreifendes Wissen des Lernenden: Problemlöseheuristiken und der aktuelle Stand des Domänenwissens

• aufgabenbezogenes Wissen des Lernenden, bzw. aufgabenbezogene Ereignisse im Problemlöseprozeß: die Repräsentation des Aufgabentexts und des aktuellen Lösungsplans, sowie Stocksituationen

• Aspekte der Umgebung: die objektiv vorliegende Aufgabe, der aktuelle Stand des Lösungsentwurfs auf dem Bildschirm, sowie Hilfen.

Das externe Modell dient

• der Präzisierung und ggf. Weiterentwicklung der ISPDL-Theorie

- der Bereitstellung möglicher Erklärungen für die durch das interne Modell beschriebenen Wissensveränderungen.

Das externe Modell beruht auf der Analyse der Verbal- und Handlungsprotokolle zweier Personen, die in Einzelsitzungen eine Sequenz von etwa 20 ABSYNT-Programmieraufgaben bearbeiteten. Zunächst soll unter Orientierung an der ISPDL-Theorie (Abb. 2 bis 5) ein Überblick gegeben werden. Dann wird die Arbeitsweise des Modells mit einigen empirischen Daten aus den erwähnten Einzelversuchen verglichen.

Das externe Modell läuft in folgenden Phasen ab (Abb. 18):

• *Verstehen der Aufgabe*: Nach dem Modell beginnt der Problemlöseprozeß in Abb. 2 mit der Stelle "Goal". Der Prozeß des Abwägens zwischen verschiedenen Aufgaben und der Auswahl einer Aufgabe ("deliberate") ist im externen Modell gegenwärtig nicht enthalten und auch nicht notwendig, da die Aufgaben von außen vorgegeben werden.
Wie der obere Bereich von Abb. 18 zeigt, wird der Aufgabentext dem Modell als Textgraph dargeboten. Der Textgraph repräsentiert die objektiv vorliegende Aufgabe. Das Modell bildet daraus eine propositionale Aufgabenrepräsentation (van Dijk & Kintsch, 1983; Kintsch & Greeno, 1985), d.h. die Aufgabe wird "verstanden".
Abb. 19 zeigt die lineare Notation des Textgraphen und die vom Modell erzeugte propositionale Repräsentation für die Aufgabe "quot": "Programm, das den Quotienten zweier positiver Zahlen so berechnet, daß dieser Quotient größer oder gleich 1 ist". Die propositionale Repräsentation stellt das Aufgabenziel dar. Durch den Aufgabentext werden Konzepte angesprochen, wie "teilen_durch", "größer_gleich_pos" (Größenvergleich für positive Zahlen), usw. Die propositionale Aufgabenrepräsentation enthält
- die angesprochenen Konzepte
- für jedes Konzept die Argumente, z.B. bedeutet "hat_Nachfolger (1, (2, 3))": Das Konzept 1 "so_berechnen_daß" hat als Nachfolger das Konzept 2 "teilen_durch" und das Konzept 3 "größer_gleich_pos"
- für jedes Konzept die Information, ob es direkt in Programmcode umgesetzt werden kann ("zu_implementieren"), oder ob zunächst weitere Planungsschritte erforderlich sind ("zu_planen"). Ein Beispiel hierzu wird unten gegeben.

Wenn während der Bildung der propositionalen Aufgabenrepräsentation mehr als 7 Konzepte aktiviert werden, signalisiert das Modell eine Stocksituation: "Text zu lang" (Abb. 18, rechte Hälfte). Die Wahl von gerade 7 Konzepten entspricht unseren Beobachtungen: Wenn der Aufgabentext diese Länge überschritt, dann traten beim Lesen der Aufgabe in mehreren Fällen Probleme auf. Im Anschluß an diese Stocksituation wird nach dem Modell die Aufgabenstellung in einen ersten Teil und einen Rest zerlegt. Diese Teilaufgaben werden dann nacheinander abgearbeitet.

• *Operationale Zielverarbeitung*. Wenn eine Aufgabenrepräsentation gebildet worden ist, wird ein Lösungsplan synthetisiert, ausgeführt und das Ergebnis bewertet (Abb. 18, linke Hälfte; vgl. auch Abb. 4). Die Synthetisierung kann prinzipiell durch Verwendung von Analogien umgangen werden. Wir haben diesen Aspekt aber ausgeklammert. Nichtoperationale Zielverarbeitung (Abb. 5) ist in dem Modell ebenfalls noch nicht enthalten. Zunächst wird der Ablauf der operationalen Zielverarbeitung ohne Stocksituationen beschrieben:
Zuerst werden die in der Aufgabenrepräsentation angesprochenen Konzepte aktualisiert. Damit wird *Konzeptwissen* verfügbar, das zum Aufbau eines Lösungsplans benutzt wird. Das Konzeptwissen enthält die Information aus den Regeln der Ziel-Mittel-Relation. Es repräsentiert also das aktuelle Domänenwissen des Lernenden und beschreibt, wie ein Konzept als ABSYNT-Programmfragment implementiert werden kann. Abb. 20 zeigt die Konzepte "teilen_durch", "Vorzeichen_vertauschen" und "größer_gleich_pos". S1 und S2 sind Variablen für Subziele (fett gedruckt), P1 und P2

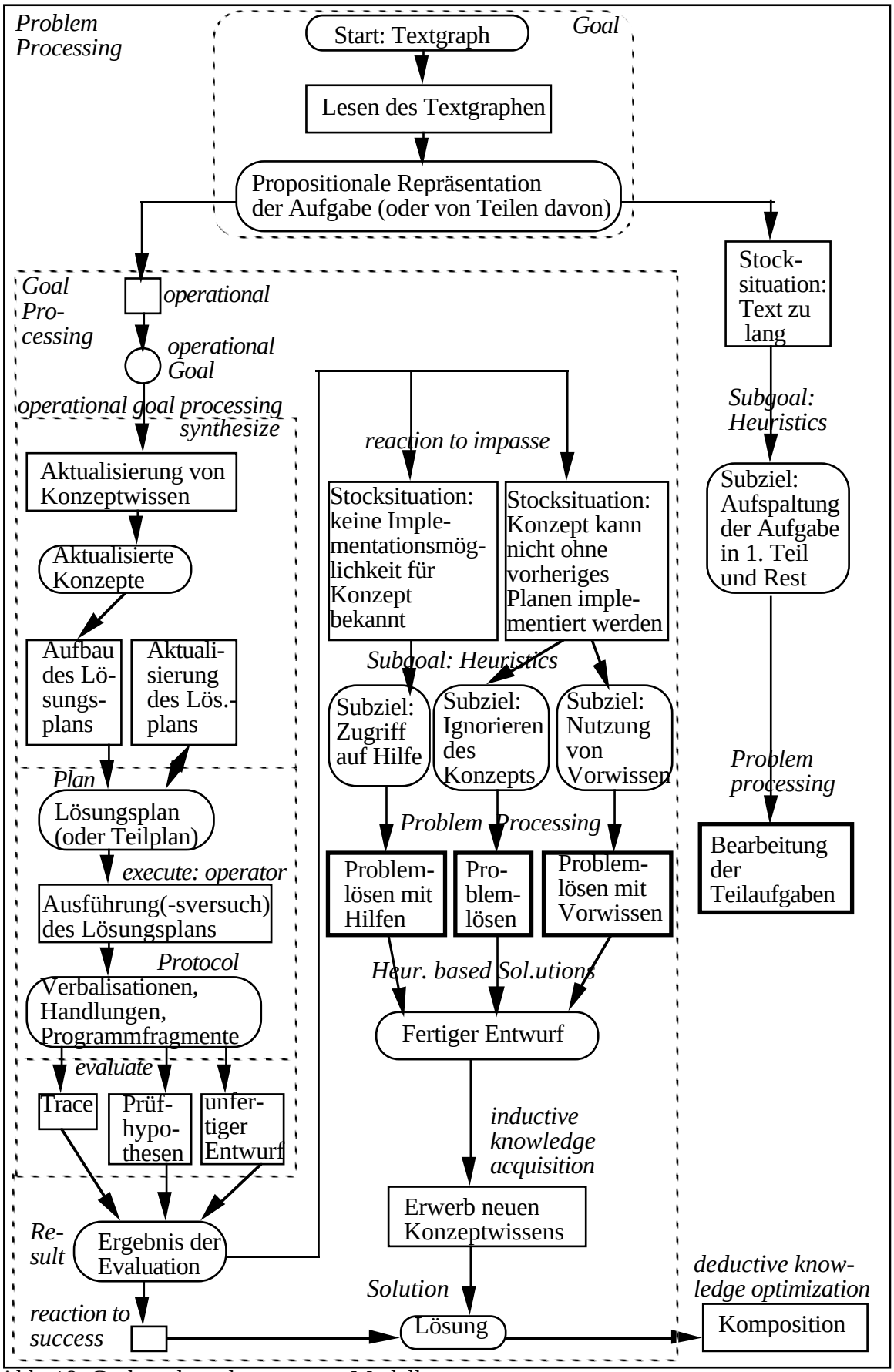


Abb. 18: Grobstruktur des externen Modells

sind Variablen für ABSYNT-Teilbäume. "teilen_durch" hat als Argumente zwei Subziele S1 und S2 und kann mit dem Divisionsknoten "/" implementiert werden, wenn S1 und S2 durch die Teilbäume P1 und P2 implementiert werden. Die Konzepte

"Vorzeichen_vertauschen" und "größer_gleich_pos" haben hier entsprechend den beiden Spalten zwei bzw. drei alternative Realisationen. Die erste Realisation von "Vorzeichen_vertauschen" enthält den Vorzeichenvertauschungsknoten "<-->". Die zweite Realisation von "Vorzeichen_vertauschen" enthält dagegen keinen ABSYNT-Knoten, sondern die Definition des Konzepts durch Ausdifferenzierung (multiplizieren (S, minus_eins)). Bei der Anwendung dieser Konzeptdefinition bzw. Ausdifferenzierung für die Entwicklung eines Lösungsplans müssen zunächst diese weiteren Konzepte ("multiplizieren", "minus_eins") aktualisiert werden. Schließlich können in jedem Konzept besondere Merkmale festgehalten werden. Z.B. enthält das Konzept "größer_gleich_pos" allgemeines Vorwissen über dieses Konzept in dem slot "besondere_Merkmale".

Aufgabe:
"Programm, das den Quotienten zweier (positiver) Zahlen so berechnet, daß dieser Quotiont größer oder gleich eins ist"

Textgraph:	*Propositionale Aufgabenrepräsentation:*
0. Programm (quot, 1)	
1. so_berechnen_daß (2, 3)	1. zu_planen (so_berechnen_daß)
2. Quotient (4, 5)	2. zu_implementieren (teilen_durch)
3. größer_gleich (2, 6)	3. zu_implementieren (größer_gleich_pos)
4. Zahl	4. zu_implementieren (Zahl_1)
5. Zahl	5. zu_implementieren (Zahl_2)
6. eins	6. zu implementieren (eins)
	hat_Nachfolger (1, (2, 3))
	hat_Nachfolger (2, (4, 5))
	hat_Nachfolger (3, (2, 6))

Abb. 19: Aufgabe, entsprechender Textgraph und vom Modell gebildete propositionale Aufgabenrepräsentation

Konzeptwissen:

Konzept: **teilen_durch**
besondere_Merkmale: nichtkommutativ

Realisationen:	1.
Argumente:	**S1**, **S2**
Implementation:	/ (P1, P2)
wird_implementiert_durch:	**S1**, P1
wird_implementiert_durch:	**S2**, P2

Konzept: **Vorzeichen_vertauschen**
besondere_Merkmale: ---

Realisationen:	1.	2.
Argumente:	**S**	**S**
Implementation:	<--> (P)	P
wird_implementiert_durch:	**S**, P	**multiplizieren (S, minus_eins)**, P

Konzept: **größer_gleich_pos**
besondere_Merkmale: wenn_dann_sonst(
größer_gleich (teilen_durch (X, Y), eins),
größer_gleich (X, Y),
größer_gleich (Y, X)).

Realisationen:	1.	2.	3.
Argumente:	**S1**, **S2**	**S1**, **S2**	**S1, S2**
Implementation:	>= (P1, P2)	<= (P1, P2)	P
wird_implementiert_durch:	**S1**, P1	**S1**, P2	**neg(größer_als (S2, S1)**, P
wird_implementiert_durch:	**S2**, P2	**S2**, P1	

Abb. 20: Beispiele für Konzeptwissen

Planelemente:

Planelement Nr: 2
Konzept: **teilen_durch**
Argumente: Planelement 4,
Planelement 5
ist_Argument_von: {Planelement 3}
Implementation: / (P1, P2)
wird_implementiert_durch
(Planelement 4, P1)
wird_implementiert_durch
(Planelement 5, P2)
Typ: zu_implementieren

Planelement: Nr: 4
Konzept: **Zahl_1**
Argumente: keine
ist_Argument_von: {Planelement 2}
Implementation: Parameter a
Typ: zu_implementieren

Abb. 21: Beispiele für Planelemente

Die Aufgabenrepräsentation und das angesprochene Konzeptwissen sind die Ausgangspunkte für die *Entwicklung des Lösungsplans* für die aktuell vorliegende Aufgabe (s. a. Abb. 18, links). Abb. 21 zeigt zwei Planelemente bei der quot-Aufgabe. Sie bestehen aus Frames (Anderson, 1989), die aufgebaut und dann aktualisiert werden. Die Zusammenhänge zwischen den Planelementen werden in die slots "Argumente", "ist_Argument_von" und "wird_implementiert_durch" eingetragen. Diese Zusammenhänge zwischen den Planelementen beruhen auf der propositionalen Aufgabenrepräsentation und ggf. auf heuristischem Wissen, das nach Stocksituationen eingesetzt wird (s.u.). Zu den einzelnen slots:
- "Argumente" verweist auf andere Planelemente in der Argument-Reihenfolge. Die entsprechende Information stammt aus der propositionalen Aufgabenrepräsentation. Z.B enthält die Aufgabenrepräsentation "hat_Nachfolger (2, (4, 5))" (vgl. Abb. 19). Entsprechend verweist der Argument-slot des Planelements Nr. 2 (Abb. 21 links) auf die Planelemente 4 und 5.
- "ist_Argument_von" enthält die Menge der Planelemente, von denen das betreffende Planelement selbst Argument ist. Da z.B. die Aufgabenrepräsentation (Abb. 19) "hat_Nachfolger (3, (2, 6))" enthält, ist Planelement 2 Argument von Planelement 3.
- "Implementation" beschreibt, wie das Planelement implementiert werden kann. Z.B. wird das Planelement Nr. 2 in Abb. 21 durch den ABSYNT-Divisionsknoten implementiert. Diese Information stammt aus dem Implementations-slot des entsprechenden Konzepts (vgl. das Konzept "teilen_durch" in Abb. 20). P1 und P2 sind wieder Variablen für Teilbäume.
- "wird_implementiert_durch" gibt an, welche Argumente durch welche Teilbäume implementiert werden. Z.B. wird das Planelement 4 (= 1. Argument von Planelement 2, Abb. 21 links) durch den Teilbaum P1 implementiert.
- "Typ" besagt, ob das betreffende Planelement in Programmcode umzusetzen ist oder nicht. Diese Information wird ebenfalls der Aufgabenrepräsentation entnommen.

Bei der *Ausführung des Lösungsplans* werden weitere Frames erzeugt, welche ABSYNT-Knoten und ihre Verbindungen auf dem Bildschirm repräsentieren ("Bildschirmelemente", Abb. 22). Die Verbindungslinien auf der Bildscirmoberfläche werden durch die Einträge im slot "Eingänge" dargestellt.

Bildschirmelemente:

Bildschirmelement Nr: 2
Ort: Programmkörper
ABSYNT-Knoten: /
Eingänge: Bildschirmelement 4,
Bildschirmelement 5

Bildschirmelement Nr: 4
Ort: Programmkörper
ABSYNT-Knoten: Parameter a
Eingänge: keine

Abb. 22: Beispiele für Bildschirmelemente

Schließlich folgt die *Bewertungsphase*. Die mentale Ausführung der Evaluation des Programms und das Auswählen und Testen von Prüfhypothesen sind in dem Modell

bisher nicht realisiert. (Der Fall "unfertiger Entwurf" betrifft Stocksituationen, s.u.). Die mentale Evaluation des Programms soll auf folgende Weise modelliert werden: Anhand des Lösungsplans wird für eine Beispielwertemenge das zu erwartende Ergebnis berechnet. Dann wird für diese Beispielwerte die Abarbeitung des visuellen Trace der Programmierumgebung ausgewählt. Das dadurch erhaltene und das erwartete Berechnungsergebnis werden miteinander verglichen.

• *Operationale Zielverarbeitung mit Stocksituationen*: Bei der operationalen Zielverarbeitung sieht das Modell zwei mögliche Stocksituationen vor. Die *erste* Form der Stocksituation ist gegeben, wenn ein für die Lösungsplanung aktiviertes Konzept keine Realisationsmöglichkeit enthält. In diesem Fall ist das entsprechende ABSYNT-spezifische Implementationswissen noch nicht erworben. Es kann also nur ein unvollständiger Lösungsplan erzeugt und ein entsprechend unfertiger Entwurf (Abb. 18, links unten) implementiert werden. Dieser Stocksituation kann mit der Heuristik begegnet werden, auf Planungshilfen zuzugreifen. Diese entsprechen den Regeln der Ziel-Mittel-Relation und enthalten daher die gewünschte Information. Abb. 23 zeigt Beispiele. Für ein Konzept (hier "Minimum_bilden") sind mehrere alternative Planungshilfen möglich.

Planungshilfe zu "teilen_durch":	
Konzept: **teilen_durch** Argumente: **S1, S2** Implementation: / (P1, P2) wird_implementiert_durch: **S1,** P1 wird_realisiert_durch: **S2,** P2	
1. Planungshilfe zu "Minimum_bilden":	*2. Planungshilfe zu "Minimum_bilden":*
Konzept: **Minumum_bilden** Argumente: **S1, S2** Implementation: MIN (P1, P2) wird_implementiert_durch: **S1,** P1 wird_implementiert_durch: **S2,** P2	Konzept: **Minumum_bilden** Argumente: **S1, S2** Implementation: P wird_implementiert_durch: **wenn_dann_sonst** **(kleiner_gleich_pos (S1, S2), S1, S2)**, P

Abb. 23: Beispiele für Planungshilfen

Die Hilfeinformation wird dann zum einen dazu benutzt, um das betreffende Planelement zu aktualisieren und die entsprechenden Programmierhandlungen auszuführen, also Bildschirmelemente zu bilden ("Problemlösen mit Hilfen", Abb. 18 Mitte). Da der gesamte Problemlöseprozeß an dieser Stelle rekursiv aufgerufen wird, sind weitere Stocksituationen und Selbsthilfeprozesse möglich. Zum zweiten wird die Hilfeinformation in das entsprechende Konzept eingesetzt, wenn das betreffende Planelement in einem Entwurf erfolgreich verwendet wurde. Es wird also neues Konzeptwissen erworben.

Die *zweite* Form der Stocksituation liegt vor, wenn ein in der Aufgabenrepräsentation angesprochenes Konzept nicht direkt (d.h. ohne weiteres Planen) in ABSYNT-Programmcode umgesetzt werden kann ("zu_planen"), wie z.B. "so_berechnen_daß" in der "quot"-Aufgabe (s.o. Abb. 19). (Das zu erstellende Programm soll den Quotienten zweier positiver Zahlen soll so berechnen, daß er größer oder gleich 1 ist.) In diesem Fall gibt es in dem Modell zwei Möglichkeiten:
Die erste Möglichkeit besteht darin, das betreffende Konzept zu ignorieren und einen Lösungplan nur für die anderen, zu implementierenden Konzepte zu erstellen.
Die zweite Möglichkeit besteht in der Erstellung eines Lösungsplans unter Nutzung von *Vorwissen*. Zuerst sucht das Modell die Argumente des betreffenden Konzepts in der Aufgabenrepräsentation auf. Das erste Argument von "so_berechnen_daß" ist "teilen_durch...". Das zweite Argument ist "größer_gleich_pos". Dann wird dem Konzeptframe für "größer_gleich_pos" das folgende allgemeine Vorwissen für positive Zahlen entnommen:

"Wenn der Quotient zweier positiver Zahlen größer oder gleich eins ist,
dann ist die erste Zahl (Zähler) größer oder gleich der zweiten,
sonst ist die zweite Zahl (Nenner) größer als die erste."
wenn_dann_sonst (größer_gleich_pos (teilen_durch (X, Y), eins),
größer_gleich_pos (X, Y), größer_gleich_pos (Y, X)).

Dieses Vorwissen wird für die vorliegende Aufgabe genutzt, indem im zweiten und dritten Zweig der Fallunterscheidung "größer_gleich_pos ..." durch "teilen_durch …" ersetzt wird. Außerdem werden X und Y durch die entsprechenden Konzeptnamen ersetzt:

wenn_dann_sonst (größer_gleich_pos (teilen_durch (Zahl_1, Zahl_2), eins),
teilen_durch (Zahl_1, Zahl_2), teilen_durch (Zahl_2, Zahl_1)).

Mit diesem Ausdruck wird ein Lösungsplan gebildet, der nur Planelemente vom Typ "zu_implementieren" enthält. Abb. 24 enthält die vom Modell erzeugten Planelemente an diesem Punkt. Dieser Plan kann zu einem vollständigen Lösungsentwurf ausgeführt werden. Aufgrund des eingesetzten Vorwissens enthält dieser Entwurf Merkmale (wie das Vertauschen der Parameter und die Fallunterscheidung), die in der ursprünglichen Aufgabenrepräsentation nicht enthalten waren.

Planelement Nr: 2
Konzept: **teilen_durch**
Argumente: Planelement 4, Planelement 5
ist_Argument_von: {Planelement 7,
Planelement 8}
Implementation: / (P1, P2)
wird_implementiert_durch (Planelement 4, P1)
wird_implementiert_durch (Planelement 5, P2)
Typ: zu_implementieren

Planelement: Nr: 4
Konzept: **Zahl_1**
Argumente: keine
ist_Argument_von: {Planelement 2,
Planelement 9}
Implementation: Parameter a
Typ: zu_implementieren

Planelement: Nr: 5
Konzept: **Zahl_2**
Argumente: keine
ist_Argument_von: {Planelement 2,
Planelement 9}
Implementation: Parameter b
Typ: zu_implementieren

Planelement: Nr: 6
Konzept: **eins**
Argumente: keine
ist_Argument_von: {Planelement 8}
Implementation: Konstante 1
Typ: zu_implementieren

Planelement Nr: 7
Konzept: **wenn_dann_sonst**
Argumente: Planelement 8, Planelement 2,
Planelement 9
ist_Argument_von: {}
Implementation: if-then-else (P1, P2, P3)
wird_implementiert_durch
(Planelement 8, P1)
wird_implementiert_durch
(Planelement 2, P2)
wird_implementiert_durch
(Planelement 9, P3)
Typ: zu_implementieren

Planelement Nr: 8
Konzept: **größer_gleich_pos**
Argumente: Planelement 2, Planelement 6
ist_Argument_von: {Planelement 7}
Implementation: >= (P1, P2)
wird_implementiert_durch
(Planelement 2, P1)
wird_implementiert_durch
(Planelement 6, P2)
Typ: zu_implementieren

Planelement Nr: 9
Konzept: **teilen_durch**
Argumente: Planelement 5, Planelement 4
ist_Argument_von: {Planelement 7}
Implementation: / (P1, P2)
wird_implementiert_durch (Planelement 5, P1)
wird_implementiert_durch (Planelement 4, P2)
Typ: zu_implementieren

Abb. 24: Mit Hilfe von Vorwissen gebildeter Lösungsplan für die "quot"-Aufgabe

• *Wissensoptimierung* nach erfolgreicher Lösung: Aus den Konzepten, die für eine erfolgreiche Lösung benutzt wurden, werden Komposita gebildet. Abb. 25 illustriert, wie die Bildung von Komposita im externen Modell konzipiert ist. Bei der Bearbeitung der o.g. Aufgabe "quot" verwendet das Modell u.a. die Konzepte "größer_gleich_pos", "teilen_durch" und "eins" für den Lösungsentwurf am IF-Zweig der Fallunterscheidung. Die in den Planelementen verwendeten Realisationen dieser drei Konzepte werden durch den Kompositionsalgorithmus miteinander verbunden. Da der so entstandene Teilbaum das Ziel "größer_gleich_pos (S1, S2)" löst, führt das Kompositum zu einer Erweiterung des Konzeptes "größer_gleich_pos" um eine weitere Realisation.

Verwendete Realisationen der eingehenden Konzepte:

	größer_gleich_pos:	teilen_durch:	eins:
Argumente:	**S1, S2**	**S1, S2**	---
Implementation:	>= (P1, P2)	/(P1, P2)	Konstante 1
wird_implementiert_durch:	**S1**, P1	**S1**, P1	---
wird_implementiert_durch:	**S2**, P2	**S2**, P2	---

Das Kompositum aus diesen Realisationen:

(größer_gleich_pos $_1$• teilen_durch) $_3$• eins

ist eine weitere Realisation für "größer_gleich_pos (S1, S2):

	größer_gleich_pos:
Argumente:	**S1, S2**
Implementation:	>= (/(P1, P2), 1)
wird_implementiert_durch:	**S1**, P1
wird_implementiert_durch:	**S2**, P2

Abb. 25: Beispiel für ein Kompositum

Abb. 26 bis 29 zeigen Vergleiche des bisherigen Standes des Modells mit einigen zusammengefaßten Protokolldaten. Abb. 26 und 27 enthalten die Bearbeitung der Aufgabe "diffmaxmin" durch jeweils eine Person (Protokollepisoden, linke Spalte) und das Modell (rechte Spalte). Die Person stockt einmal und endet mit einer fehlerhaften Lösung. In Abb. 26 beginnt das Modell ohne Wissen über die ABSYNT-Implementation von "Maximum" und "Minimum". Deshalb tritt bei der Lösungsplanung eine Stocksituation ein. Die Hilfen werden aufgesucht. Die benötigte Implementationsinformation wird gefunden und in den Lösungsplan sowie in das Konzeptwissen integriert, und es wird ein Lösungsentwurf programmiert. (Die jeweils hinzuprogrammierten Teile sind fett gedruckt.)

Außerdem ist im in Abb. 26 dargestellten Ablauf in der Implementation des Konzepts "abziehen_von" nicht berücksichtigt, daß für die Benutzung des Subtraktionsoperators die Argumente vertauscht werden müssen. Deshalb wird in der Lösung das Maximum vom Minimum abgezogen, statt umgekehrt. Die Lösung von Person 8 sowie von dem Modell ist also fehlerhaft. Wird die Aufgabe dagegen mit diesen beiden

Wissensbestandteilen bearbeitet, so resultiert der in Abb. 27 dargestellte, nicht stockende sowie korrekte Verlauf, der mit dem Verhalten von Person 2 übereinstimmt.

Aufgabe "diffmaxmin": Programm, das das Minimum zweier Zahlen vom Maximum dieser Zahlen abzieht"

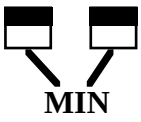
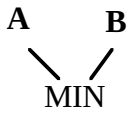
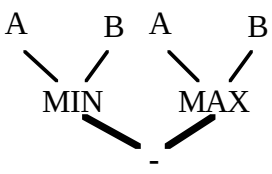
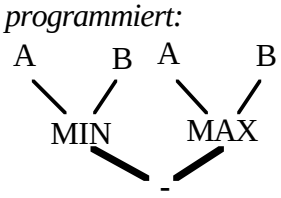
Person 8	Modell: a. *Ohne* Implementationswissen der Konzepte "Minimum" und "Maximum" b. *Kein* Vertauschen der Argumente bei Realisation von "abziehen_von" durch Subtraktionsoperator
Stocksituation: verbalisiert: "Minimum und Maximum, was ist denn das?"	*Stocksituation* bei Zugriff auf die Konzepte Minimum und Maximum während Lösungsplanung: Kein Realisationseintrag
blättert in Hilfen (Erläuterungen zu den primitiven Operatoren)	*Zugriff* auf die Planungshilfen für Minimum und Maximum
verbalisiert: "Da kann man doch jetzt bestimmt mit diesem Minimum- und Maximum-Knoten arbeiten"	*Planelemente* für Minimum und Maximum werden mit der Implementationsinformation aus den Hilfen gefüllt.
Fehler: programmiert zwei Konstantenknoten 	
Korrektur mit zwei Parameterknoten 	*programmiert:* 
programmiert: 	*programmiert:* 
(fehlerhafte Lösung) *programmiert:* 	*programmiert:* 

Abb. 26: Stockendes und fehlerhaftes Verhalten von Person 8 und dem Modell bei der Aufgabe "diffmaxmin"

Aufgabe "diffmaxmin": Programm, das das Minimum zweier Zahlen vom Maximum dieser Zahlen abzieht"	
Person 2	Modell: a. *Mit* Implementationswissen der Konzepte "Minimum" und "Maximum" b. *Mit* Vertauschen der Argumente bei Realisation von "abziehen_von" durch Subtraktionsoperator
programmiert: **A B** **MAX** *programmiert:* A B **A B** MAX **MIN** *programmiert:* A B A B MAX MIN -	*programmiert:* **A B** **MAX** *programmiert:* A B **A B** MAX **MIN** *programmiert:* A B A B MAX MIN -

Abb. 27: Nicht stockendes und fehlerfreies Verhalten von Person 2 und dem Modell bei der Aufgabe "diffmaxmin"

Bei der Aufgabe "diffmaxmin" werden die Handlungssequenzen der Versuchsteilnehmer von dem Modell zumindest auf dieser groben Analyseebene recht gut reproduziert. Lediglich die Programmierung von Konstanten und ihre anschließende spontane Korrektur (Person 8, Abb. 26) enthält das Modell nicht.

Abb. 28 vergleicht den Lauf des Modells mit dem Verhalten von Person 8 und Person 2 bei der Aufgabe "quot". Bei dem Modellauf in Abb. 28 oben ignoriert das Modell das weitere Planung erfordernde Konzept "so_berechnen_daß". Bei dem Lauf in Abb. 28 unten dagegen wird Vorwissen zur Bildung eines Lösungsplans genutzt, wie beschrieben. Bei den Aufgaben "quot" und "gerade" treten jedoch insbesondere hinsichtlich der Reihenfolge der Programmierhandlungen Abweichungen auf (Abb. 28 und 29). Hier sind weitere Analysen von Handlungsprotokollen erforderlich.

Aufgabe "quot": Programm, das den Quotienten zweier Zahlen so berechnet, daß dieser größer oder gleich 1 ist."

Person 8	Modell: Mit Ignorieren des Konzepts "so_berechnen_daß"
programmiert: **X** **Y** / *programmiert:* X Y / **1** >=	*programmiert:* **X** **Y** / *programmiert:* X Y / **1** >=
Person 2	**Modell: Mit Nutzung von Vorwissen**
programmiert: **B** **A** <= *programmiert:* B A <= **if-then-else** *programmiert:* B A **A** **B** <= / if-then-else *programmiert:* B A A B **B** **A** <= / / if-then-else	*programmiert:* **A** **B** / *programmiert:* A B **B** **A** / / *programmiert:* **A** **B** A B B A / / / **1** < *programmiert:* A B A B B A / / / 1 < **if-then-else**

Abb. 28: Verhalten von Person 8, Person 2 und dem Modell bei der Aufgabe "quot"

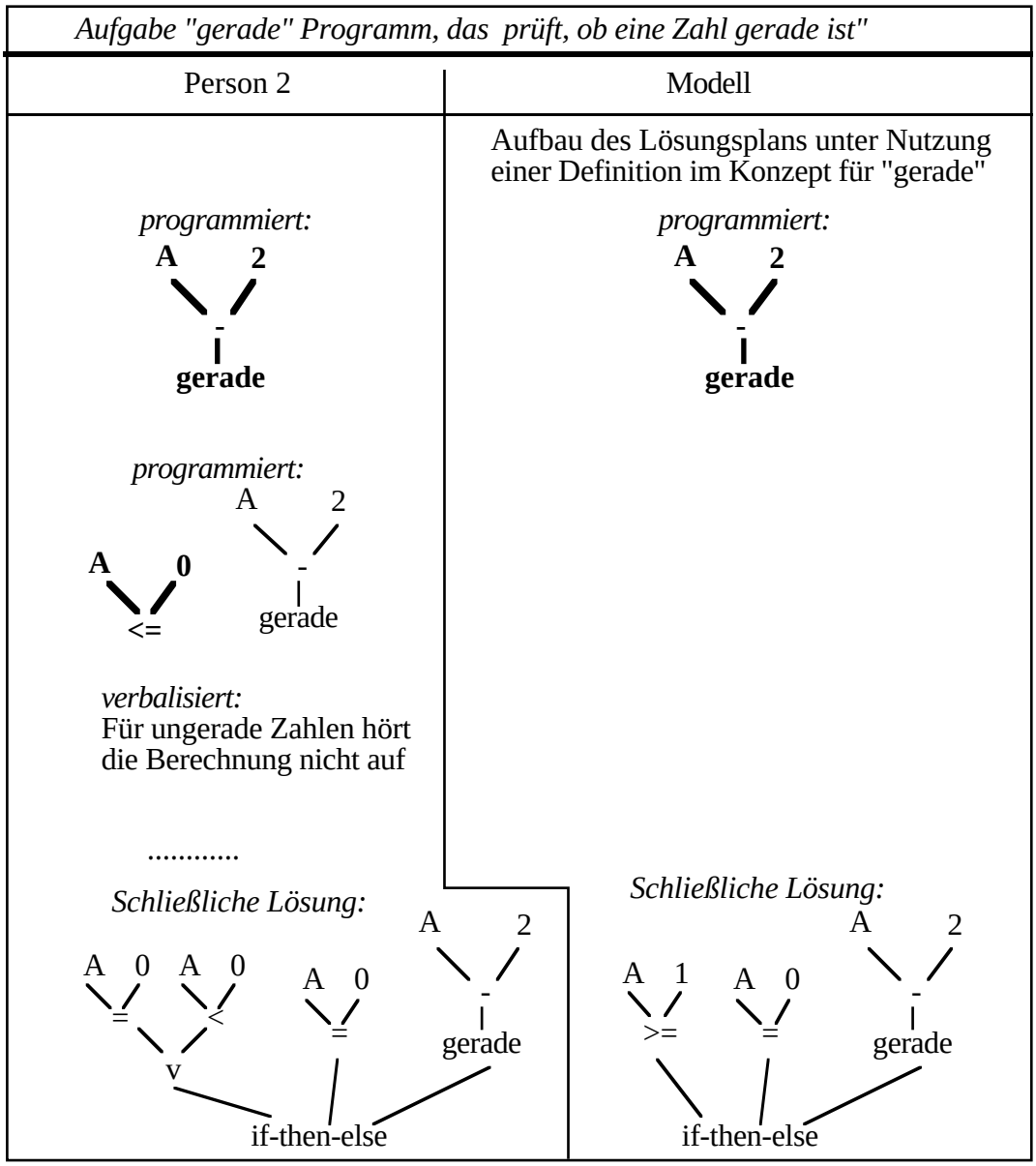


Abb. 29: Verhalten von Person 2 und dem Modell bei der Aufgabe "gerade"

Ausblick

Gegenstand der laufenden und zukünftigen Arbeiten ist die Weiterentwicklung der ISPDL-Theorie, des internen (Stadien-) Modells und des externen (Prozeß-) Modells. Dabei sollen alle drei Arbeitsschwerpunkte noch enger aufeinander bezogen sein, um eine möglichst hohe Konsistenz zwischen den Ebenen der Theorie, der Daten und der beiden Modelle (siehe auch Abb. 1) zu gewährleisten. Die folgenden Arbeitsschritte sind geplant:

• Hinsichtlich der *ISPDL-Theorie*: die weitere Spezifikation der verschiedenen Wissensstrukturen und Prozesse und ihrer empirischen Indikatoren. So müssen in weiteren empirischen Analysen das aktuelle hypothetische Domänenwissen, Gedächtnisspuren, Zielsetzungen und die hypothetischen Heuristiken und relevanten Vorwissensbestände spezifiziert werden. Weiterhin sind Hypothesen erforderlich über das Abwägen zwischen Zielen und zwischen verschiedenen Heuristiken (wie Informationsbeschaffung über Hilfen vs. Selbsterklärung von Beispielen), über das Planen, Ausführen und Bewerten von Lösungsentwürfen und über mögliche

Rücksprünge zwischen diesen Phasen und ihre Bedingungen, sowie über Bedingungen des Wissensoptimierung und über den Einsatz von Kontrollwissen.

• Hinsichtlich des *internen Modells*: Die Weiterentwicklung im Sinne einer möglichst validen und effizienten Wissensdiagnose und Hilfengenerierung. Dazu gehört die Überprüfung der Vorhersagen des internen Modells hinsichtlich Stocksituationen, Verbalisationen und Sequenzen vs. Mengen von Handlungsschritten.
Liegt ein valides und effizientes Modell vor, so kann in einem weiteren Arbeitsschritt die differentielle Wirksamkeit unterschiedlich stark wissensstands- bzw. defizitbezogener Hilfeinformationen in interventionsorientierten Einzelfallexperimenten überprüft werden. Dazu kann die Beziehung zwischen dem jeweiligen Domänenwissen des Lernenden (wie es im internen Modell repräsentiert ist) und der jeweils aktuell angebotenen Hilfeinformation variiert werden. Die Hilfeinformation kann sowohl hinsichtlich ihrer Korngröße als auch hinsichtlich der angebotenen Informationsmenge variiert werden. Hinsichtlich der Korngröße kann die angebotene Hilfeinformation "zu grob" sein (stärker komponiert als das Wissen des Lernenden), "zu fein" (schwächer komponiert) oder aber "synchron" (mit gleichem Kompositionsgrad). Die Informationsmenge kann auf die aktuelle Stocksituation zugeschnitten sein ("synchron"), oder aber mehr oder weniger Information bereitstellen.

• Hinsichtlich des *externen Modells*: Die vollständige Implementation, die anschließende Durchführung weiterer Einzelversuche und die Überprüfung und Weiterentwicklung des Modells anhand detaillierter Protokollanalysen. Das Modell enthält bereits einige Stocksituationen und Heuristiken und kann damit einige Prozesse der Veränderung des Domänenwissens simulieren. Bisher fehlen jedoch detaillierte Hypothesen insbesondere darüber, wann und warum welche Komposita gebildet werden, wann der Lernende auf welche Heuristiken zugreift und aufgrund welcher Kontrollprozesse zwischen verschiedenen Realisationsmöglichkeiten bei der Lösungsplanung ausgewählt wird. Des weiteren hat sich auch gezeigt, daß Annahmen über das Vorwissen erforderlich sind, die gegenwärtig vorwiegend auf Plausibilitätserwägungen beruhen. Auch die Bildung von Prüfhypothesen und die Verfolgung von Hypothesenprüfstrategien ist in dem Modell noch nicht enthalten.

Literatur

Adelson, B., Problem Solving and the Development of Abstract Categories in Programming Languages, Memory and Cognition, 1981, 9, 422-433

Anderson, J.R., The Architecture of Cognition. Cambridge: Harvard University Press, 1983a

Anderson, J.R., Acquisition of Proof Skills in Geometry, in Michalski, R.S., Carbonell, J.G., Mitchell, T.M.(eds), Machine Learning. Palo Alto: Tioga Press, 1983b, S. 191-219

Anderson, J.R., Knowledge Compilation: The General Learning Mechanism, in Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds), Machine Learning, Vol. II. Los Altos: Kaufman, 1986, S.289-310

Anderson, J.R., A Theory of the Origins of Human Knowledge, Artificial Intelligence, 1989, 40, S. 313-351

Bögelsack, A., Weber, G., Automatische Diagnose von LISP-Programmen, in Gunzenhäuser, R., Mandl, H. (Hrsg), 3. Workshop der Fachgruppe "Intelligente Lernsysteme", Tübingen, Juni 1989, S. 15-24

Brown, J.S., van Lehn, K., Repair Theory: A Generative Theory of Bugs in Procedural Skills, Cognitive Science, 4, 1980, S. 379-426

Chase, N.G., Simon, H.A., Perception in Chess, Cognitive Psychology, 1973, 4, S. 55-81

Cheng, P.W., Carbonell, J.G., The FERMI System: Inducing Iterative Macro-operators from Experience. a a a i 86 Proceedings, Los Altos: Morgan Kaufman, 1986, S. 490-495

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R., Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems, Cognitive Science, 1989, 13, S. 145-182

Elio, R., Representation of Similar Well-Learned Cognitive Procedures, Cognitive Science, 1986, 10, S.41-73

Ernst, G.W., Newell, A., GPS: A Case Study in Generality and Problem Solving, New York: Academic Press, 1969

Glinert, E.P., Nontextual Programming Environments, in Chang (ed), Principles of Visual Programming Systems, Englewood Cliffs: Prentice Hall, 1990, S. 144-230

Gollwitzer, P.M., Action Phases and Mind-Sets, in: E.T. Higgins & R.M. Sorrentino (eds), Handbook of Motivation and Cognition: Foundations of Social Behavior, 1990, Vol.2, S. 53-92

Gugerty, L., Olson, G.M., Comprehension Differences in Debugging by Skilled and Novice Programmers, in: E.Soloway & S. Iyengar (eds), Empirical Studies of Programmers, Norwood, N.J.: Ablex, 1986, S. 13-27

Haack, U., Hahn K., Wagner, K.-U., Programmieren als Problemlösen: Die Struktur von Expertenwissen, Universität Marburg: FB Psychologie, 1988

Haack, U., Hahn K., Wagner, K.-U., Programmieren als schematisches Problemlösen, Zeitschrift für Psychologie, 1989, 197, S. 247-262

Heckhausen, H., Wünschen-Wählen-Wollen, in: H. Heckhausen, P.M. Gollwitzer & F.E.Weinert (eds), Jenseits des Rubikon: Der Wille in den Humanwissenschaften, Heidelberg: Springer-Verlag, 1987

Heckhausen, H., Motivation und Handeln, Heidelberg: Springer, 1989 (2. Aufl.)

Hofbauer, D., Kutsche, R.-D., Grundlagen des maschinellen Beweisens, Braunschweig: Vieweg, 1989

Huber, P., Jensen, K., Shapiro, R.M., Hierarchies in Coloured Petri Nets, in G. Rozenberg (ed), Advances in Petri Nets, Lecture Notes in Computer Science, Heidelberg: Springer, 1990

Iba, G.A., A Heuristic Approach to the Discovery of Macro-operators, Machine Learning, 1989, 3, S. 285-317

Janke, G., Kohnert, K. Interface Design of a Visual Programming Language: Evaluating Runnable Specifications. in: F. Klix, N.A. Streitz, Y. Waern & N. Wandke (eds), MACINTER-II Man-Computer-Interaction Research, Proceedings of the Second Network Seminar of MACINTER held in Berlin/GDR, March 21 - 25, 1988, Amsterdam: North Holland, 1989, S. 567 - 581

Kintsch, W., Greeno, J. G., Understanding and Solving Word Arithmetic Problems. Psych. Review, 1985, 92(1), S. 109-129

Korf, R.E., Planning as Search: A Quantitative Approach, Artificial Intelligence, 1987, 33, S. 65-88

Kowalski, R., Logic for Problem Solving, Amsterdam: Elsevier Publ., 1979

Laird, J.E., Rosenbloom, P.S., Newell, A., Universal Subgoaling and Chunking. The Automatic Generation and Learning of Goal Hierarchies, Boston: Kluwer, 1986

Laird, J.E., Rosenbloom, P.S., Newell, A., SOAR: An Architecture for General Intelligence, Artificial Intelligence, 1987, 33, S. 1-64

Lewis, C., Composition of Productions, in Klahr, D., Langley, P., Neches, R. (eds), Production, System Models of Learning and Development, Cambridge: MIT Press, 1987, S. 329-358

Möbus, C. Toward the Design of Adaptive Instructions and Helps for Knowledge Communication with the Problem Solving Monitor ABSYNT. in: V. Marik, O. Stepankova & Z. Zdrahal (eds): Artificial Intelligence in Higher Education, Proceedings of the CEPES UNESCO International Symposium Prague, CSFR, October 23 - 25, 1989, Berlin - Heidelberg - New York: Springer, Lecture Notes in Computer Science, Nr.451 (subseries LNAI), 1990, S. 138 - 145

Möbus, C., The Relevance of Computational Models of Knowledge Acquisition for the Design of Helps in the Problem Solving Monitor ABSYNT, in R.Lewis & S.Otsuki (eds), Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education Tokyo, Japan, 18-20 July, 1990, Elsevier Science Publishers B.V. (North-Holland), 1991a, S. 137-144

Möbus, C., Wissenserwerb mit kooperativen Systemen, in P. Gorny (Hrsg), Informatik und Schule 1991 - Informatik: Wege zur Vielfalt beim Lehren und Lernen, GI-Fachtagung, Oldenburg, Oktober 1991, Informatik-Fachberichte 292, Berlin: Springer, 1991b, S. 288-298

Möbus, C., Schröder, O., Knowledge Specification and Instructions for a Visual Computer Language, in: F. Klix, N.A. Streitz, Y. Waern & N. Wandke (eds), MACINTER-II Man-Computer-Interaction Research, Proceedings of the Second Network Seminar of MACINTER held in Berlin/GDR, March 21 - 25, 1988, Amsterdam: North Holland, 1989, S. 535 - 565

Möbus, C., Schröder, O., Representing Semantic Knowledge with 2-dimensional Rules in the Domain of Functional Programming, in: P.Gorny & M. Tauber (eds), Visualization in Human-Computer Interaction, 7th Interdisciplinary Workshop in Informatics and Psychology, Schärding. Austria, May 1988; Lecture Notes in Computer Science, Vol. 439, Berlin: Springer, 1990, S. 47-81

Möbus, C., Schröder, O., Thole, H.J., Runtime Modeling the Novice-Expert Shift in Programming Skills on a Rule-Schema-Case Continuum, ABSYNT-Report 12/91, Universität Oldenburg, FB Informatik, Abt. Lehr- Lernsysteme, 1991a

Möbus, C., Schröder, O., Thole, H.J., Runtime Modeling the Novice-Expert Shift in Programming Skills on a Rule-Schema-Case Continuum, in: J. Kay; A. Quilici (eds), Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction, 12th Int. Joint Conf. on Artificial Intelligence, Darling Harbour, Sydney, Australia, 24-30 August 1991b, S. 137-143

Möbus, C., Thole, H.-J., Tutors, Instructions and Helps, in: Christaller, Th. (ed), Künstliche Intelligenz KIFS 1987, Informatik-Fachberichte 202, Heidelberg: Springer, 1989, S. 336 - 385

Möbus, C., Thole, H.-J., Interactive Support for Planning Visual Programs in the Problem Solving Monitor ABSYNT: Giving Feedback to User Hypotheses on the Basis of a Goals-Means-Relation, in: D.H. Norrie, H.-W. Six (eds), Computer Assisted Learning. Proceedings of the 3rd International Conference on Computer-Assisted Learning ICCAL 90, Hagen, F.R.Germany, Lecture Notes in Computer Science, Vol. 438, Heidelberg: Springer, 1990, S. 36-49

Müller, B., Weber, G., Individualisierung der Lernerunterstützung im ELM-LISP-Tutor, in P. Gorny (Hrsg), Informatik und Schule 1991 - Informatik: Wege zur Vielfalt beim Lehren und Lernen, GI-Fachtagung, Oldenburg, Oktober 1991, Informatik-Fachberichte 292, Berlin: Springer, 1991, S. 306-313

Neves, D.M., Anderson, J.R., Knowledge Compilation: Mechanisms for the Automatization of, Cognitive Skills, in Anderson, J.R. (ed), Cognitive Skills and their Acquisition, Hillsdale, Erlbaum, 1981, S. 57-84

Reisig, W., Petrinetze - eine Einführung, Heidelberg: Springer, 1982

Reisig, W., Petri Nets - An Introduction, Springer EATCS Monographs on Theoretical Computer Science, New York: Springer, 1985

Rosenbloom, P., Newell, A., The Chunking of Goal Hierarchies: A Generalized Model of Practice, in Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds), Machine Learning, Vol. II. Los Altos: Kaufman, 1986, S. 247-288

Rosenbloom, P., Newell, A., Learning by Chunking: A Production System Model of Practice, in Klahr, D., Langley, P., Neches, R. (eds), Production System Models of Learning and Development, Cambridge: MIT Press, 1987, S. 221-286

Schröder, O., Erwerb von Regelwissen mit visuellen Hilfen: Das Semantikwissen für eine graphische funktionale Programmiersprache, Dissertation, Universität Oldenburg, 1990a, Frankfurt: Lang (i.Dr.)

Schröder, O., A Model of the Acquisition of Rule Knowledge with Visual Helps: The Operational Knowledge for a Functional, Visual Programming Language, in: D.H. Norrie, H.-W. Six (eds), Computer Assisted Learning. Proceedings of the 3rd International Conference on Computer-Assisted Learning ICCAL 90, Hagen, F.R.Germany, Lecture Notes in Computer Science, Vol. 438, Heidelberg: Springer, 1990b, S. 142-157

Self, J., Bypassing the Intractable Problem of Student Modeling, in: C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, S. 107-123

Simon, H.A., Simon, D.P., Individual Differences in Solving Physics Problems, in R.S. Siegler (ed), Childrens´Thinking: What Develops? Hillsdale: Erlbaum, 1978, S. 325-348

Slade, S., Case-Based Reasoning: A Research Paradigm, AI Magazine, 1991, 12(1), S. 42-55

Van Dijk, T.A.; Kintsch, W., Strategies of Discourse Comprehension. New York: Academic Press, 1983

Van Lehn, K., Toward a Theory of Impasse-Driven Learning, in Mandl, H.; Lesgold, A. (eds), Learning Issues for Intelligent Tutoring Systems, New York: Springer, 1988, S. 19-41

Van Lehn, K., Mind Bugs: The Origins of Procedural Misconceptions, Cambridge: MIT Press, 1990

Van Lehn, K., Rule Acquisition Events in the Discovery of Problem-Solving Strategies, Cognitive Science, 1991, 15, S. 1-47

Vere, S.A., Relational Production Systems, Artificial Intelligence, 1977, 8, S. 47-68

Vorberg, D., Programmierkonzepte als Werkzeuge zum Problemlösen, in: Bericht über das DFG-Schwerpunktprogramm "Wissenspsychologie" Herbst 1985 bis Frühjahr 1989, S. 243-253

Vorberg, D., Göbel, R., Rekursionsschemata als Problemlösepläne: Das Lösen rekursiver Programmierprobleme mit LOGO. Universität Marburg, FB Psychologie, 1989

Weber, G., Automatische kognitive Diagnose in einem Programmier-Tutor, in D. Metzing (Hrsg.), GWAI 89 - 13th German Workshop on Artificial Intelligence, Informatik-Fachberichte 216, Berlin: Springer, 1989, S. 331-336

Weber, G., Bögelsack, A., Entwicklung der Diagnosekomponente und episodischen Schüler-Modellierung in einem intelligenten LISP-Tutor, in Gunzenhäuser, R., Mandl, H. (Hrsg), 2. Workshop der Fachgruppe "Intelligente Lernsysteme", Tübingen, Juni 1988, S. 129-137

Winkels, R., Breuker, J., Discourse Planning in Intelligent Help Systems, in: C. Frasson, G. Gauthier (eds), Intelligent Tutoring Systems, Norwood: Ablex, 1990, S. 124-139

Wolff, J.G., Cognitive Development as Optimisation, in L. Bolc (ed), Computational Models of Learning, Berlin: Springer, 1987, S. 161-205

Young, R., Sowing Seeds Versus Planting Trees: Implications of a Problem-Space Model of Cognition for Intelligent Tutoring, paper presented at the NATO Advanced Research Workshop on Student Modelling, St.Adele, Quebec, Canada, May 5-8, 1991

Diese Arbeit wurde durch die Deutsche Forschungsgemeinschaft im SPP Wissenspsychologie unterstützt (Mo 292/3-3).
Wir möchten uns bei den folgenden Mitarbeitern bedanken:
bei stud. inf. Jörg Folckers für die Implementation von ABSYNT in MacProlog,
bei cand. inf. Hermann Göhler für die Implementation des Externen Modells in FLEX-Prolog,
bei Dipl. Inf. Knut Pitschke für die Diskussionen bezüglich der Taxonomie von Lernverfahren,
bei cand. psych. Erika Roxin für die sorgfältige Auswertung der Videoprotokolle,
bei cand. math. Heinz-Jürgen Thole für die Implementation des Internen Modells in MacProlog.