Das Erlernen einer Programmiersprache: Wissenserwerb aus Texten, Beispielen und komplexen Programmen

Ralph Bergmann, Stefan Boschert
und Franz Schmalhofer

1 Einleitung

Bekanntlich können Menschen aus sehr unterschiedlichen Erfahrungen und Lernmaterialien gerade diejenigen Konzepte erwerben, die sich später für ihr weiteres Handeln als nützlich erweisen. Dies gilt auch für das Erlernen einer Programmiersprache. So können relevante Kenntnisse über eine Programmiersprache aus Lehrtexten, aus konkreten Beispielen der verfügbaren Programmkonstrukte und aus vollständigen Programmen mit komplexen Kontrollstrukturen erworben werden. Weitergehende Kenntnisse werden oft durch die Interaktion mit einem Programmiertutor (Weber, 1992), durch das Explorieren einer Programmierumgebung oder beim Erstellen eines speziellen Programms erworben.

Für das erfolgreiche Studium eines Lernmaterials werden meist mehr oder weniger umfangreiche Kenntnisse vorausgesetzt. Während sich in manchen Fällen so eine natürliche Sequenz der Lernmaterialien ergibt, können manchmal auch verschiedene Lernmaterialien zum Erwerb des gleichen Wissens eingesetzt werden. Die vorliegende Arbeit beschreibt kognitive Modellierungen des Wissenserwerbs aus drei verschiedenen Materialien: Texten, Beispielen von Programmkonstrukten und komplexen Programmen. Während die Texte und Beipiele alternativ oder zur gegenseitigen Ergänzung beim Erwerb des gleichen Wissens eingesetzt werden können, setzt das Studium von komplexen Programmen bereits solche Kenntnisse über Programmkonstrukte voraus.

Wir beschäftigen uns in dieser Arbeit mit dem Erlernen einer Programmiersprache von Anfang an: Zuerst wird Erwerb von Programmierkonstrukten aus Text und Beispielen modelliert, was als die "erste Stunde" des Erlernens einer bezeichnet werden kann. Das darauf aufbauende Erlernen von zielorientiertem Problemlösewissen aus vollständigen Programmen bezeichnen wir als die "zweite Stunde".

2 Die "erste Stunde": Das Erlernen von Programmkonstrukten aus Text und Beispielen

Die relevanten Konzepte einer Programmiersprache kann ein Anfänger durch das Studieren eines einführenden Textes oder durch das Studium von mehreren Programmbeispielen erwerben. Sowohl das Lernen aus Text als auch das Lernen aus Beispielen bringt spezielle Vor- und Nachteile mit sich. Deshalb ist der Wissenserwerb aus einer geeigneten Kombination der beiden Materialien nahezu immer günstiger als das Lernen aus nur einem Material.

In den bisherigen Forschungen zu diesem Thema wurde meist versucht, die Effektivität des Lernens aus verschiedenen Lernmaterialien nahezu ausschließlich an Hand der Ergebnisse empirischer Untersuchungen zu beurteilen. So wurde in der pädagogischen Psychologie mit einem aufwendigen Forschungsprogramm die Frage erkundet, ob rezeptives Lernen (Ausubel, 1964) oder Entdeckungslernen (Bruner, 1961) zu einem günstigeren Lernerfolg führen würden. In einer Vielzahl von Untersuchungen (etwa Guthrie, 1967), die von Neber (1981) zusammenfassend beschrieben wurden, führte einmal das eine und dann wieder das andere Lernmaterial zu einem besseren Lernerfolg.

Aufgrund der verschiedenen in den Untersuchungen verwendeten Operationalisierungen und wegen der zunächst widersprüchlich erscheinenden Ergebnisse mußten zunehmend stärkere Differenzierungen verwendet werden, um die empirischen Befunde überblicksmäßig beschreiben zu können. Die starke Differenzierung, von denen anfängliches versus spätes Behalten und anfänglicher Transfer versus später Transfer noch als vergleichsweise nützliche Unterscheidungen anzusehen sind, führte zu großer Unübersichtlichkeit. So kam Lefrancois (1976) auch zu der Einschätzung, daß sich diese Untersuchungen insgesamt als erfolglos erwiesen haben.

Heute werden im Bereich der Mensch-Maschine-Kommunikation ähnliche Fragen mit der gleichen Forschungsmethodik wieder untersucht. Lehren, die aus dem mangelnden Ertrag der Untersuchungen der pädagogischen Psychologie gezogen werden könnten, werden dabei nicht berücksichtigt, d.h. die gleichen Fehler scheinen sich in einem anderen Gegenstandsgebiet (Lernen der Computernutzung durch Neulinge) zu wiederholen. So wird beispielsweise in Experimenten mit sehr geringem oder keinerlei theoretischem Überbau erforscht, ob sich die Lernschwierigkeiten von Computerneulingen durch vollständige Handlungsanweisungen beheben lassen. Andererseits wird für Computerneulinge ein aktives Entdeckungslernen durch "learning by doing" empfohlen (Carroll, 1985).

Im Gegensatz zu der pädagogischen Psychologie und empirischen Untersuchungen auf dem Gebiet der Mensch-Maschine Kommunikation wurden in der experimentellen allgemeinen Psychologie Textverstehen und der Konzepterwerb aus Beispielen nicht vergleichend untersucht. Textverstehen (z. B. Kintsch, 1974) wurde im Rahmen eines Gedächtnisparadigmas untersucht. Dabei wurden Gedächtnisrepräsentationen, Enkodierungsprozesse und Repräsentationsfragen im allgemeinen thematisiert. Der Konzepterwerb beim Studieren von Beispielen wurde innerhalb des Problemlöseparadigmas untersucht, wobei Verarbeitungsstrategien, Hypothesenbildung und Konzeptidentifikation thematisiert wurden.

Dadurch, daß Textverstehen und der Konzepterwerb aus Beispielen voneinander getrennt, dafür aber gründlicher untersucht wurden, haben wir heute einerseits ein gutes Verständnis der kognitiven Prozesse beim Textverstehen und andererseits auch ein gleichermaßen gutes Verständnis der kognitiven Prozesse des Konzepterwerbs aus Beispielen. Dagegen liegt bisher keine integrative Theorie darüber vor, wie Konzepte aus Text und aus Beispielen erworben werden.

Wie die bereits erläuterten Themenstellungen aus der pädagogischen Psychologie und der Mensch-Maschine-Kommunikation zeigen, ist eine vergleichende Betrachtung des Konzepterwerbs aus Text und Beispielen oder einer Kombination dieser Materialien in vielen Anwendungssituationen von Bedeutung. Schließlich stellen Lernen aus Text und Lernen aus Beispielen zwei der wichtigsten Formen des menschlichen Konzepterwerbs dar. Da empirische und experimentelle Untersuchungen ohne eine adäquate theoretische Fundierung zu unfruchtbaren Differenzierungen führen, durch die das Kernthema leicht aus dem Auge verloren wird, sollte hier ein neuer Forschungsweg beschritten werden. Im folgenden soll ein kognitives Modell des Wissenserwerbs aus Texten und Beispielen für die Domäne des Erlernens elementarer Funktionen einer Programmiersprache beschrieben werden.

2.1 Modellierung des Erlernens elementarer LISP-Funktionen

Jede elementare LISP-Funktion, wie FIRST, REST, EQUAL und LIST, läßt sich durch vier Attribute definieren. Durch die Angabe von Spezifikationen für die Attribute "NAME", "ANZAHL_DER_ARGUMENTE", "ARGUMENT_TYP", und "I/O_BEZIEHUNG" kann bei entsprechenden Vorkenntnissen jede dieser Funktionen als vollständig definiert betrachtet werden. Aus der Kombination der Attribute ergibt sich das folgende Funktionsschema mit vier Slots:

Funktionsschema:

```
NAME:
ANZAHL_DER_ARGUMENTE:
ARGUMENT_TYP:
I/O_BEZIEHUNG:
```

Durch das Füllen der Slots ergibt sich ein instanziiertes Funktionsschema, wodurch eine LISP-Funktion definiert wird. Die Funktion FIRST ist somit bestimmt durch:

```
NAME: first
ANZAHL_DER_ARGUMENTE: eins
ARGUMENT_TYP: liste
I/O_BEZIEHUNG: erstes-element-der-liste
```

Personen, die bereits Kenntnisse der LISP-Syntax und über die elementaren Datenstrukturen in LISP (Atome und Listen) aufweisen und das oben spezifizierte Funktionsschema besitzen, sollten nun eine LISP-Funktion wie FIRST sowohl aus einem Text als auch aus einer geeigneten Menge von Beispielen erlernen können. Tabelle 1 zeigt einen Ausschnitt des Texts und eine Sequenz von Funktionseingaben in den LISP-Interpreter mit dem dazu korrespondieren Output.

```
Text
Die Funktion FIRST wird verwendet, um den ersten S-Term aus
einem zusammengesetzten S-Term zu extrahieren.
Die Funktion FIRST hat genau ein Argument.
Das Argument der Funktion FIRST muß ein zusammengesetzter
S-Term sein.
Der Wert der Funktion FIRST ist der erste S-Term des Arguments.

Beispiele
Anhand der folgenden Beispiele können Sie Kenntnisse der Funk-
tion FIRST erwerben. Beachten Sie die Anführungszeichen!
Mögliche Eingaben und die dazugehörigen Ausgaben des LISP-
Systems werden jeweils links und rechts in einer Zeile angege-
ben.
```

(FIRST '(A B))	→	A
(FIRST '((A B) C))	→	(A B)
(FIRST '(A (B C)))	→	A
(FIRST '((A B) (C D)))	→	(A B)
(FIRST '(A))	→	A
(FIRST (FIRST '((A B) C)))	→	A
(FIRST '(FIRST ((A B) C)))	→	FIRST
(FIRST 'A 'B)	→	ERROR
FIRST '(A B)	→	ERROR
(FIRST (A B))	→	ERROR
(FIRST 'A)	→	ERROR
(FIRST (A 'B))	→	ERROR

Tabelle 1: Ausschnitte aus den Lernmaterialien

Im Text sind in natürlichsprachlicher Form die Werte für die Attribute des Funktionsschemas explizit angegeben. Deshalb sollte ein Lernender aus diesem Text das relevante Wissen über die Funktion FIRST erwerben können.

Bei gleichem Vorwissen kann die Funktion FIRST aber auch aus den in Tabelle 1 angegebenen Beispielen erlernt werden. Wenn der Lernende das erste Beispiel mit seinem Vorwissen analysiert, kann er feststellen, daß es sich dabei um eine syntaktisch korrekte Eingabe in das LISP-System handelt. Desweiteren läßt sich erkennen, daß FIRST eine Funktion ist und (A B) das Argument. Der Output des Funktionsaufrufs ist A. Daraus können nun Hypothesen über Typ und Anzahl der Argumente und über die I/O-Beziehung gebildet werden, die wiederum richtig oder falsch sein können. Nachdem eine genügende Anzahl der angegebenen Beispiele für die Hypothesenprüfung und Hypothesenmodifikation genutzt worden ist, haben sich die Hypothesen bestätigt, die im Text als Aussagen explizit mitgeteilt wurden. Somit sollte aus einem Text und aus Beispielen, die bei einem bestimmten Vorkenntnisstand als "informationsäquivalent" (Larkin & Simon, 1987; Schmalhofer, Boschert & Kühn, 1990) angesehen werden können, das gleiche Konzept erworben werden.

2.2 Ein kognitives Modell des Konzepterwerbs

Eine erste graphische Veranschaulichung des Modells ist in Abbildung 1 gegeben. Das Vorwissen des Lernenden, das einerseits aus dem Alltagswissen, das mehrere Heuristiken einschließt, und andererseits aus bereichsspezifischen Vorkenntnissen wie dem Funktionsschema besteht, ist in dieser Abbildung durch zwei Kästen dargestellt. Das beim Lernen neu aufgebaute Wissen ist durch drei Ellipsen dargestellt: Beim Lernen aus Text wird eine Bedeutungsrepräsentation des Textes (Textbasis) erstellt. Ein Situationsmodell, das eine Repräsentation der Gegenstandsdomäne (zum Beispiel das Wissen über den LISP-Interpreter) darstellt, kann ebenfalls beim Lernen aus Text erworben oder erneuert ("updating") werden. Wie das obengenannte Beispiel des Erlernens der LISP-Funktion FIRST gezeigt hat, sollte das gleiche Situationsmodell auch aus einer Menge informationsäquivalenter Beispiele erworben werden. Beim Lernen aus Beispielen werden Schablonen (Template-Base, vgl. Anderson, Farrell & Sauers, 1984) als periphere Wissensrepräsentationen aufgebaut.

Das Modell basiert auf den folgenden Grundannahmen (vgl. Schmalhofer, Boschert & Kühn, 1990; S. 178-179):

a) *Multiple Wissensrepräsentationen:* Es werden drei Wissensrepräsentationen postuliert, die im folgenden als 1) Situationsmodell, 2) Schablonenbasis und 3) als Textbasis bezeichnet werden. Das Situationsmodell (van Dijk & Kintsch, 1983) repräsentiert die strukturelle Basis des Gegenstandsbereiches. Dagegen beinhaltet die Schablonenbasis Eigenschaften von Situationsbeispielen, die durch Schablonen (Anderson, Farrell & Sauers, 1984) dargestellt werden. Die Textbasis (Kintsch, 1974) ist aus Propositionen aufgebaut, die in Texten zum Ausdruck kommen können.

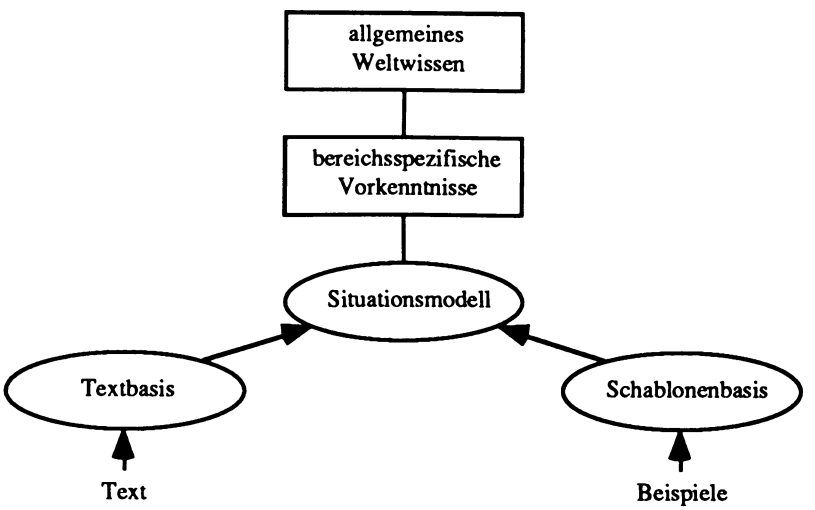

Abbildung 1: Integriertes Modell des menschlichen Lernens

b) *Text- und situationsbasierter Wissenserwerb:* Der Erwerb von allgemeinem Situationswissen kann aus natürlichsprachlichen Beschreibungen und/oder aus beispielhaften Einzelsituationen, also text- und/ oder situationsbasiert erfolgen. In Abhängigkeit vom Lernmaterial entstehen dabei zuerst text- und situations-

basierte Wissensrepräsentationen, d.h. eine propositionale Textbasis oder Schablonen. Beim Lernen aus Text und Lernen aus Beispielen werden also verschiedene periphere Wissensrepräsentationen (Textbasis oder Schablonenbasis) aufgebaut.

c) *Gleiches allgemeines Situationswissen bei informationsäquivalenten Lernmaterial:* Da sowohl die Textbasis als auch die Schablonenbasis weiter verarbeitet werden, kann aus text- und situationsbasiertem Lernmaterial das gleiche allgemeine Situationswissen erworben werden. Dazu müssen die beiden Lernmaterialien jedoch informationsäquivalent sein.

d) *Abhängigkeit von Vorwissen und Zielsetzung:* Zur Wissensenkodierung wird stets das für den jeweiligen Bereich spezifische Vorwissen herangezogen. Beim Wissenserwerb können in Abhängigkeit von der jeweiligen Zielsetzung durch Umsetzungsprozesse zwischen Schablonenbasis und Textbasis implizite Informationen des Materials in explizites Wissen umgewandelt werden.

Lernen aus Text: Auf der situativen Ebene laufen beim Lernen aus Text Inferenzprozesse ab. So werden aus neu hinzukommenden Informationen und dem aktivierten alten Wissen Inferenzen gezogen (Schmalhofer, Kühn & Messamer, 1989). Dadurch werden weitere Bezüge zwischen den neuen Informationen und dem alten Wissen erzeugt und es findet eine teilweise Operationalisierung allgemeineren Wissens statt.

Lernen aus Beispielen: Beim Lernen aus Beispielen werden ebenfalls die Wissenseinheiten aktiviert, die zu dem Beispiel in Beziehung stehen. Mit dem so aktivierten Wissen wird dann das vorliegende Beispiel erklärt. Dabei können durch die *empirical generation heuristic* oder die *wishful thinking heuristic* Hypothesen generiert werden: Die *emprical generation heuristic* postuliert, daß das in einem Beispiel Gegebene allgemein gültig ist. Die *wishful thinking heuristic* erzeugt durch Abduktion gerade die Hypothesen, die zur Vervollständigung der Erklärung eines Beispiels erforderlich sind. Durch Generalisierungs-, Spezialisierungs- und Selektionsheuristiken werden die so erzeugten Hypothesen dann modifiziert. Dann bestimmt die Wissensintegrationsphase mit welcher Stärke die einzelnen Wissenseinheiten im Gedächtnis Bestand haben.

Die bisherige Modellierung beschreibt, wie aus Texten und Beispielen elementare Programmkonstrukte erworben werden. Mit den in der "ersten Stunde" erworbenen Programmkonstrukten werden nun in der "zweiten Stunde" komplexere Programme gelernt.

3 Die "zweite Stunde": Das Erlernen komplexer Programme

Ziel der im folgenden beschriebenen Modellierung ist es, den Wissenserwerb der "zweiten Stunde" beim Erlernen der Programmiersprache LISP zu modellieren. Diese Wissenserwerbsphase ist dadurch charakterisiert, daß die Lernmaterialien nicht mehr nur die Programmkonstrukte von LISP beschreiben, sondern daß ganze Programme als Lernmaterialien vorgegeben werden. Damit ist ein geändertes Wissenserwerbsziel verbunden, das darin besteht, zielorientiertes Problemlösewissen zu akquirieren, also

Wissen, das effektiv zum Lösen von Programmierproblemen in LISP eingesetzt werden kann. Die Lernprozesse, die für diesen Wissenserwerbsschritt postuliert werden, unterscheiden sich für Anfänger und Fortgeschrittene als Konsequenz ihres unterschiedlichen Vorwissens. Aus diesem Grunde werden zunächst Anfänger und Fortgeschrittene durch ihr Vorwissen charakterisiert. Diese Differenzierung führt zu einer Unterscheidung von zwei Wissensformen auf situativer Ebene, die im Rahmen einer Modellerweiterung als System- und Prozedurwissen eingeführt werden. Um die eigentlichen Wissenserwerbsprozesse konkreter darstellen zu können, wird zunächst eine Beispielsituation eingeführt, anhand deren der Erwerb von Prozedurwissen bei Anfängern und Fortgeschrittenen erläutert wird. Die Modellierung dieser Prozesse erfolgt in einer Computersimulation durch die Anwendung erklärungsbasierter Lernverfahren (Mitchell, Keller & Kedar-Cabelli, 1986; DeJong & Mooney, 1986) unter Verwendung einer formalen Repräsentation des Vorwissens als Theorie.

3.1 Vorwissen bei Programmieranfängern und Fortgeschrittenen

Programmieranfänger und fortgeschrittene Programmierer lassen sich wie folgt durch ihr unterschiedliches Vorwissen charakterisieren: Zu Beginn der "zweiten Stunde" des Erlernens der Programmiersprache LISP wird vorausgesetzt, daß der Lerner, egal ob Anfänger oder Fortgeschrittener, die "erste Stunde" erfolgreich absolviert hat, d.h. sämtliches relevante Wissen aus den jeweiligen Lernmaterialien erworben hat. Hierbei sei unbestritten, daß der Fortgeschrittene diesen Lernerfolg in kürzerer Zeit bzw. mit weniger Lernaufwand erzielt hat als der Anfänger. Da die "erste Stunde" des Wissenserwerbs dadurch gekennzeichnet ist, daß die Konstrukte der Programmiersprache LISP mit ihrer Syntax und Semantik vermittelt werden, haben Anfänger und Fortgeschrittene folglich das gleiche Wissen über das LISP-System erworben. Beide kennen also den gleichen Satz von Programmkonstrukten und den damit verbundenen Interaktionsmöglichkeiten, die das LISP-System zur Verfügung stellt.

Dennoch gibt es Unterschiede zwischen Anfängern und Fortgeschrittenen in bezug auf ihre Problemlösekompetenz beim Programmieren. Anfänger haben vor der "ersten Stunde" keinerlei Erfahrung im Umgang mit einer Programmiersprache, haben also noch keine Programmierprobleme selbständig oder unter Anleitung gelöst. Auch während der "ersten Stunde" wird kein Problemlösewissen vermittelt, so daß Anfänger nicht über zielorientiertes Problemlösewissen für Programmierprobleme verfügen. Fortgeschrittene hingegen haben bereits vor der "ersten Stunde" Programmiererfahrung dadurch, daß sie selbständig Programmierprobleme in einer anderen Sprache als LISP gelöst haben. Dadurch verfügen sie über allgemeines, LISP-unabhängiges Problemlösewissen, welches jedoch zum Verstehen eines Programmes in LISP sehr nützlich ist.

In unserem kognitiven Modell des Wissenserwerbs muß nun gerade der Unterschied zwischen Anfängern und Fortgeschrittenen, also das zielorientierte Problemlösewissen, reflektiert werden. Hierzu werden ähnlich wie bei Ryle (1973), Anderson (1983) und Newell (1982) die folgenden Wissensarten eingeführt:

Systemwissen: Systemwissen beschreibt das gesamte Wissen über ein bestimmtes System. Dies sind die Interaktionsmöglichkeiten, die (von außen) mit dem System bestehen zusammen mit den daraus resultierenden Veränderungen im System. Im LISP-System bestehen die Interaktionsmöglichkeiten im Auswerten bestimmter

Funktionen, wobei bestimmte Symbolbindungen als Eingaben eine Rolle spielen und gegebenenfalls neue Symbolbindungen resultieren. Die Konstrukte der Programmiersprache LISP sind nun gerade als spezielle Funktionen realisiert, so daß das Wissen hierüber als Systemwissen zu bezeichnen ist.

Prozedurwissen: Prozedurwissen beschreibt im Gegensatz zum Systemwissen alles Wissen, das notwendig ist, um gezielt Probleme in einem System zu lösen. Allgemein kann ein Problem in einem System durch die Angabe eines Ausgangs- und eines Zielzustandes beschrieben werden. Die Lösung eines solchen Problems besteht aus einer Menge von (evtl. abstrakten) Aktionen oder Interaktionen, die in dem System zulässig sind, sowie einer Anweisung, wie diese Aktionen anzuwenden sind. Die Ausführung dieser Aktionen in dem System muß den Ausgangszustand des Problems in dessen Zielzustand überführen. Im LISP-System besteht die Beschreibung eines Programmierproblems in der Spezifikation einer Prozedur, die durch Vor- und Nachbedingung im Sinne der Programmverifikation gegeben ist. Lösung des Problems ist ein korrektes LISP-Programm, welches aus den zulässigen LISP-Anweisungen und dem darin implizit vorhandenen Kontrollfluß bestehen.

Prozedurwissen kann zum einen sehr spezifisch sein und die konkrete Lösung eines konkreten Problems beschreiben oder auch genereller, so daß ein allgemeiner Lösungsplan für eine ganze Klasse von Problemen beschrieben ist. In diesem Sinne verfügen also Programmieranfänger über keinerlei Prozedurwissen, fortgeschrittene Programmierer hingegen über allgemeines Prozedurwissen zum generellen Lösen von Programmierproblemen, jedoch nicht über LISP-spezifisches Prozedurwissen. Beim Systemwissen unterscheiden sich Programmieranfänger und Fortgeschrittene nicht. Ziel der "zweiten Stunde" des Wissenserwerbs beim Erlernen der Programmiersprache LISP ist es nun, LISP-spezifisches Prozedurwissen zu erwerben, was den Lernenden in die Lage versetzt, bestimmte Programmierprobleme zu lösen. Hierbei wird sich zeigen, daß der fortgeschrittene Programmierer durch sein Vorwissen im Bereich des allgemeinen Prozedurwissens einen erheblichen Vorteil gegenüber dem Anfänger hat.

3.2 Modellerweiterung: Unterscheidung von System- und Prozedurwissen

Aufgrund der eingeführten Unterscheidung der beiden Wissensarten ist eine Erweiterung des Modelles aus Abbildung 1 erforderlich, so daß sich diese Unterscheidung widerspiegelt.

Abbildung 2 zeigt das so erweiterte Modell. Man erkennt, daß die Grundstruktur des Modelles und die drei Wissensrepräsentationen (Textbasis, Schablonen und Situationsmodell) erhalten geblieben sind. Das neue Lernmaterial, die Programme, werden auf der Seite der Beispiele eingeordnet, da sie ebenfalls in einer Kunstsprache (der Programmiersprache) abgefaßt sind und nicht in natürlichsprachlichem Text. Die Differenzierung zwischen System- und Prozedurwissen ist auf der Ebene des Situationsmodelles eingeführt worden, da erst auf dieser Ebene das Verständnis der erworbenen Konzepte aus Text und Beispiel modelliert ist und sich somit der rein inhaltliche Unterschied, der zwischen System- und Prozedurwissen besteht, dort erst niederschlägt.

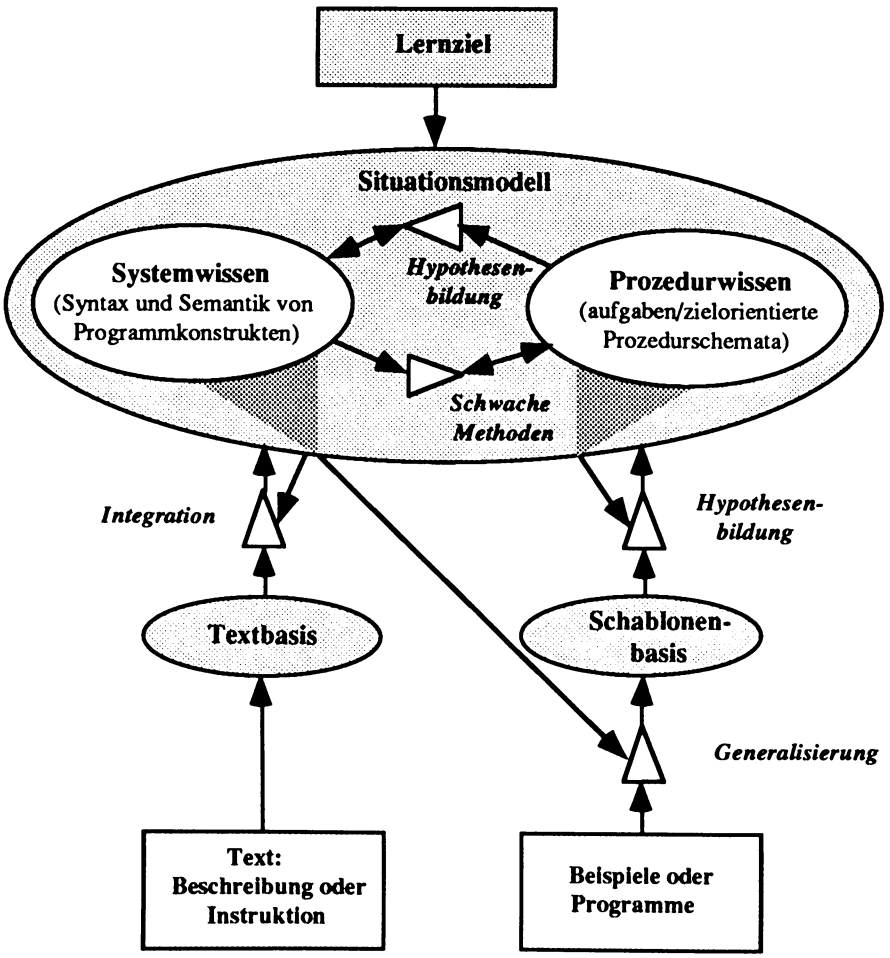

Abbildung 2: Makroskopische Erweiterung des Lernmodells.

System- und Prozedurwissen stehen jedoch nicht lose nebeneinander, sondern es gibt durchaus die Möglichkeit, daß die eine Wissensform in die andere überführt wird. Hierbei werden in dieser Modellerweiterung folgende Transformationen postuliert:

Aus Systemwissen kann Prozedurwissen mit Hilfe von schwachen Methoden generiert werden. Als schwache Methoden werden hierbei im wesentlichen reine Suchprozesse verstanden, wobei im Suchraum aller möglichen Kombinationen von LISP-Konstrukten nach der Lösung für ein Programmierproblem gesucht wird, ohne daß der Suchraum problemspezifisch eingeschränkt wird. Dies entspricht der Situation, in der sich ein Anfänger nach erfolgreicher Absolvierung der "ersten Stunde" befindet, wenn er ein Programmierproblem lösen soll. Lösungen können nur für extrem einfache Probleme gefunden werden, was sich aus der Komplexität des Prozesses begründen läßt. Falls einfache Probleme von guten Lernenden dennoch gelöst werden, so wurde gerade Prozedurwissen direkt aus Systemwissen gebildet.

Der umgekehrte Fall, in dem Systemwissen aus Prozedurwissen abgeleitet wird, liegt bei der folgender Lernkonstellation vor: Ein Fortgeschrittener lernt aus einem

Programm für ein Problem, für das er bereits Prozedurwissen besitzt, zum Beispiel dadurch, daß er dasselbe Problem bereits vorher in einer anderen Programmiersprache gelöst hat. In seinem Lernmaterial tritt jetzt aber ein Programmkonstrukt auf, das er bisher nicht kannte, wozu er also kein Systemwissen besitzt. Aufgrund seiner allgemeinen Kenntnis des Lösungsprinzips, das gerade im Prozedurwissen steckt, ist es ihm aber möglich, Hypothesen darüber zu generieren, was das bislang unbekannte Programmkonstrukt bewirkt und kann so neues Systemwissen generieren. Dies ist jedoch nur durch die Einbettung der unbekannten Aktion in einer größere Menge von bereits bekannten Aktionen möglich.

Die für den Wissenserwerb der "zweiten Stunde" relevanten Prozesse sind jedoch die Generalisierung von Programmen hin zu prozeduralen Programmschablonen sowie die Hypothesenbildung zur Instanziierung allgemeinen Prozedurwissens hin zu LISP-spezifischem Prozedurwissen auf der situativen Ebene. Die Generalisierung von Programmen erfolgt unter Zuhilfenahme von Systemwissen und wird im Modell als ein im wesentlichen deduktiver Prozeß modelliert. Prozedurwissen spielt für diesen Prozeß keine Rolle, so daß dieser gerade von Programmieranfängern durchgeführt werden kann. Die Prozesse der Hypothesenbildung aus Programmen (oder Schablonen) bedürfen aber gerade des allgemeinen Prozedurwissens und können demzufolge lediglich von den Fortgeschrittenen geleistet werden.

3.3 Lernen aus Programmen: Eine Beispielsituation

Um die beiden Prozesse des Erwerbs von Prozedurwissen bei Anfängern und Fortgeschrittenen etwas konkreter darstellen zu können, wird nun zunächst exemplarisch ein Programm vorgestellt und daran der Wissenserwerb allgemein erörtert.

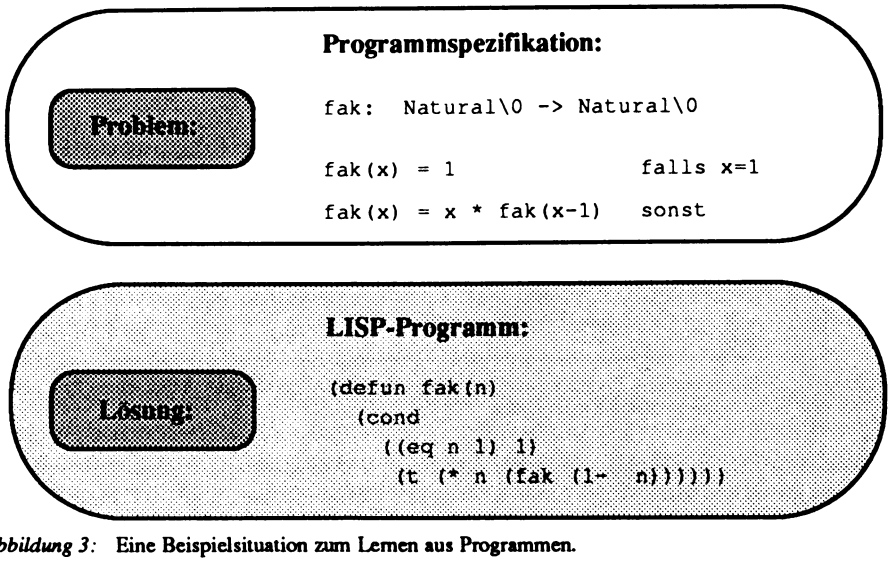

Abbildung 3: Eine Beispielsituation zum Lernen aus Programmen.

Abbildung 3 zeigt eine konkrete Beschreibung eines Programmierproblems. Die Aufgabe besteht darin, die Fakultätsfunktion zu programmieren. Sie wird mit der zu-

gehörigen Lösung gezeigt, in der die Fakultät in Anlehnung an die rekursive Definition auch in LISP rekursiv berechnet wird.

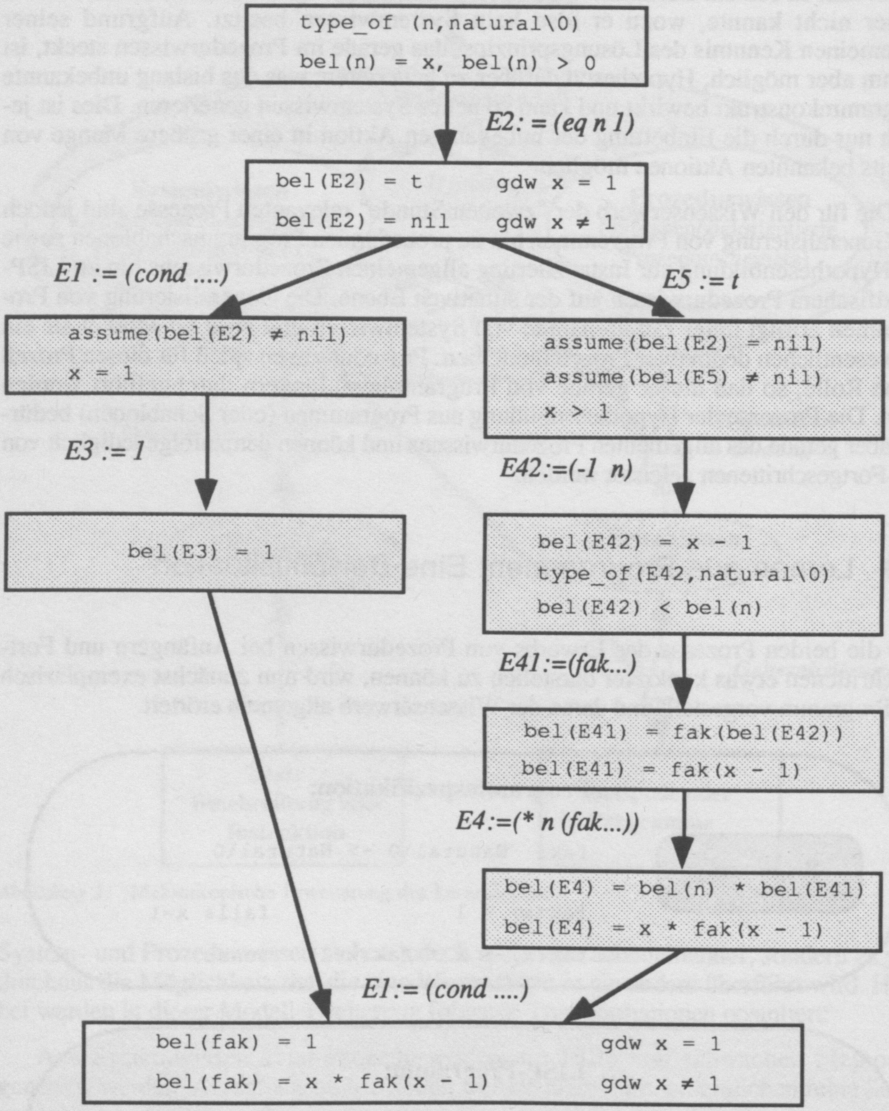

Abbildung 4: Erklärungen mit Systemwissen bei einem Programmieranfänger. Trace einer Programmsimulation in symbolischer Form.

Es stellt sich nun die Frage, was ein Lernender, der eine solche Problemlösung studiert, aus ihr lernen kann. Zum einen könnte der Lernende die Problemlösung einfach nur oberflächlich erinnern, ohne die Lösung überhaupt zu verstehen. Diese Form des Auswendiglernens würde einen Lernenden, der die "erste Stunde" nicht absolviert hat, jedoch nicht von einem solchen Lernenden unterscheiden, der bereits Systemwissen erworben hat. Diese Form des reinen Auswendiglernens ist selbst für Anfänger nicht plausibel.

Im Gegensatz zum Auswendiglernen versucht der Lernende, die dargebotene Problemlösung unter Zuhilfenahme seines vorhandenen Wissens zu erklären, um das Lösungsprinzip zu verstehen. Da Anfänger und Fortgeschrittene für diesen Prozeß jedoch unterschiedliches Vorwissen besitzen, werden auch unterschiedliche Erklärungsstrukturen bei Anfängern und Fortgeschrittenen erzeugt, was letztlich auch unterschiedliche Lernresultate hervorbringt. Im folgenden werden die Wissenserwerbsprozesse bei Anfängern und Fortgeschrittenen separat diskutiert.

3.4 Erwerb von Prozedurwissen bei Anfängern

Anfänger besitzen Systemwissen, welches die Konstrukte der Programmiersprache LISP beschreibt. Dieses Wissen erlaubt es dem Anfänger, den Ablauf des Programmes zu bestimmen und diesen mental zu simulieren. Hierdurch kann eine Erklärung des Programms dadurch erfolgen, daß die Funktionsweise des Programms in seinem zeitlichen Verlauf nachgebildet wird. Solche Abbildungen oder Protokolle von Programmabläufen werden in der Informatik häufig in Form eines Trace zur Fehlersuche in einem Programm eingesetzt. Ein solcher Trace beschreibt den Ablauf eines Programms jedoch mit konkreten Eingabewerten und dokumentiert die konkreten Zwischenergebnisse, die bei der Programmabarbeitung auftreten. Bei der "mentalen Simulation" eines Programms wird jedoch kein konkreter Trace erstellt, sondern die Berechnungsfolgen allgemeiner Art bestimmt. Hierbei werden keine konkreten Zahlenwerte für Berechnungsergebnisse ermittelt, sondern vielmehr allgemeinere Relationen zwischen einzelnen Ergebnisse der Berechnung.

In einer Computersimulation dieses Erklärungsprozesses kann ein solcher symbolischer Trace automatisch aus einer formalen Repräsentation des Systemwissens erzeugt werden (Bergmann, 1992a; Haupenthal, 1992). Hierbei werden Eingabevariablen in das zu simulierende Programm mit symbolischen Werten belegt und die Konsequenzen allgemeiner Art aus dem Systemwissen abgeleitet. Hieraus ergibt sich eine Abfolge von Systemzuständen, die in Abbildung 4 dargestellt ist.

In dieser Abbildung stellt ein Kasten jeweils einen Zustand des LISP-Systems dar, der durch bestimmte Prädikate näher beschrieben ist. Die Pfeile beschreiben den Übergang von einem Systemzustand in den nächsten durch die "mentale" Ausführung einer Programmanweisung. Hierbei beschreibt das Systemwissen, repräsentiert in Form von STRIPS-artigen Operatorbeschreibungen sowie zusätzlichen Inferenzregeln, wie der Folgezustand bestimmt ist. Da dieses Systemwissen in einer deklarativen Form repräsentiert ist, kann auch für den Effekt jeder einzelnen Programmanweisung eine Erklärung gefunden werden, die gerade aus dem Beweis auf der Basis des Systemwissens als Theorie besteht.

Erklärungsbasiertes Lernen (EBL) dieser separaten Erklärungen für die einzelnen Programmanweisungen führt durch eine Generaliserung der konkret ausgeführten Programmanweisung zu einer variabilisierten Anweisung, die eine größere Menge von potentiellen Programmkonstrukten ermöglicht. Hierbei bestimmt EBL eine Menge von *constraints* über die Art der Programmanweisung derart, daß die Effekte der Programmanweisung, die im konkreten Fall aufgetreten sind, auch in jeder Instanzierung des generalisierten Programmkonstruktes auftreten. Die so erhaltene Generalisierung des gesamten Ablauftraces ist in Abbildung 5 dargestellt.

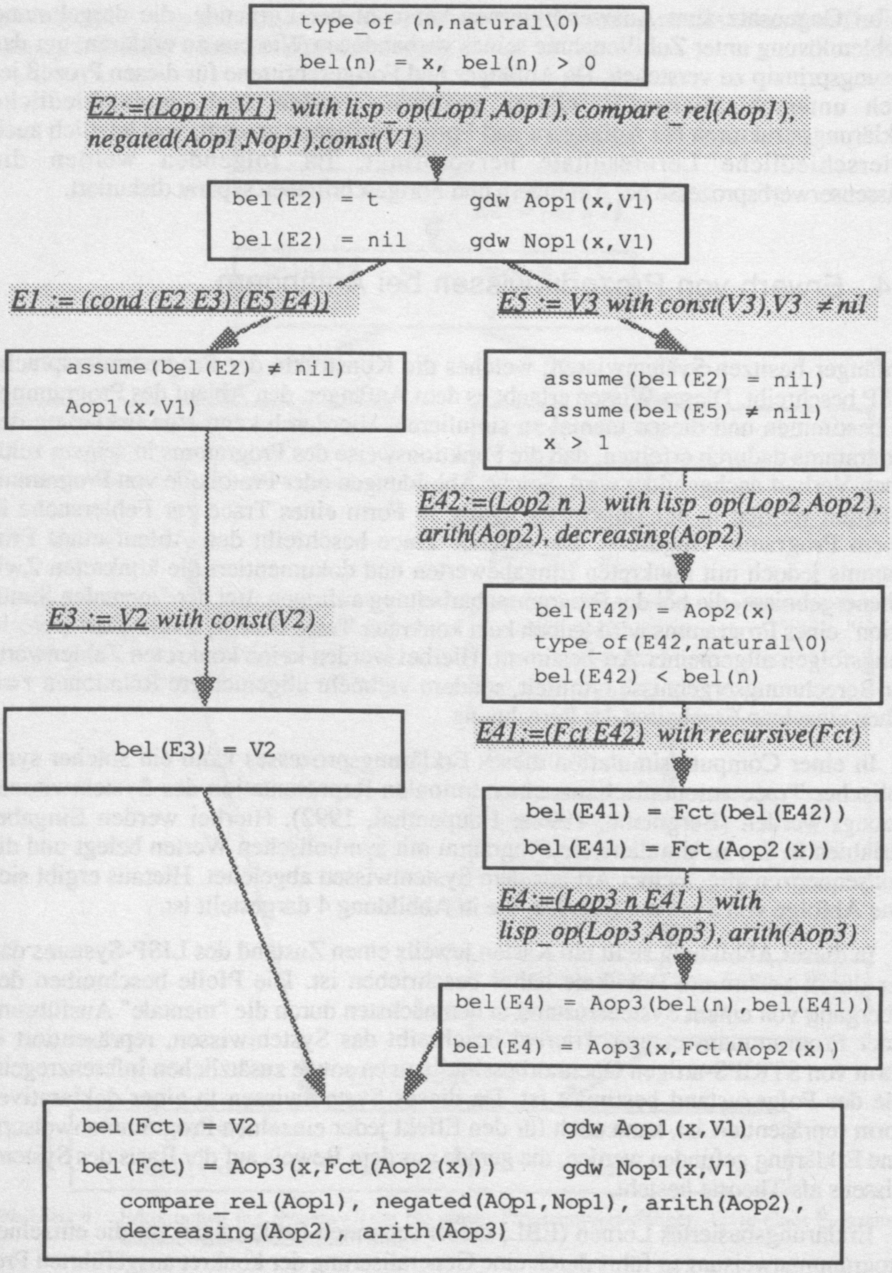

Abbildung 5: Generalisierung der Erklärung bei einem Programmieranfänger.

Die Zusammenfassung der generalisierten Programmkonstrukte unter Berücksichtigung der Abhängigkeiten, die zwischen den Konstrukten bestehen, ermöglicht das Erstellen einer Problemlöseschablone. In dieser Schablone ist die konkrete Problemstellung dadurch zu einem allgemeineren Problemschema generalisiert, daß bestimmte Variablen mit bestimmten *constraints* eingeführt sind (Bergmann, 1992b).

Ebenso werden im Lösungsschema bestimmte Variabilisierungen zugelassen, die jedoch durch die Problemstellung bestimmt sind. Abbildung 6 zeigt die für den in Abbildung 5 dargestellten generalisierten Ablauftrace erzeugte Problemlöseschablone. Diese Schablone spiegelt gerade das Wissen wieder, das der Anfänger, der das Beispielprogramm von Abbildung 3 studiert hat, als Ergebnis des Generalisierungsprozesses speichert.

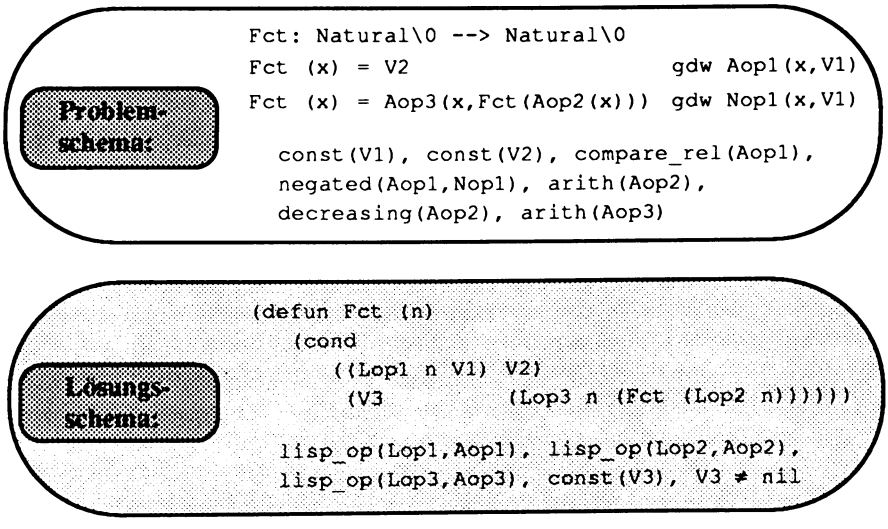

Abbildung 6: Erzeugtes Prozedurwissen eines Programmieranfängers.

3.5 Erwerb von Prozedurwissen bei Fortgeschrittenen

Der Wissenserwerb bei Fortgeschrittenen, die aus dem Beispielprogramm lernen, unterscheidet sich von dem der Anfänger dadurch, daß die Fortgeschrittenen weiteres Wissen in Form allgemeinen Prozedurwissens bereits zur Verfügung haben und gerade dieses zur Erklärung der Funktionsweise des speziellen LISP-Programmes einsetzen können. Für die Erklärung der Fakultätsfunktion von Abbildung 3 kann das folgende allgemeine Problemlöseschema bei einem Fortgeschrittenen vorausgesetzt werden (vgl. Goebel & Vorberg, 1991; Vorberg & Goebel, 1991; Soloway, 1985):

```
Problemschema:
Zu programmieren sei eine rekursiv definierte Funktion, wobei
in der Definition die folgenden Rollen vorkommen:
Tb: eine Terminierungsbedingung
Tf: das Ergebnis im terminierenden Fall
Ar: eine Reduzierung des Argumentes des Funktionsaufrufes
Ra: ein rekursiver Aufruf der Funktion
Rf: die Berechnung des Ergebnisses im rekursiven Fall

Lösungsschema:
Berechne Tb
Wenn Tb erfüllt ist          Dann:   Berechne Tf
                             Sonst:  Berechne Ar
                                     Berechne Ra
                                     Berechne Rf
```

Ein solches allgemeines Lösungsschema, das unabhängig von einer spezifischen Programmiersprache ist, kann der Fortgeschrittene beispielsweise dadurch erworben haben, daß er selbst (vor der "ersten Stunde") rekursive Probleme in einer anderen Programmiersprache gelöst hat und dabei dieses allgemeine Schema identifiziert und zum Problemlösen eingesetzt hat.

Dieses Schema ist auch nützlich, um die Funktionsweise der LISP-Lösung zu verstehen, da sie ebenfalls auf diesem Schema aufbaut. Hierbei kann nun der Fortgeschrittene zunächst anhand der Problemstellung und durch oberflächliche Betrachtung der Lösung die Hypothese generieren, daß dieses Schema gerade zum Versehen der Lösung relevant ist. Es wird dann versucht, diese Hypothese zu bestätigen.

Aufgrund der gegebenen Problembeschreibung können die in diesem Schema vorkommenden Rollen gefüllt werden und die Anwendbarkeit des Problemschemas kann getestet werden. Dadurch sind auch die Rollen, die im Lösungsschema vorkommen, gefüllt, und es verbleibt der Erklärungsbedarf, wie die im Lösungsschema vorkommenden Programmiersprachen-unabhängigen Problemlösungsschritte LISP-spezifisch in dem Programm von Abbildung 3 realisiert sind. Dadurch kann das gegebene Programm erklärt werden, ohne daß eine aufwendige Ablaufsimulation notwendig ist, da durch das allgemeine Lösungsschema wesentliche Teile der Erklärung auf einem allgemeineren Niveau bereits abgedeckt sind. Die daraus resultierende Erklärungsstruktur ist in Abbildung 7 dargestellt.

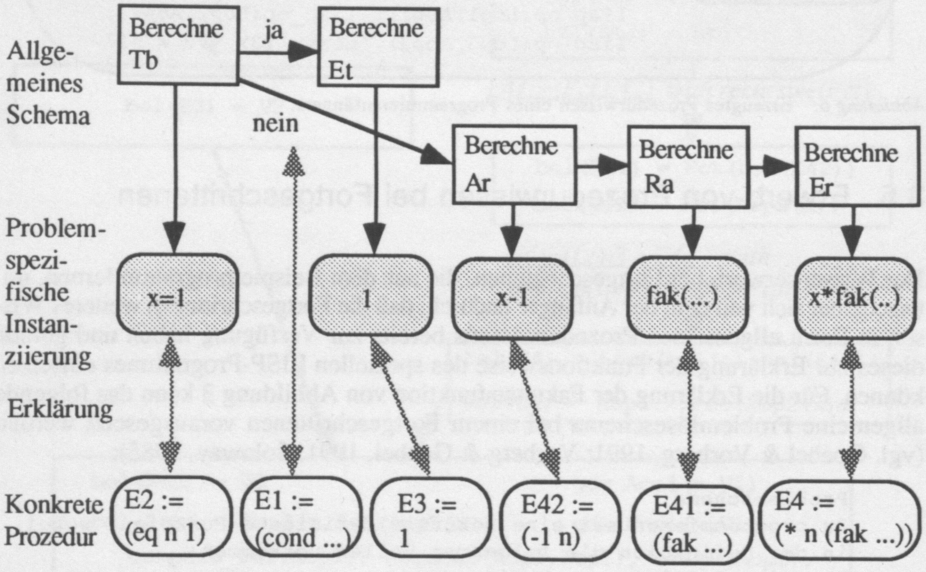

Abbildung 7: Erzeugte Erklärungsstruktur eines fortgeschrittenen Programmierers.

Man erkennt hier das problemspezifisch instanziierte Schema mit den zugehörigen fünf Rollen sowie die Zuordnung dieser generellen Problemlöseschritte zu dem speziellen LISP-Programm. Ergebnis dieses Lernprozesses ist es zunächst, die Anwendbarkeit der allgemeinen Problemlösung auch für den Bereich LISP zu erkennen und zu erlernen, wie dieses allgemeine Lösungsschema LISP-spezifisch instanziiert werden kann. Zur Erklärung dieser LISP-spezifischen Instanziierung kann wieder Gebrauch

von Systemwissen gemacht werden, und im Falle von komplexeren LISP-Realisierungen für eine allgemeine Lösungsaktion kann erneut die Simulationstechnik zur Konstruktion einer Erklärung herangezogen werden.

3.6 Diskussion der Modellaussagen für den Wissenserwerb

Die Wissenserwerbsprozesse beim Lernen aus Programmen, wie sie in der "zweiten Stunde" beim Lernen von LISP auftreten, können nun für Anfänger und Fortgeschrittene wie folgt zusammenfassend verglichen werden: Anfänger und Fortgeschrittene lernen dadurch aus Programmen, daß sie sich die Funktionsweise des Programmes aufgrund ihres bisherigen Wissens erklären und dadurch lernen, bei welchen Problemstellungen ähnliche Lösungen möglich sind. Anfänger und Fortgeschrittene haben zur Erklärung verschiedenes Wissen zur Verfügung. Anfänger verfügen lediglich über Systemwissen, das die Wirkungsweise der einzelnen Konstrukte der Programmiersprache erklärt, wohingegen Fortgeschrittene über allgemeines Prozedurwissen (zielgerichtetes Problemlösewissen) verfügen, das unabhängig von einer spezifischen Programmiersprache ist.

Aufgrund des unterschiedlichen Vorwissens wird das Programm auch unterschiedlich erklärt, wodurch bei Anfängern und Fortgeschrittenen auch unterschiedliche Erklärungsstrukturen für dasselbe Programm entstehen. Anfänger erklären durch eine zeitliche Ablaufsimulation, wohingegen die fortgeschrittenen Programmierer eine eher statische Erklärung für die Instanziierung des allgemeinen Problemlösewissens generieren.

Diese unterschiedlichen Erklärungsstrukturen bewirken auch unterschiedliche Generalisierungen als Ergebnis des Lernprozesses. Anfänger erwerben eine sehr spezifische Problemlöseschablone, Fortgeschrittene jedoch erklären und erwerben ein LISP-spezifisches Instanziierungsprinzip für allgemeines Problemlösewissen, das bereits zu ihrem Vorwissen gehört. Dieser Unterschied beim erworbenen Wissen resultiert ebenfalls in einer unterschiedlichen Problemlösekompetenz bei Anfängern und Fortgeschrittenen. Anfänger können nach dem Wissenserwerb aus Programmen genau solche Probleme lösen, die gerade durch die spezifische Lösungsschablone abgedeckt sind, jedoch keine weiteren. Fortgeschrittene Programmierer sind in der Lage, eine größere Menge allgemeinen Problemlösewissens aus ihrem Vorwissen für LISP spezifisch zu instanziieren und so auch Probleme zu lösen, die anders geartet sind als diejenigen, welche im Lernmaterial vorkamen.

4 Überprüfung der Grundannahmen des Modells durch psychologische Experimente

In mehreren Experimenten wurde die psychologische Adäquatheit des ursprünglichen Modells untersucht. Die Modellierung des Lernens aus Programmen (das "neue" Modell) hat mehrere Ähnlichkeiten mit den Arbeiten von Pirolli (1991) und van Lehn, Jones und Chi (1991). Pirolli berichtet auch über Experimente, die die psychologische Plausibilität dieser Modellierungen belegen. Experimente, die sich auf das Erlernen von Programmen beziehen ("zweite Stunde") werden zur Zeit durchgeführt.

Im folgenden werden Ergebnisse von Experimenten berichtet, die sich mit dem Erwerb von Programmkonstrukten beschäftigten.

4.1 Zielabhängigkeit des Wissenserwerbs

Zur Überprüfung der Zielabhängigkeit des Wissenserwerbs wurden je 20 Versuchspersonen (Vpn) unterschiedliche Lernziele vorgegeben. Die Vpn der einen Bedingung wurden instruiert, daß sie nach dem Lesen eines "kleinen LISP-Manuals" eine Textzusammenfassung schreiben sollten. Den Vpn der anderen Bedingung wurde mitgeteilt, daß sie nach dem Lesen kleine Programmieraufgaben lösen sollten (vgl. Schmalhofer & Glavanov, 1986). Bei allen Vpn wurden dann die Lesezeiten der einzelnen Sätze des Texts registriert. Betrachtet man nun die Lesezeit pro Wort in Abhängigkeit der Hierarchiestufe, auf der sich ein Satz in der bekanntlich hierarchisch strukturierten Textbasis befindet, so ergibt sich folgendes Ergebnis: Für die Vpn, die den Text unter der Zielsetzung der Textzusammenfassung studierten, nehmen die Verarbeitungszeiten (Lesezeit pro Wort) bei niedrigeren Hierarchiestufen ab. Der Satz, der die Wurzel der Hierarchie bildet, wird also am längsten gelesen. Bei den Versuchspersonen der zweiten Bedingung wurde dieser *level's effect* (Cirilo & Foss, 1980) dagegen nicht beobachtet. Daraus läßt sich schließen, daß die Personen mit dem Ziel der Textzusammenfassung mehr Gewicht auf den Aufbau der Textbasis gelegt haben, so daß der bekannte *level's effect* zu beobachten war. Die anderen Personen haben dagegen mehr mentale Ressourcen in den Aufbau des Situationsmodells investiert.

Aufgrund der Lesezeitanalysen würde man ebenfalls erwarten, daß unter der Zielsetzung der Textzusammenfassung die Textbasis stärker ausgeprägt wurde und bei den anderen Vpn das Situationsmodell gedächtnismäßig stärker ausgebildet wurde. Diese Vorhersage konnte in einem Folgeexperiment auch bestätigt werden. Für Satzverifikationsaufgaben können verschiedene Arten von Testsätzen konstruiert werden, so daß sich daraus die Stärke der Gedächtnisspuren für die propositionale Textbasis und das Situationsmodell getrennt errechnen lassen (Schmalhofer, 1986; Schmalhofer & Glavanov, 1986). So läßt sich die Stärke der propositionalen Textbasis aus dem Unterschied in der Antwort zu Paraphrasen und Inferenzsätzen errechnen. Die Stärke der situativen Repräsentation wird auf ähnliche Weise aus den Reaktionsunterschieden zu Inferenzsätzen und falschen Sätzen errechnet.

In den Experimenten von Schmalhofer und Glavanov zeigte sich, daß die Textzusammenfassungsinstruktion tatsächlich zu einer stärker ausgeprägten Textbasis und die Programmierinstruktion zu einem stärker ausgeprägten Situationsmodell führten. Somit konnte die Zielabhängigkeit des Wissenserwerbs sowohl mit Lesezeitanalysen als auch durch das im Gedächtnis gespeicherte Wissen nachgewiesen werden.

4.2 Einfluß des Vorwissens auf den Wissenserwerb

Der Wissenserwerb hängt nicht nur von dem Lernziel, sondern auch von den Vorkenntnissen des Lernenden ab. Wenn eine Versuchsperson bereichsspezifische Vorkenntnisse besitzt, so kann dadurch eine tiefergehendere Verarbeitung des Lernmaterials erfolgen als beim Fehlen solcher Vorkenntnisse. Insbesondere können durch die Vorkenntnisse mehrere nützliche Inferenzen gezogen werden. In solchen Fällen

sollten sich deshalb für Vpn mit Vorkenntnissen längere Lesezeiten ergeben als für Vpn ohne Vorkenntnissen. Hinsichtlich des erworbenen Wissens sollten Vpn mit Vorkenntnissen ein deutlich stärker ausgeprägtes Situationsmodell und Vpn ohne Vorkenntnisse eine vergleichsweise stark ausgeprägte Textbasis aufweisen. Der Aufbau einer Textbasis ist eine allgemeine Fertigkeit und daher nicht von bereichs-spezifischen Vorkenntnissen abhängig. Die Defizite im Aufbau des Situationsmodells führen bei Vpn ohne Vorkenntnissen zu einer Intensivierung bei der Erstellung der propositionalen Textbasis (Mani & Johnson-Laird, 1982).

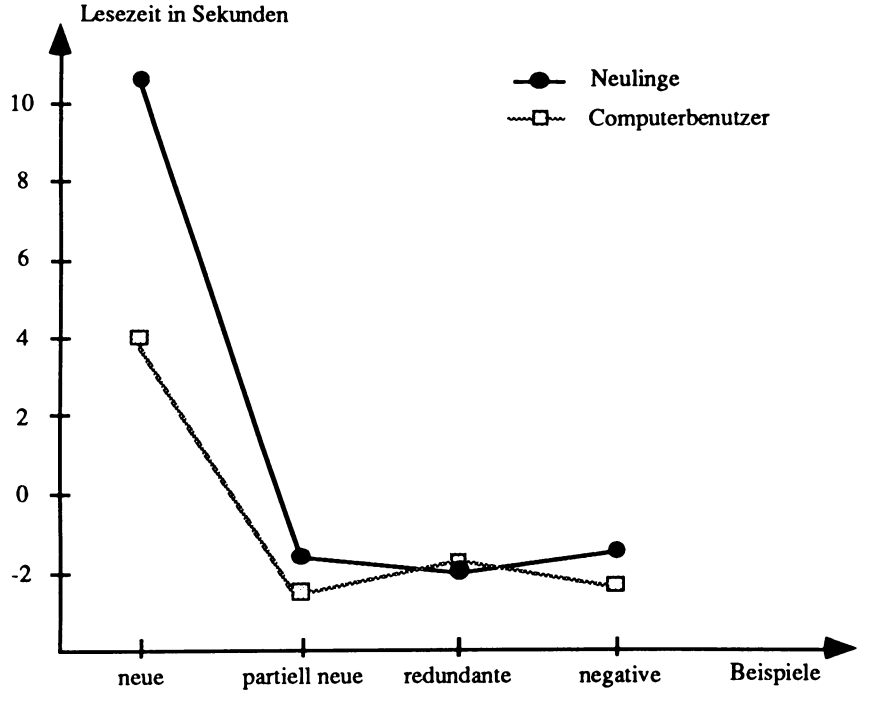

Abbildung 8: Lesezeitprofile für Beispiele bei unterschiedlichen Benutzerkategorien.

Diese Vorhersagen wurden in Experimenten von Schmalhofer & Boschert (1991) geprüft. In dieser Untersuchung studierten Computerbenutzer, die eine Programmiersprache wie z. B. PASCAL kannten (aber nicht LISP), und Computerneulinge, die keinerlei Programmier- oder Computererfahrung besaßen, Text oder Beispiele über LISP-Funktionen.

Die Untersuchungsergebnisse bestätigten die getroffenen Vorhersagen: Größere Vorkenntnisse führten bei bestimmten Segmenten des Instruktionsmaterials wegen einer tiefergehenden Verarbeitung zu einer anfangs längeren Bearbeitungszeit. Wegen dieser tiefergehenden Verarbeitung erwies sich das darauf folgende Material für die Vpn mit Vorkenntnissen als redundant, so daß das Folgematerial schneller gelesen wurde. Für Beispiele, die nach dem Lesen eines zugehörigen Texts studiert wurden, zeigt Abbildung 8 Lesezeitprofile für neue, partiell redundante, redundante und negative LISP-Beispiele bei Computerbenutzern und Neulingen. Die Klassifizierung der Beispiele wurde modellgeleitet durchgeführt (vgl. Schmalhofer & Kühn, 1988).

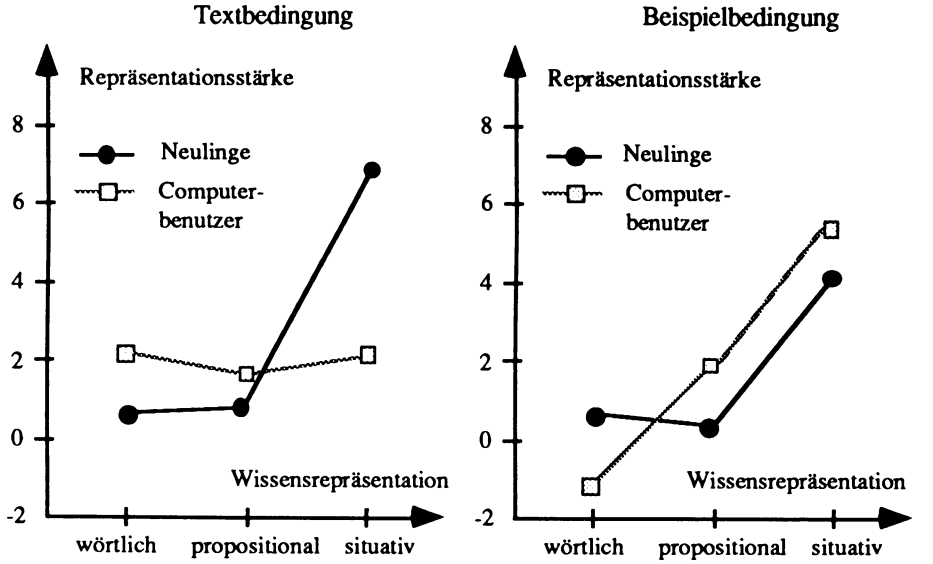

Abbildung 9: Vorhersage hinsichtlich der beim Lernen erstellten Wissensrepräsentationen unter der Text- und der Beispielbedingung.

Wie Abbildung 9 zeigt, wurden auch die Vorhersagen hinsichtlich der beim Lernen erstellten Wissensrepräsentationen bestätigt. Abbildung 9 stellt die Stärke der wörtlichen und propositionalen Textrepräsentation im Vergleich zu der Stärke des Situationsmodells bei Neulingen und Computerbenutzern dar. Die linke Seite der Abbildung zeigt die Wissensrepräsentationen die beim ausschließlichen Lernen aus Text aufgebaut wurde. Rechts sind die Wissensrepräsentationen zu sehen, die beim ausschließlichen Lernen aus Beispielen aufgebaut wurden.

4.3 Wissenserwerb aus informationsäquivalenten Lernmaterialien

Durch das vorgeschlagene Modell können die Gemeinsamkeiten zwischen Lernen aus Text und dem Konzepterwerb aus einer Sequenz von Beispielen besser verstanden werden. Nach dem Modell sollte aus Text und dazu informationsäquivalenten Beispielen das gleiche Situationsmodell aufgebaut werden. Unterschiede würden sich somit nur in den peripheren Wissensrepräsentationen ergeben. Während beim Lernen aus Text auf dem Weg zum Situationsmodell eine Textbasis als peripherere Wissensrepräsentation aufgebaut wird, wird beim Lernen aus Beispielen keine Textbasis erstellt. Selbst wenn Text und informationsäquivalente Beispiele zusammen studiert werden, sollte sich wegen der Informationsäquivalenz der Materialien kein stärkeres Situationsmodell ergeben als aus einem der Lernmaterialien. Diese Vorhersagen konnten in einer Untersuchung von Schmalhofer et al. (1990) experimentell bestätigt werden. Abbildung 10 zeigt die wörtliche und propositionale Textrepräsentation sowie die situativen Repräsentationsstärken der drei Lernbedingungen (Lernen aus Text; Lernen aus informationsäquivalenten Beispielen; Lernen aus Text und Beispielen).

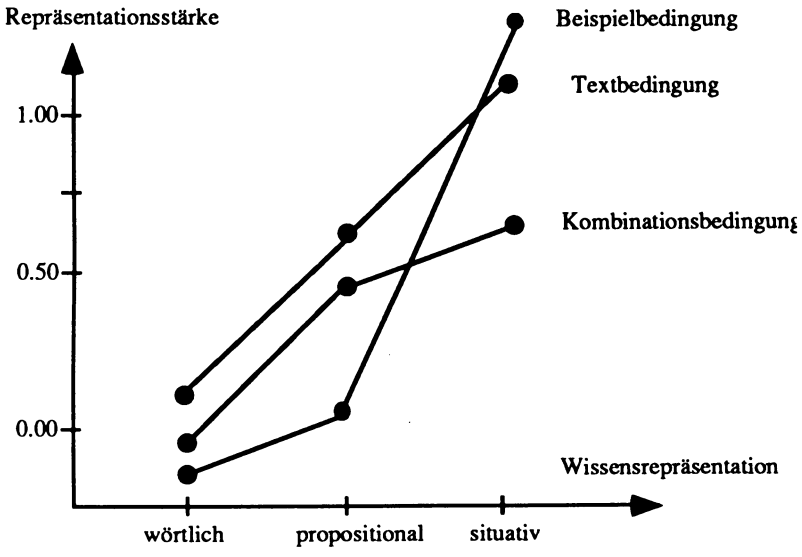

Abbildung 10: Mittlere Repräsentationsstärken von wörtlicher, propositionaler und situativer Information für die drei Bedingungen

5 Diskussion

In der vorliegenden Arbeit wurde das Erlernen von elementaren Programmkonstrukten und der Erwerb von komplexen Kontrollstrukturen, die für rekursive Programme von Bedeutung sind, für den Erwerb der Programmiersprache LISP beschrieben. In den Folgestunden eines Programmierunterrichts können Personen durch Interaktion mit einer Programmierumgebung oder mit Hilfe von Computer-Tutoren (Weber, 1992) die für die Erstellung von Programmen erforderlichen kognitiven Fähigkeiten erwerben. Durch die Arbeiten aus den verschiedenen Forschungsprojekten ergibt sich somit ein zusammenhängendes Bild vom Erlernen einer Programmiersprache von der ersten Stunde bis zum Expertenstatus.

Literatur

Anderson, J.R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J.R., Farrell, R. & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8,* 87-129.

Ausubel, D.P. (1964). Some psychological and educational limitations of learning by discovery. *Arithmetic Teacher, 11,* 290-302.

Bergmann, R. (1992a). Learning from programs for an automated reuse of code. Proceedings of the IEEE Conference on Computer Systems and Software Engineering, COMPEURO '92.

Bergmann, R. (1992b). Knowledge acquisition by generating skeletal plans. In F. Schmalhofer, G. Strube & Th. Wetter (Eds.) *Contemporary Knowledge Engineering and Cognition..*

Bruner, J.S. (1961). The act of discovery. *Harvard Educational Review, 31,* 21-32.

Carroll, J.M. Mack, R.L. & Lewis, C.H (1985) Exploring exploring a word processor. *Human-Computer-Interaction, 1*, 283-307.

Cirilo, R.K. & Foss, D.J. (1980) Text structure and reading time for sentences. *Journal of Verbal Learning and Verbal Behavior, 19*, 96-109.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1* (2), 145-176.

Goebel, R. & Vorberg, D. (1991). Das Lösen rekursiver Programmierprobleme: Ein Simulationsmodell. *Kognitionswissenschaft, 2*, 27-36.

Guthrie, E.R. (1935). *The psychology of learning*. New York: Harper & Row.

Haupenthal, W. (1992). *Erklärungsbasiertes Lernen aus Programmen*. Projektarbeit, Universität Kaiserslautern.

Kintsch, W. (1974). *The representation of meaning in memory*, Hillsdale, N.J.: Erlbaum.

Larkin, J. H. & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11*, 65-100.

Mani, K. & Johnson-Laird, P.N. (1982) The mental representation of spatial descriptions. *Memory and Cognition, 10*, 181-187.

Mitchell, T.M., Keller, R.M. & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1* (1), 47-80.

Neber, H. (1981). *Entdeckendes Lernen*. Weinheim: Beltz.

Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18*, 87-127.

Ryle, G. (1973). *The concept of mind*. Harmondsworth: Penguin.

Schmalhofer, F. (1986). The construction of programming knowledge from system explorations and exlanatory text: A cognitive model. In C.R. Rollinger & W. Horn (Eds.), *GWAI-86 und 2. Österreichische Artificial-Intelligence-Tagung* (S. 152-163). Heidelberg: Springer.

Schmalhofer, F. & Glavanov, D. (1986). Three components of understanding a programmer's manual: Verbatim, propositional, and situational representations. *Journal of Memory and Language. , 25*, 279-294.

Schmalhofer, F. & Kühn, O. (1988). Acquiring computer skills by exploration versus demonstration. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 724-730). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Schmalhofer, F. & Boschert, S. (1991). Learning from text and examples: How situation model, text representation, and template base depend on instruction materials and prior domain knowledge. Manuskript. Kaiserslautern: Deutsches Forschungszentrum für Künstliche Intelligenz.

Schmalhofer, F., Boschert, S. & Kühn, O. (1990). Der Aufbau allgemeinen Situationswissens aus Text und Beispielen. *Zeitschrift für Pädagogische Psychlogie, 4* (3), 177-186.

Schmalhofer, F., Kühn, O. & Messamer, P. (1989). Receptive and exploratory learning in intelligent tutoring systems. *Proceedings of the Fourth Annual Rocky Mountain Conference on Artificial Intelligence* (pp. 71-82). Denver, Colorado.

Soloway, E. (1985). From problems to programs via plans: the content and structure of knowledge for introductory LISP programming. *Journal of Educational Research, 1*, 157-172.

van Dijk, T.A. & Kintsch, W. (1983). *Strategies of discourse comprehension*. Academic Press: New York.

Vorberg, D. & Goebel, R. (1991). Das Lösen rekursiver Programmierprobleme: Rekursionsschemata. *Kognitionswissenschaft, 1*, 83-95.

Weber, G. (1992). Analogien in einem fallbasierten Lernmodell. In K. Reiss, M. Reiss & H. Spandl, *Maschinelles Lernen - Modellieren von Lernen auf Maschinen*. Heidelberg: Springer.