DOI: 10.20378/irb-90967

Technische Universität Berlin

FORSCHUNGSBERICHT
aus dem
INSTITUT FÜR PSYCHOLOGIE

Nr. 94/1

Der Einfluß von Beispielähnlichkeit auf induktive Lernprozesse beim rekursiven Programmieren*

Ute Schmid und Barbara Kaup

10587 Berlin, Dovestraße 1-5 · Telefon (030) 314-223 70

Nr. 94/1
Der Einfluß von Beispielähnlichkeit auf induktive Lernprozesse beim rekursiven Programmieren*

Ute Schmid und Barbara Kaup

* Die vorliegenden Untersuchung ging aus den Arbeiten eines Orientierungsprojektes hervor, das am Institut für Psychologie der TU Berlin in der Abteilung Allgemeine Psychologie I im WS 1992/93 und SS 1993 durchgeführt wurde. Wir bedanken uns bei allen Teilnehmern des Projektes für ihr großes Engagement, das zum Gelingen dieser Untersuchung wesentlich beigetragen hat.

Die Untersuchung wäre sicher nicht ohne die große Unterstützung vieler weiterer Personen möglich gewesen. Für verschiedenste Hilfen bedanken wir uns bei: Dr. Eugen Diesch, Guido Dunker, Prof. Dr. Klaus Eyferth, Dr. Hans Gustat, Christa Henze, Martin Christof Kindsmüller, Harald Kolrep, Dr. Uwe Konerding, Günther Molz, Anja Sachs und Carla Umbach.

Außerdem möchten wir uns natürlich bei allen Studierenden der Berliner Universitäten bedanken, die als Probanden an unserer Untersuchung teilgenommen und tapfer bis zum Ende durchgehalten haben.

Zu guter Letzt bedanken wir uns besonders bei Berry Claus, Nikola Schmidt und Paula Wolff, die wesentliche Vorarbeiten zu dieser Arbeit geleistet haben.

The impact of problem similarity on the inductive acquisition of recursive programming skills

Summary.

In this experiment, novice programming students (N=55) were presented with different example problems while acquiring recursive programming skills. We manipulated the similarity between the examples (source problems) and the problems to be solved (target problems). In addition we measured mapping and adaptation independantly by explicitly providing the results of the mapping process to one group and requiring the other group to do the mapping themselves. We measured success of solving the target problems as well as knowledge acquisition. We measured knowledge acquisition in two ways. Students solved novel transfer problems in a paper-and-pencil task at the end of the study. They also categorized 28 functions at three different times during the study. This categorization task reveales the structure of the students' induced schemes of recursive functions.

We could show that learning conditions have an important influence on problem solving: The greater the similarity between source and target problem the better the problem solving. Subjects who were presented the results of the mapping process showed better problem solving as well. As far as the knowledge acquisition was concerned there was no well defined influence of learning conditions to be shown. On a closer look to the features the subjects used to categorize the functions it got obvious that those who used type of recursion as a classification feature solved more problems and showed better results in the final test than others.

Zusammenfassung.

Es wurde eine Untersuchung durchgeführt, bei der geprüft wurde, auf welche Weise der induktive Erwerb von rekursiven Programmiertechniken durch die Ähnlichkeit von Beispiel- und Zielproblem beeinflußt wird. Dabei wurde davon ausgegangen, daß die Art der Beispiele sowohl den Lösungserfolg rekursiver Aufgaben als auch das induzierte Wissen über den neuen Problembereich beeinflussen.

Die Beispielähnlichkeit wurde systematisch variiert. Zusätzlich wurde die Wirkung von Vergleichs- und Anpassungsprozessen getrennt erfaßt, indem einer Gruppe von Probanden das Ergebnis des Vergleichsprozesses vorgegeben wurde. 55 Programmieranfäger nahmen an vier aufeinanderfolgenden Tagen jeweils 3 Stunden an der Untersuchung teil. Nach Vermittlung der Syntax einer einfachen funktionalen Sprache, sollten die Probanden am dritten und vierten Tag sechs rekursive Funktionen am Rechner programmieren. Dabei wurde ihnen je ein Beispiel, entsprechend der experimentellen Bedingungen vorgegeben. Geprüft wurde der Einfluß der Lernbedingung auf (1) den Prozeß der Aufgabenlösungen und (2) den resultierenden Wissenserwerb. Der Wissenserwerb wurde dabei zum einen durch einen Abschlußtest erfaßt, bei dem verschiedene Transferaufgaben zu lösen waren. Zum anderen wurde die Schemainduktion explizit erfaßt, indem die Probanden zu drei Zeitpunkten jeweils 28 Funktionen sortieren sollten.

Es zeigte sich, daß die Lernbedingung einen bedeutsamen Einfluß auf die Aufgabenlösung hat: Je ähnlicher ein Beispiel zum Zielproblem ist, desto wahrscheinlicher wird die Aufgabe gelöst; die Vorgabe des Vergleichsprozesses führt ebenfalls zu einer erhöhten Lösungswahrscheinlichkeit. Bezüglich des Wissenserwerbs zeigt die Lernbedingung keinen eindeutigen Einfluß. Betrachtet man die Merkmale nach denen die Probanden sortieren, zeigt sich, daß Probanden, die das Merkmal "Rekursionstyp" zur Klassifikation verwenden, sowohl mehr Aufgaben lösen als auch eine höhere Transferleistung im Abschlußtest zeigen.

1 Einleitung

In der-Wissenserwerbsforschung wird induktiven Lernprozessen zunehmend größere Bedeutung zugemessen (vgl. Anderson & Thompson 1989; Reed & Bolstedt 1991; Ross & Kennedy 1990; Holland, Holyoak, Nisbett & Thagard 1987). Dabei wird häufig betont, daß induktives Lernen im Gegensatz zu deduktiven Lernen dem *natürlichen* Lernen nahe kommt (vgl. Fodor 1980). Wissenserwerb ist nur in sehr begrenzten Bereichen ein bloßes Anhäufen von neuen Fakten. Beim induktiven Lernen geht es vornehmlich darum, für die Fragestellung relevantes Wissen im Gedächtnis aufzufinden und dieses als Ausgangspunkt für den Wissenserwerb heranzuziehen.

Sofern bei der Anwendung dieses Wissens die erkannte **Ähnlichkeit** zwischen diesem Wissen und der aktuellen Problemstellung eine zentrale Rolle spielt, kann dieser Prozeß als *Analoges Lernen* oder als *Analoger Transfer* bezeichnet werden. Analoger Transfer wird allgemein als das Übertragen eines Beispielproblems auf ein aktuelles Zielproblem definiert. Gentner (1983) betont, daß dabei **strukturelle** Ähnlichkeiten zwischen Beispiel- und Zielproblem vorhanden sein müssen. Sie betrachtet in ihrem Modell vornehmlich Analogen Transfer zwischen unterschiedlichen Wissensbereichen (*between domain analogical transfer*). Ihre Analysen treffen jedoch ebenfalls auf Analogen Transfer innerhalb von Wissensbereichen zu (*within domain analogical transfer*). Während für Konzepterwerb und das Verstehen technischer Zusammenhänge hauptsächlich Untersuchungen zu *between domain analogies* durchgeführt wurden (Holyoak & Thagard 1989; Forbus & Gentner 1986), wurden bei Untersuchungen zum Erwerb kognitiver Fertigkeiten (z.B. Lösen mathematischer Textaufgaben, Programmieren) vor allem *within domain analogies* betrachtet.

Beim Analogen Transfer lassen sich nach Novick & Holyoak (1991) vier Teilprozesse unterscheiden: Zunächst muß ein dem Zielproblem strukturell ähnliches Beispielproblem gefunden werden (*retrieval*), dann müssen die strukturellen Ähnlichkeiten zwischen Ziel- und Beispielproblem erkannt (*mapping*) und bei der Anpassung des Lösungsweges berücksichtigt werden (*adaptation*). Das Ergebnis des Analogen Transfers schließlich ist der Aufbau eines abstrakten Schemas der Problemklasse, die durch Ziel- und Beispielproblem repräsentiert ist (*learning*).

Das Retrieval scheint keine Auswirkung auf den Schemaerwerb zu haben, da sich keine Unterschiede im Transfer ergeben, wenn Beispielprobleme vorgegeben bzw. selbständig gefunden werden (Ross & Kennedy 1990). Mapping und vor allem Adaptation sind dagegen notwendige Prozesse für die Schemainduktion (Novick & Holyoak 1991).

Die Ähnlichkeit zwischen Beispiel- und Zielproblem beeinflußt sowohl den Lösungserfolg der Zielaufgabe (vgl. Reed & Bolstadt 1991; Novick & Holyoak 1991) als auch den Schemaerwerb (Pirolli & Anderson 1985; Novick & Holyoak 1991). Allerdings gibt es bislang kaum Untersuchungen, die den Einfluß der vorhandenen Ähnlichkeiten auf die Induktion eines abstrakten Schemas systematisch prüfen. Uns ist nur ein einziges Experiment bekannt, bei dem die strukturelle Ähnlichkeit zwischen Ziel- und Beispielproblem variiert wird (Reed & Bolstadt 1991). Die Autoren erfassen jedoch lediglich den Aufgabenlösungserfolg, nicht aber den resultierenden Schemaerwerb und zeigen, daß sich die Lösungswahrscheinlichkeit für Zielprobleme erhöht, wenn Beispielprobleme dargeboten werden, die in der Komplexität den Zielproblemen ähnlich sind. Generell wird in Untersuchungen zum Analogen Lernen der Schemaerwerb nur indirekt erhoben (Novick & Holyoak 1991; Ross & Kennedy 1990). Ross & Kennedy betrachten lediglich die Fähigkeit analoge Zielprobleme lösen zu können, was ihrer Meinung nach Rückschlüsse auf den Schemaerwerb zuläßt. Novick & Holyoak erheben den Schemaaufbau, indem sie mit Testaufgaben am Ende der Lernphase das erworbene Wissen über die Problemklasse abprüfen. Sie können damit zwar generelle Aussagen über die Qualität der erworbenen Schemata machen. Unserer Meinung nach wären jedoch zusätzlich Informationen über die **Struktur** der aufgebauten Schemata interessant.

Die genannten Arbeiten untersuchen Analogen Transfer bei der Lösung mathematischer Textaufgaben. Die Bedeutung des Analogen Transfers ist aber auch für das rekursive Programmierenlernen betont worden (vgl. Anderson, Farrell & Sauers 1984, Pirolli & Anderson 1985). So geht die Forschergruppe um Anderson davon aus, daß bei der Lösung anfänglicher Programmieraufgaben den konkreten Beispielen eine große Bedeutung zukommt (Pirolli & Anderson 1985). Erst auf einer späteren Stufe des Wissens- und Fertigkeitserwerbs kommen die, in ACT* (vgl. Anderson 1983) angenommenen, Prozesse der Wissenskompilierung und Generalisierung zur Anwendung (Anderson & Thompson 1989). Pirolli & Anderson

(1985) simulieren verbale Protokolle von Programmieranfängern unter Einbeziehung der in ACT* formulierten Lernprinzipien.

Es wird angenommen, daß das Lernen am Beispiel zum Aufbau von neuen Produktionen führt, die für das Lösen analoger Zielprobleme anwendbar sind. Dabei hängt es vom Abstraktionsgrad bzw. der Generalität der Repräsentation des Beispiels ab, wie abstrakt bzw. generell die aufgebaute Produktionsregel ist und für welche Zielprobleme die neue Produktion damit verwendbar ist. Die Generalität der Repräsentation des Beispieles und damit die Generalität der aufgebauten Produktionen ist beeinflußt durch die Arten der Ähnlichkeiten, die für den Mapping-Prozeß herangezogen wurden. Gehen nur Oberflächenähnlichkeiten in den Mapping-Prozeß mit ein, so ist die Produktion sehr spezifisch und nur für Zielprobleme mit ähnlichen Oberflächenmerkmalen anwendbar. Werden jedoch im Mapping-Prozeß tieferliegende strukturelle Ähnlichkeiten berücksichtigt, so wird eine Produktion aufgebaut, die für die Lösung einer ganzen Klasse von Zielproblemen herangezogen werden kann. Allerdings muß eine sehr generelle Produktion auf das Spezifizieren von Detailwissen verzichten und hilft, insofern sie die einzige aufgebaute Produktion ist, bei der konkreten Lösung von Problemen nicht weiter. Pirolli & Anderson (1985) simulieren analogen Transfer und dessen Ergebnis zwar mit Produktionsregeln, nähern sich jedoch stark einer schematheoretischen Herangehensweise an, indem sie aufgebaute Produktionsregeln unterschiedlicher Abstraktheit betrachten, die hierarchisch ineinander geschachtelt sein können und aus einer Art schematischen Repräsentation der Beispiele gebildet werden.

Anderson und Thompson (1989) führen in ihrer PUPS Architektur zur Modellierung des Analogen Transfers beim rekursiven Programmieren eine Schemakomponente ein. Beispiele und Programmierwissen sind bei Anderson & Thompson (1989) als Schemata repräsentiert. Diese Schemata haben Einträge für Syntax (*form*) und Semantik (*function*) der entsprechenden LISP-Funktion. Ein analoger Schluß beim Programmierenlernen betrifft die Relation zwischen Syntax und Semantik im Beispielproblem und Syntax und Semantik im Zielproblem. Dabei ist meistens im Zielproblem die Semantik der zu programmierenden Funktion gegeben, und die Syntax dieser Funktion gesucht. Anderson und Thompson (1989) postulieren im Prinzip dieselben Teilprozesse wie Novick und Holyoak (1991). Der

erfolgreiche Analoge Transfer vom Beispielproblem zum Zielproblem führt bei Anderson und Thompson (1989) einerseits zum Ausbau des in Schemata repräsentierten Wissens über die Problemklasse, als auch zur Prozeduralisierung dieses deklarativen Wissens und zur Generalisierung.

Im Folgenden wird eine Untersuchung vorgestellt, bei der geprüft wird, auf welche Weise der induktive Erwerb von rekursiven Programmiertechniken durch die Ähnlichkeit von Beispiel- und Zielproblem beeinflußt wird. Dabei wird die Ähnlichkeit systematisch stufenweise variiert. Zusätzlich wird die Wirkung der Teilprozesse *mapping* und *adaptation* getrennt erfaßt, indem einer Gruppe von Probanden die Ergebnisse des Mapping-Prozesses vorgegeben werden. Dabei soll der Einfluß der Ähnlichkeit sowohl auf den Lösungserfolg der Zielprobleme (Reed & Bolstadt 1991), als auch auf die Schemainduktion erhoben werden. Die induzierten Schemata werden dabei einerseits implizit durch generalisierende Testaufgaben (Novick & Hoyoak 1991) und andererseits explizit durch Sortieraufgaben erfaßt.

Während der Einfluß von Mapping und Adaptation auf den Analogen Transfer explorativ betrachtet wird, haben wir für den Einfluß der Beispielähnlichkeit konkrete Hypothesen: Wir gehen erstens von der Annahme aus, daß ähnliche Beispiele sich günstig auf die Lösungswahrscheinlichkeit von Zielaufgaben auswirken (vgl. Reed & Bolstadt 1991); zweitens leiten wir aus den Analysen von Pirolli & Anderson (1985) ab, daß eine mittlere Ähnlichkeit zwischen Beispielen und Aufgaben sich günstig auf das aufgebaute Wissen auswirken sollte.

2 Methode

2.1 Design und Hypothesen

In der Untersuchung wird der Einfluß der Beispielähnlichkeit und des Mapping-Prozesses auf den Erwerb von rekursiven Programmiertechniken geprüft. Die Beispielähnlichkeit wird dabei systematisch in fünf Stufen variiert (Faktor Beispielähnlichkeit: von I: sehr ähnlich bis V: sehr unähnlich). Der Mapping-Prozeß wird den Probanden entweder abgenommen oder muß von ihnen selbständig durchgeführt werden (Faktor Mapping: 0: keine Vorgabe; 1: Vorgabe).

Es wird dabei die Wirkung dieser Beispielbedingungen sowohl auf die Programmierfertigkeiten als auch auf den Wissenserwerb erhoben. Die Programmierfertigkeit wird durch die Anzahl gelöster Zielaufgaben erfaßt. Der Wissenserwerb wird über die Transferleistung in einem Abschlußtest geprüft. Zusätzlich wird die Schemainduktion direkt erfaßt, in dem zu drei Lernzeitpunkten Sortierversuche durchgeführt werden. Es wurde angenommen, daß die Beispielähnlichkeit und das Mapping sowohl den Lösungserfolg als auch die Transferleistung und die Schemainduktion beeinflussen.

Dabei gehen wir von folgenden Hypothesen aus: Die Lösungswahrscheinlichkeit bei Programmieraufgaben erhöht sich mit zunehmender Nähe der Beispiele zur Zielaufgabe. Die Vorgabe des Mappings erhöht ebenfalls die Lösungswahrscheinlichkeit. Der Schemaaufbau und die Transferleistung werden am stärksten durch Beispiele mittlerer Ähnlichkeit zur Zielaufgabe unterstützt. Bezüglich der Wirkung des Mappings auf den Wissenserwerb lassen sich zwei Vorstellungen gegenüberstellen: Durch Markierung gleichbleibender Strukturen kann ein Schema für rekursive Funktionen möglicherweise leichter aufgebaut werden, so daß sich dadurch die Transferleistung verbessert. Andererseits könnte die Markierung gleichbleibender Strukturen zwar den unmittelbaren Lösungsprozeß erleichtern, jedoch zu unreflektierterem Lösen der Zielaufgaben verführen, so daß eine Schemainduktion vielleicht wenig gefördert wird und somit die Möglichkeit zum Transfer begrenzt bleibt.

2.2 Konstruktion der Beispiele

Die unterschiedliche Beispielähnlichkeit wird durch Anzahl und Schwierigkeit der Transformationen gegenüber dem Zielproblem realisiert. Um die Konstruktion der Beispiele nachvollziehen zu können, sind Grundkenntnisse über die Struktur rekursiver Funktionen notwendig. Eine kurze Einführung in das Prinzip der Rekursion, sowie die Umsetzung dieses Prinzips in die Syntax einer funktionalen Programmiersprache ist Anhang A zu entnehmen.

Im folgenden wird ausgeführt, auf welche Art die fünf Stufen der Beispielähnlichkeit konstruiert wurden: In der ersten Ähnlichkeitsstufe wurde die Struktur der Zielaufgabe beibehalten und lediglich Konstanten bzw. definierte Operationen durch Ausdrücke gleichen Typs ersetzt. Bei der zweiten Ähnlichkeitsstufe wurde **entweder** die Struktur der Abbruchbedingung **oder** des direkten Falls variiert, indem der entsprechende Ausdruck um

eine Operation erweitert wurde. In der dritten Ähnlichkeitsstufe wird der rekursive Aufruf um eine Operation erweitert. Damit wird die zentrale Struktur rekursiver Funktionen verändert. Die vierte Ähnlichkeitsstufe ist eine Kombination der Stufen zwei und drei. In der fünften Stufe wurde das Beispiel der Stufe vier um eine zusätzliche Operation erweitert. Diese Operation wurde bei dem in der zweiten Ähnlichkeitsstufe nicht manipultierten Ausdruck eingefügt. Anhang B zeigt exemplarisch alle Beispielähnlichkeiten für die zweite Zielaufgabe.

Ein entscheidender Teilprozeß beim Analogen Lernen ist das Erkennen der Übereinstimmungen zwischen Beispiel- und Zielaufgabe. Dieser Prozeß wird der Hälfte der Probanden dadurch abgenommen, daß die identischen Strukturen markiert sind. Die Probanden wurden darauf hingewiesen, daß das Fettgedruckte des Beispiels für die zu lösende Zielaufgabe übernommen werden kann. Die im Anhang B dargestellten Beispieltypen entsprechen der Bedingung mit Mapping-Hilfe.

Während eine Zielaufgabe bearbeitet wurde, war das Beispiel die ganze Zeit im rechten Teil des Bildschirms eingeblendet. Eine ausführlichere Beschreibung der Lernumgebung findet sich in Angang A. Alle rekursiven Zielaufgaben sind in Anhang C aufgeführt.

2.3 Beschreibung der Stichprobe

An der Untersuchung nahmen 55 Studierende der Psychologie teil, die zufällig. auf die 10 experimentellen Bedingungen verteilt wurden (siehe Tab. 1). Die Probanden waren im Durchschnitt 26 Jahre alt und befanden sich im zweiten Semester. Dabei waren 27 Probanden männlichen und 28 weiblichen Geschlechts. Die Geschlechtszugeh örigkeit war über die Bedingungen gleichverteilt. Weitere in einem demographischen Fragebogen erhobenen Kenngrößen, wie Computervorerfahrung und Mathematikkenntnisse, verteilen sich ebenfalls annähernd gleich über die Bedinungen. Für die Teilnahme bekamen die Probanden Versuchspersonenstunden und ein Entgeld von ca 40 DM.

2.4 Durchführung

Die Probanden nahmen an vier aufeinanderfolgenden Tagen für je drei Stunden an den Sitzungen teil. Eine Übersicht über den Untersuchungsablauf ist Tabelle 2 zu entnehmen.

TABELLE 1

Verteilung der Probanden auf die Beispielbedingungen

		Beispielähnlichkeit					
		I	II	III	IV	V	$\sum$
Mapping	0	5	5	5	6	6	27
	1	5	7	6	5	5	28
	$\sum$	10	12	11	11	11	55

Am ersten Tag wurden zunächst demographische Daten sowie Computer- und Mathematikkenntnisse erhoben. Danach begann die Einführungsphase, die sich über die ersten beiden Tage erstreckte. Die Syntax einer einfachen funktionalen Programmiersprache (siehe Anhang A) wurde durch Frontalunterricht vermittelt. Am Ende jeder Sitzung sollten die Probanden das Gelernte am Rechner umsetzen und selbständig nicht rekursive Funktionen programmieren. Am dritten Tag wurden die Syntaxkenntnisse in einem aus 28 Items bestehenden Test abgeprüft. Die Testergebnisse wurden als Kontrollvariable verwendet, um zu gewährleisten, daß die Beherrschung der Syntax über die Beispielbedingungen hinweg vergleichbar ist. Anschließend wurde das Konzept der Rekursion vermittelt und der erste Durchgang der Sortierversuche durchgeführt (siehe Abschnitt 2.4). Diese Sortierungen dienen als Ausgangslage für die im Verlauf des Lernprozesses noch zweimal erhobenen Sortierungen mit denen der Schemaaufbau in Abhängigkeit von der Beispielbedingung erfaßt werden soll.

Diese Sitzung wurde mit dem selbständigen Programmieren dreier endrekursiver Funktionen (Anhang C) abgeschlossen, wobei als Lernhilfen nun nur noch die konstruierten Beispiele und ein Merkblatt zur Syntax zur Verfügung standen. Zu Beginn des vierten Tages wurde zum zweiten Mal sortiert, dann waren drei teil-rest-rekursive Funktionen (Anhang C) zu programmieren, und danach fand der dritte Durchgang der Sortierversuche statt. Dann folgte der Abschlußtest (siehe Abschnitt 2.4), dem sich noch die Bearbeitung eines Evaluationsfragebogens anschloß, mit dessen Hilfe der subjektive Lernerfolg erfaßt und die Nützlichkeit der Beispiele beurteilt werden sollte.

TABELLE 2

Untersuchungsablauf

1. Tag	2. Tag
Begrüßung > Erhebung von demographischen Daten, Computervorerfahrung und Mathematikkenntnissen (30 min) **Einführungsphase** Unterricht zu Lektion 1: Def. von Funktionen, Datentypen, arithm. Anweisungen (60 min) Pause (10 min) Hinweise zur PC Bedienung Lektion 1: Programmieren von zwei nicht-rekursiven Funktionen (60 min)	Wiederholung der Syntax aus Lektion 1 (30 min) Unterricht zu Lektion 2: Bool'sche Ausdrücke, Bedingte Anweisungen, Listen (60 min) Pause (10 min) Lektion 2: Programmieren von vier nicht-rekursiven Funktionen (90 min)
3. Tag	**4. Tag**
> **Syntaxtest** (30 min) Einführung der Rekursion (30 min) **Experimentelle Phase** **> 1. Durchgang der Sortierversuche** (30 min) Pause (10 min) **Lektion 3:** Programmieren von drei endrekursiven Funktionen (75 min)	**> 2. Durchgang der Sortierversuche** (30 min) **Lektion 4:** Programmieren von drei teilrestrekursiven Funktionen (75 min) Pause (10 min) **> 3. Durchgang der Sortierversuche** (30 min) > **Abschlußtest** (30 min) > Evaluationsfragebogen (15 min) Entlohnung

Grau hinterlegt = Experimentelle Phase

2.5 Erfassung der Programmierfertigkeit und des Wissenserwerbs

2.5.1 Programmierfertigkeit

Insgesamt waren sechs rekursive Zielaufgaben zu bearbeiten. Konstruierten die Probanden innerhalb von 25 Minuten die richtige Lösung, wurde die nächste Aufgabe gestellt, waren sie innerhalb dieser Frist dazu nicht in der Lage, wurde die Zielaufgabe als falsch bewertet, automatisch die korrekte Lösung und anschließend die nächste Zielaufgabe eingeblendet.

Als Kennwert für die Programmierfertigkeit wurde die Anzahl der in der vorgegebenen Zeit korrekt gelösten Zielaufgaben verwendet. Die abhängige Variable Aufgabenlösung kann also Werte zwischen null und sechs annehmen.

2.5.2 Wissenserwerb I: Abschlußtest

Der Abschlußtest bestand aus neun Aufgaben mit insgesamt 22 zu lösenden Items, die sich in vier Gruppen klassifizieren lassen. Sieben Items betrafen grundlegendes Wissen über Rekursion (Wissensitems). Vier mal waren angegebene Funktionen hinsichtlich ihres semantischen Gehalts zu beurteilen (Semantikitems). Bei acht Items sollte die Ein- oder Ausgabe von konkreten Werten bei vorgegebener Funktion berrechnet werden (Ein-Ausgabeitems). Der klassische Fall von Transfer wurde durch drei Items gemessen, bei denen eine rekursive Funktion programmiert werden sollte (Programmieritems). Die Probanden hatten 30 Minuten Zeit, den auf Papier dargebotenen Test zu bearbeiten.

2.5.3 Wissenserwerb II: Sortierversuche

Durch die mehrmalige Durchführung der Sortierversuche sollte erfaßt werden, ob die Probanden Schemata über rekursive Funktionen aufbauen. Dabei gehen wir von der Annahme aus, daß Experten Wissen über rekursive Funktionen in einer Schemahierarchie repräsentieren (Vorberg & Goebel 1991). Eine Expertenhierarchie, in die sich die von uns verwendeten Zielaufgaben einordnen lassen, ist in Abbildung 1 dargestellt. Bei erfolgreichem Analogem Transfer während des Lernprozesses müßten die Probanden Wissensstrukturen erwerben, die sich dieser Hierarchie annähern. Anhand unserer Expertenhierarchie konstruierten wir 28 Funktionen namens "test". Die Probanden wurden instruiert, die Funktionen nach ihrer Ähnlichkeit zu sortieren, wobei ihnen freigestellt war, linear oder hierarchisch zu strukturieren und die Merkmale, nach denen sie sortierten, zu benennen oder nicht. Sie wurden ebenfalls darauf hingewiesen, daß es keine "richtigen" oder "falschen" Lösungen gäbe, sondern daß nur die Art der Sortierugen von Interesse sei. Die Probanden hatten 30 Minuten Zeit, die auf Karteikarten gedruckten, numerierten Funktionen nach Belieben zu ordnen und die Funktionsnummern auf ein Ergebnisblatt zu notieren.

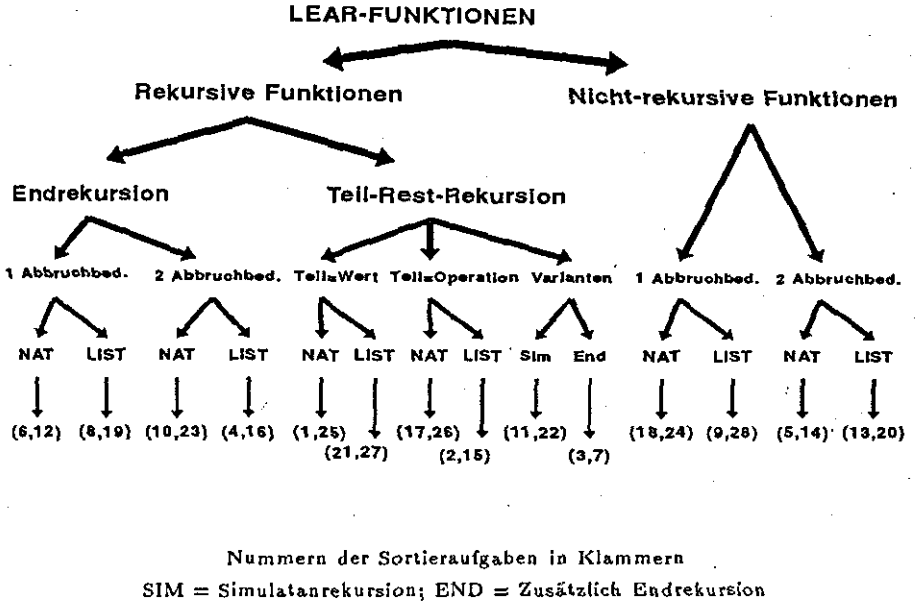

Abb. 1 Expertenhierarchie rekursiver Funktionen

Ein erster Sortierdurchgang vor der experimentellen Bedingungsvariation diente als Ausgangslage für zwei weitere Sortierungen. Die zweite Sortierung wurde nach der Bearbeitung der ersten drei Zielaufgaben erhoben, die dritte nach Bearbeitung aller Zielaufgaben.

Die Sortierungen wurden erstens bezüglich der Distanz zur Expertenhierarchie und zweitens hinsichtlich der verwendeten Sortiermerkmale analysiert.

Distanz zur Expertenhierarchie

Mit der Distanz zur Expertenhierarchie sollte betrachtet werden, wie ähnlich die von den Probanden erworbenen Schemata dem eines Experten seien. In einem ersten Schritt wurden alle Sortierungen in Baumstrukturen transformiert, wobei lineare Sortierungen als Bäume, die nur aus Wurzel und Blättern bestehen, dargestellt wurden. Wir entwickelten ein Verfahren, welches ermöglicht, über die Baumstrukturen vergleichbare Distanzmaße zwischen den einzelnen Funktionen zu berrechnen. Die Distanz zwischen zwei Funktionen ergibt sich aus der Entfernung der beiden Funktionen in der Baumstruktur. Die Kanten, die durchlaufen

werden müssen, um von der einen Funktion zu der anderen zu gelangen, werden aufaddiert und am längsten Weg des Baumes relativiert. Dadurch erhält man für verschiedene Sortierungen vergleichbare Distanzmaße zwischen Null (sehr ähnlich) und Eins (maximal unähnlich). Diese wurden in Matrizen aus 28 × 28 Funktionen übertragen und konnten so mit der aus der Expertenhierarchie gewonnen Matrix der Funktionsähnlichkeiten verglichen werden.

Erfassung der verwendeten Sortiermerkmale

Neben der Erfassung der Distanz zur Expertenhierarchie sollten auch die Merkmale erhoben werden, nach denen die Probanden die Funktionen beurteilten. Da die Probanden nicht gezwungen waren, die gebildeten Kategorien zu benennen, wurde ein Verfahren entwickelt, das die Zuweisung von inhaltlichen Merkmalen zu den gebildeten Kategorien ermöglicht. Zu diesem Zweck wurde aufgrund der Sortierergebnisse eine Liste der verwendeten Merkmale erstellt. Insbesondere gehen alle in der Expertenhierarchie verwendeten Klassifikationskriterien in diese Liste ein, die dann um zusätzlich von den Probanden verwendete Kriterien ergänzt wurde. Jede der 28 Funktionen kann dann in Form eines Merkmalvektors in allen Merkmalen beschrieben werden. Schließlich wurden alle von einem Probanden gebildeten Kategorien dahingehend geprüft, welche Merkmale die Funktionen einer Kategorie gemeinsam besitzen. Für jedes Merkmal wurde ein relativer Kennwert berechnet: Die Anzahl der Kategorien, in denen ein Merkmal verwendet wurde, wird für Merkmale, die bei allen Funktionen beurteilt werden können (z.B.: Rekursion ja/nein; Zahl der Abbruchbedingungen, Datentyp), an der Anzahl der gebildeten Kategorien relativiert. Für das Merkmal "Rekursionstyp" (Endrekursion, Teil-Rest-Rekursion, Simultanrekursion, Teil-Rest- und Endrekursion) wird zur Relativierung die Anzahl aller von einem Probanden gebildeten Kategorien verwendet, in denen nur rekursive Funktionen enthalten sind.

3 Ergebnisse

Bevor auf den Einfluß der Beispielbedingungen auf Programmierfertigkeit und Wissenserwerb eingegangen wird, geben wir einen kurzen Überblick über die Ausprägung der erhobenen Variablen in der Stichprobe. Mehr als zwei Drittel der 55 Probanden (69.01%)

verfügten bereits über Computererfahrung, die sich vor allem auf die Anwendung von Standardprogrammen (Textverarbeitung, Statistikpakete) bezieht. Nur zwei Probanden gaben an, rudimentäre Programmiererfahrung (in PASCAL bzw. BASIC) zu haben. Weniger als ein Drittel der Probanden (29.09%) hatte Mathematik in der gymnasialen Oberstufe als Schwerpunktfach belegt. Die letzte Mathematiknote ($AM = 2.65$, $sd = 1.12$), sowie die Selbsteinschätzung der Begabung ($AM = 2.82$, $sd = 0.82$; auf einer 5-stufigen Skala von 1=gar nicht begabt bis 5=sehr begabt) liegen im mittleren Bereich. Im Syntaxtest erzielten die Probanden im Schnitt 16.36 ($sd = 5.58$) von 28 Punkten. Damit verfügen die Probanden über ausreichende Syntaxkenntnisse zur Bewältigung der nachfolgenden rekursiven Aufgaben. Von den sechs Zielaufgaben, die nur mit Unterstützung der gemäß den Lernbedingungen vorgegebenen Beispielen bearbeitet wurden, lösten die Probanden fast 70 Prozent ($AM = 4.14$, $sd = 1.95$). Die im Abschlußtest gemessene Transferleistung liegt knapp unter der Hälfte der maximal erreichbaren 22 Rohwertpunkte ($AM = 10.64$, $sd = 4.15$).

3.1 Einfluß der Beispielbedingung auf Programmierfertigkeit und Wissenserwerb

Die Kontrollvariable Syntaxtest ist über die Lernbedingungen (Beispielähnlichkeit × Mapping) hinweg leider nicht vergleichbar. Sowohl der Haupteffekt Beispielähnlichkeit, als auch die Interaktion von Beispielähnlichkeit und Mapping werden bei einer Irrtumswahrscheinlichkeit von 25% signifikant ($n = 55$; Mapping: $F = 0.004$, $p = 0.95$; Beispielähnlichkeit: $F = 2.04$, $p = 0.11$; Mapping × Beispielähnlichkeit: $F = 2.13$, $p = 0.09$). Am deutlichsten unterscheiden sich die Lernbedingungen der Beispielähnlichkeiten II und III bezüglich der Mapping-Variation (siehe Abb. 2). Aus diesem Grund wird der Syntaxtest bei den folgenden statistischen Analysen als Kovariate mitberücksichtigt.

3.1.1 Programmierfertigkeit

Die Lernbedingungen beeinflussen die Anzahl gelöster Aufgaben signifikant ($n = 49$, 6 missing values; Mapping: $F = 5.98$, $p = 0.02$; Beispielähnlichkeit: $F = 4.00$, $p = 0.01$; Mapping x Beispielähnlichkeit: $F = 1.5$, $p = 0.22$; Kovariate Syntaxtest: $F = 21.48$, $p <$

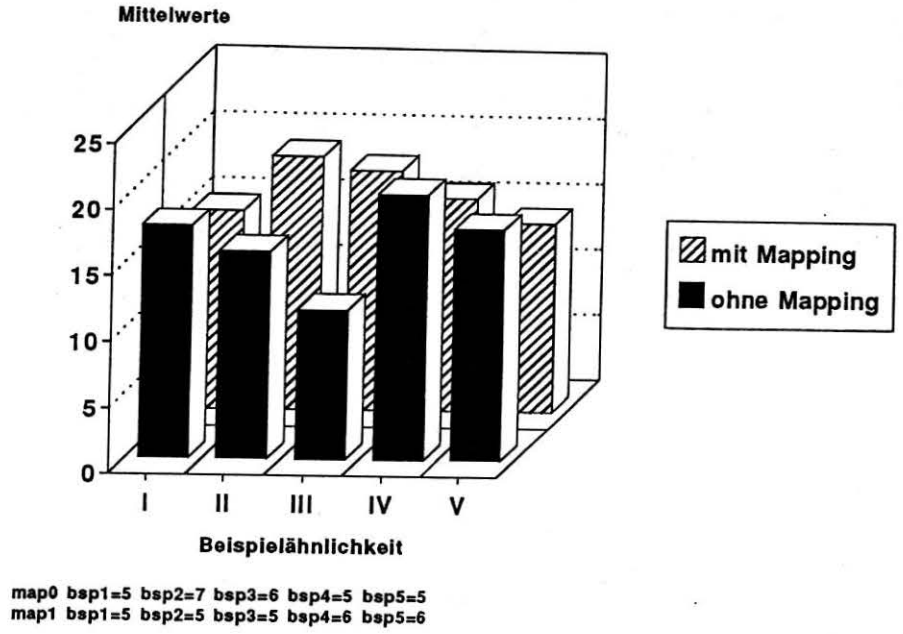

Abb. 2 Vergleichbarkeit der Syntaxkenntnisse über die Bedingungen Mapping × Beispielähnlichkeit

0.0001). Die Vorgabe der Ergebnisse des Mapping-Prozesses, sowie hohe Beispielähnlichkeit führen zu einer erhöhten Anzahl gelöster Aufgaben (siehe Abb. 3).

3.1.2 Wissenserwerb I: Lerntransfer

Um eine zuverlässige Skala zur Beschreibung des Lerntransfers zu erhalten, wurde der Abschlußtest raschskaliert. Nach Ausschluß eines Probanden der Beispielbedingung Ähnlichkeit II, ohne Mapping, konnte das Modell angepaßt werden ($n = 59$); Itemmodell: $\chi^2 = 402.63$, $df = 378$, $z = 0.90$; Personenmodell: $\chi^2 = 1042.96$, $df = 986$, $z = 1.27$). Die Werte des Personenmodells liefern den Kennwert für die Transferleistung der Probanden. Alle statistischen Analysen wurden mit den Raschskalenwerten berechnet. Zusätzlich zu den raschskalierten Werten des Gesamttests wurden auch die Rohwertsummen (vom Syntaxtest bereinigt) der vier Untertestgruppen des Abschlußtests betrachtet. Die Korrelationen der Untertests mit der Gesamtskala sind durchweg signifikant. Die Interkorrelationen der Untertests sind ebenfalls signifikant, weisen jedoch geringere Koeffizienten auf (siehe Tab. 3).

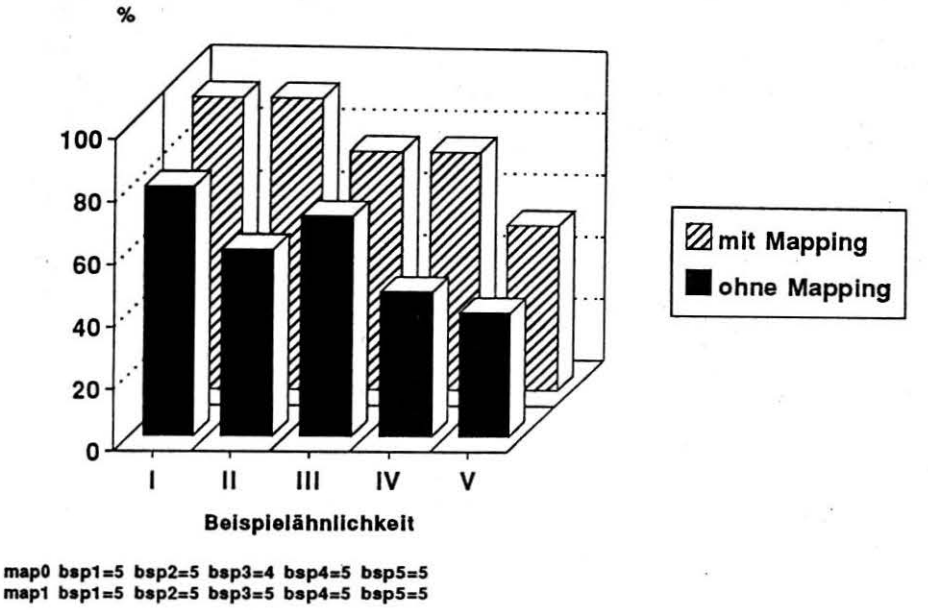

Abb. 3 Einfluß der Beispielbedingung auf die Anzahl gelöster Aufgaben

Die Lernbedingungen haben keinen bedeutsamen Einfluß auf den Lerntransfer ($n = 54$; Mapping: $F = 1.9$, $p = 0.18$; Beispielähnlichkeit: $F = 0.45$, $p = 0.77$; Mapping × Beispielähnlichkeit: $F = 0.35$, $p = 0.84$; Kovariate Syntaxtest: $F = 18.346$, $p < 0.0001$). Tendenziell zeigen jedoch Probanden, denen die Ergebnisse des Mapping-Prozesses vorgegeben wurden, eine bessere Transferleistung (siehe Abb. 4).

Dieser Eindruck wird durch signifikant bessere Leistungen dieser Probandengruppe bei der Beantwortung der Wissensitems (Rangvarianzanalyse: $n = 54$, $\chi^2 = 5.1$, $p = 0.024$) sowie der Lösung der Programmieraufgaben ($\chi^2 = 8.5$, $p = 0.004$) bestätigt (siehe Abb. 5 und Abb. 6).

3.1.3 Wissenserwerb II: Schemaerwerb

Die Distanz zur Expertenhierarchie ist im ersten Durchgang am höchsten ($n = 54$, $AM = 8.16$), im zweiten Sortierdurchgang am geringsten ($n = 53$, $AM = 7.66$) und steigt beim dritten Durchgang wieder an ($n = 52$, $AM = 8.04$). Die Distanzen zur Expertenhierarchie

TABELLE 3
Korrelation der Itemgruppen des Abschlußtests mit dem Gesamttestwert und Interkorrelationen

	Wissen	Semantik	E/A	Programm
Gesamttest	0.648	0.781	0.871	0.670
	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$
Wissen		0.316	0.341	0.326
		$p = 0.02$	$p = 0.01$	$p = 0.02$
Semantik			0.545	0.434
			$p < 0.0001$	$p = 0.001$
E/A				0.606
				$p < 0.0001$

Anmerkungen.
$n = 55$ bzw. bei Korrelation mit Gesamttest $n = 54$.
Gesamttest: Personenskala des Raschmodells.
Wissen, Semantik, E/A, Programm.: Rohwertsummen der Itemgruppen.

interkorrelieren über alle drei Sortierdurchgänge signifikant, wobei die jeweils benachbarten Durchgänge höher korrelieren (siehe Tab. 4).

Leider ergab sich für den ersten Sortierdurchgang, der vor der experimentellen Bedingungsvariation durchgeführt wurde, ein bedeutsamer Unterschied für die Beispielbedingungen ($n = 54$; Mapping: $F = 10.11$, $p = 0.003$; Beispielähnlichkeit: $F = 2.44$, $p = 0.06$; Mapping × Beispielähnlichkeit: $F = 1.87$, $p = 0.13$; Kovariate Syntaxtest: $F = 16.46$, $p < 0.0001$).

Wird für die beiden folgenden Sortierversuche zusätzlich zum Syntaxttest die Distanz zur Expertenhierarchie im ersten Durchgang als Kovariate verwendet, so ergeben sich keine signifikanten Unterschiede zwischen den Beispielbedingungen ($n = 53$; Durchgang 2: Mapping: $F = 2.23$, $p = 0.14$; Beispielähnlichkeit: $F = 0.31$, $p = 0.87$; Mapping × Beispielähnlichkeit: $F = 0.57$, $p = 0.69$; Kovariate Syntaxtest: $F = 7.69$, $p = 0.01$; Kovariate Distanz bei Durchgang 1: $F = 3.28$, $p = 0.08$; Durchgang 3: Mapping: $F = 0.98$, $p = 0.33$; Beispielähnlichkeit: $F = 0.47$, $p = 0.76$; Mapping × Beispielähnlichkeit: $F = 0.57$,

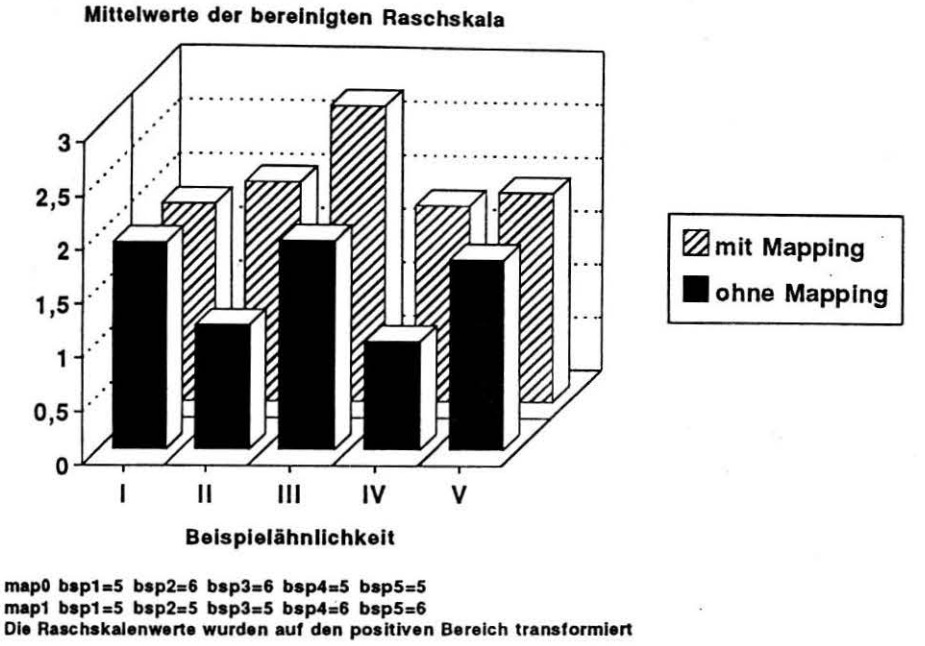

Abb. 4 Einfluß der Beispielbedingungen auf den Lerntransfer

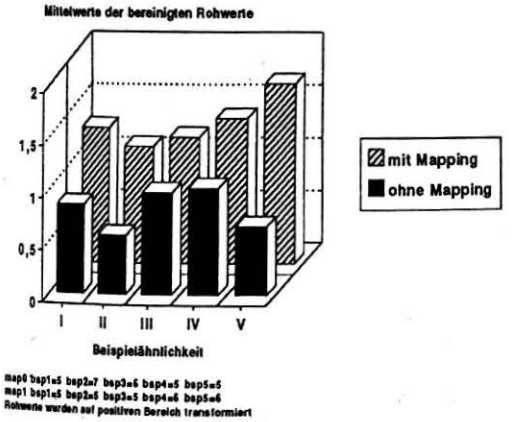

Abb. 5 Einfluß der Beispielbedingungen auf Wissensitems

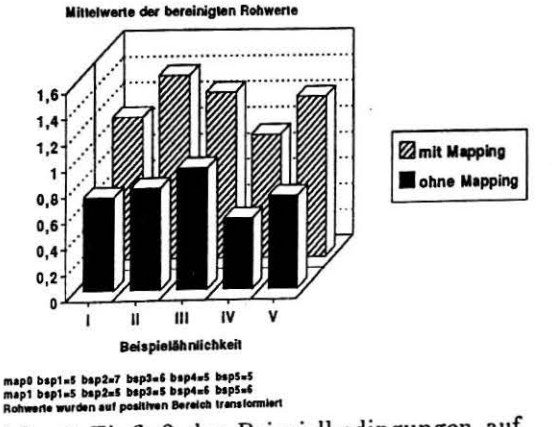

Abb. 6 Einfluß der Beispielbedingungen auf den Programmieritems

$p = 0.69$; Kovariate Syntaxtest: $F = 10.39$, $p = 0.003$; Kovariate Distanz bei Durchgang 1: $F = 0.11$, $p = 0.74$). Generell ist über alle drei Sortierdurchgänge festzustellen, daß Probanden ohne Mapping-Vorgabe tendenziell ähnlicher zur Expertenhierarchie sortieren. Abbildung 7 zeigt dies exemplarisch für Durchgang 2.

TABELLE 4
Interkorrelationen der Distanzen
zur Expertenhierarchie

	Durchg. 1	Durchg. 2
Durchg. 2	0.529 ($n = 53$) $p < 0.0001$	
Durchg. 3	0.282 ($n = 52$) $p = 0.021$	0.68 ($n = 52$) $p < 0.0001$

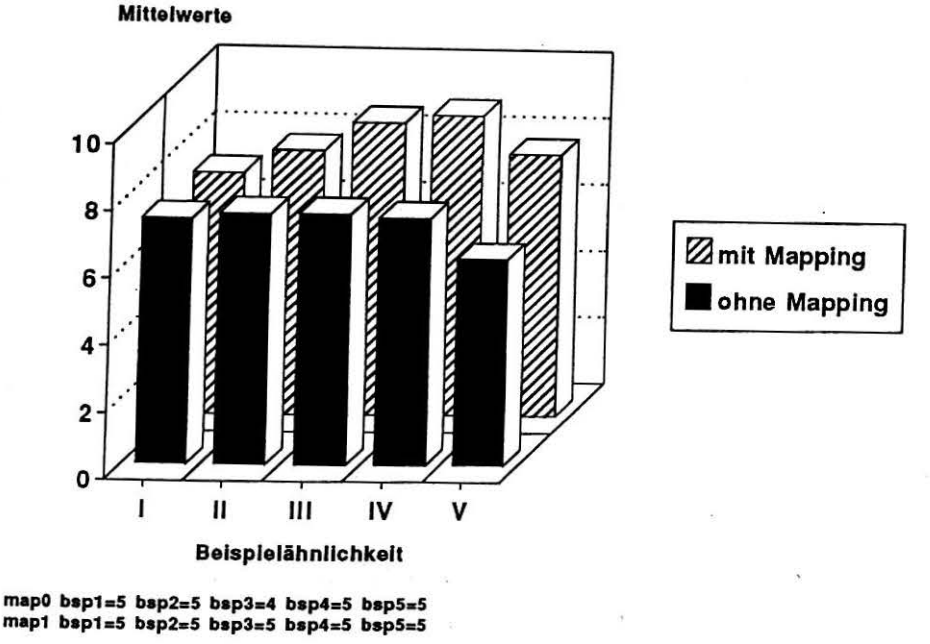

Abb. 7 Einfluß der Beispielbedingung auf die Distanz zur Expertenhierarchie

Die Probanden verwendeten insgesamt fünf Merkmale zur Sortierung der 28 Funktionen (siehe Tab. 5). Vier dieser Merkmale sind auch der Expertenhierarchie zugrundegelegt. Während wir jedoch annehmen, daß Experten rekursive Funktionen vor allem nach dem

Merkmal Rekursionstyp beurteilen, wird dieses Kriterium nur von einer geringen Probandenzahl verwendet. Die Probanden orientieren sich beim Vergleich der Funktionen eher an Oberflächenmerkmalen, wie Datentypen und Anzahl von Abbruchbedingungen. Bei einer nur viertägigen Lernzeit ist jedoch auch keine Expertise im Umgang mit rekursiven Funktionen zu erwarten. Daß Anfänger eher nach Oberflächenmerkmalen klassifizieren, ist ein gut belegter Befund (z.B. Adelson 1981; Novick 1988).

TABELLE 5

Verwendete Sortiermerkmale

	Rang	Durchg. 1	Durchg. 2	Durchg. 3
Rekursion ja/nein	3	Mod 1	1	1
		Md 0.66	0.75	0.55
		AM 0.62	0.66	0.59
Rekursionstyp	5	Mod 0	0	0
		Md 0	0	0
		AM 0.12	0.10	0.10
Anzahl der Abbruchbed.	2	Mod 1	1	1
		Md 0.69	0.75	0.70
		AM 0.65	0.70	0.63
Datentypen	1	Mod 1	1	1
		Md 1	0.89	0.89
		AM 0.78	0.75	0.71
boolsche Anweisung	4	Mod 1	1	1
		Md 0.63	0.67	0.57
		AM 0.59	0.62	0.55

Anmerkung.
Die Maße der zentralen Tendenz sind über die relativen Häufigkeiten der Verwendung der Merkmale berechnet.

Tendenziell verwendeten Probanden ohne Vorgabe des Mapping-Prozesses das Merkmal Rekursionstyp häufiger als Probanden mit Mapping-Vorgabe. Probanden dieser Gruppe sortierten auch ähnlicher zur Expertenhierarchie. Deutlich wird dieser Unterschied für Sortierdurchgang 2, hier verwendeten acht Probanden ohne Mapping-Vorgabe und zwei Probanden mit Mapping-Vorgabe das Merkmal Rekursionstyp (die Wahrscheinlichkeit, daß die Zufallsvariable Rekursionstyp in der Binomialverteilung für $p = 10/54$ einen Wert größer gleich 8 annimmt beträgt 0.02 und daß sie einen Wert kleiner gleich 2 annimmt beträgt 0.11). In Durchgang 3 verringert sich der Unterschied zu fünf Probanden ohne und vier Probanden mit Mapping-Vorgabe.

3.1.4 Zusammenhänge zwischen Programmierfertigkeit und Wissenserwerb

Während sich ein bedeutsamer Einfluß der Beispielbedingungen auf die Anzahl gelöster Aufgaben zeigt, kann ein solcher Einfluß für die Maße des Wissenserwerbs nur teilweise nachgewiesen werden. Im folgenden werden Zusammenhänge zwischen Aufgabenlösung und erworbenem Wissen unabhängig von den Beispielbedingungen betrachtet. Zwischen der Anzahl gelöster Aufgaben und dem Lerntransfer, sowie den Itemgruppen Semantik und Programmierung, bestehen signifikante Zusammenhänge. Die Distanz zur Expertenhierarchie im dritten Sortierdurchgang steht ebenfalls in signifikanten Zusammenhang mit der Aufgabenlösung (siehe Tab. 6).

Zwischen den beiden Maßen des Wissenserwerbs – Lerntransfer und Nähe zur Expertensortierung – bestehen dagegen keine bedeutsamen Zusammenhänge. Es zeigt sich jedoch, daß Probanden, die im zweiten Durchgang das Merkmal "Rekursionstyp" zur Sortierung der Funktionen verwendeten, signifikant höheren Lerntransfer aufwiesen, als Probanden, die dieses Merkmal *nicht* verwendeten (Rekursionstyp verwendet: $n = 12$, $AM = 0.81$ vs. nicht verwendet: $n = 41$, $AM = -0.59$; die Mittelwerte wurden über die Raschskala berechnet; Rangvarianzanalyse: $\chi^2 = 5.79$, $p = 0.016$).

4 Diskussion

Ausgehend von der Annahme, daß beim analogen Erwerb von Problemlösefertigkeiten abstrakte Schemata inferiert werden (Anderson & Thompson 1989; Novick & Holyoak 1991),

TABELLE 6

Zusammenhang zwischen Anzahl gelöster Aufgaben
und Maßen des Wissenserwerbs

	Abschlußtest	Wissen	Semantik	E/A	Progr.
Aufgabenlösung	0.408	0.267	0.444	0.280	0.319
	($n = 48$)	($n = 49$)	($n = 49$)	($n = 49$)	($n = 49$)
	$p = 0.004$	$p = 0.064$	$p = 0.001$	$p = 0.051$	$p = 0.026$
	Durchg. 1	Durchg. 2	Durchg. 3		
Aufgabenlösung	-0.041	-0.109	-0.331		
	($n = 48$)	($n = 48$)	($n = 48$)		
	$p = 0.779$	$p = 0.463$	$p = 0.021$		

wurde geprüft, inwieweit die gebildeten Schemastrukturen von der Ähnlichkeit der Beispiele zu den zu lösenden Aufgaben beeinflußt wurden. Dabei wurde der Schemaerwerb einmal indirekt über einen Test zum Lerntransfer und einmal explizit durch Sortieraufgaben erfaßt. Es zeigte sich, daß die Programmierfertigkeiten während der Lernphase durch hohe Ähnlichkeit der Beispiele zu den Aufgaben und durch Vorgabe der Ergebnisse des Mapping-Prozesses positiv beeinflußt werden. Ein Einfluß der Beispielähnlichkeit auf den resultierenden Wissenserwerb ließ sich jedoch nicht nachweisen. Dagegen beeinflußt die Vorgabe bzw. Nicht-Vorgabe der Ergebnisse des Mapping-Prozesses den resultierenden Wissenserwerb, jedoch in unterschiedlichen Richtungen: Während eine Vorgabe des Mappings den Lerntranfer, insbesondere die Lösung weiterer Programmieraufgaben, erleichtert, verwenden Probanden, die den Mapping-Prozeß selbst leisten mußten, eher das Merkmal des Rekursionstyps zur Klassifikation von Funktionen und sind damit auch näher an der Wissensstruktur eines Experten im Bereich rekursives Programmieren (vgl. Vorberg & Goebel 1991). Probanden, die das Merkmal "Rekursionstyp" zur Klassifikation von Funktionen verwenden, zeigen auch signifikant besseren Lerntransfer.

Der Befund, daß hohe Beispielähnlichkeiten die Lösungswahrscheinlichkeit analoger Zielprobleme erhöht, unterstützt die von Reed und Bolstadt (1991) berichteten Ergebnisse. Der fehlende Einfluß der Beispielähnlichkeit auf den resultierenden Wissenserwerb steht im Widerspruch zu den Annahmen von Pirolli und Anderson (1985). Leider ist uns keine andere Studie bekannt, die den Einfluß von Beispielähnlichkeit auf den Wissenserwerb untersucht, so daß eine Interpretation dieses Ergebnisses nicht durch andere Befunde gestützt werden kann. Eine mögliche Ursache für die fehlende Wirkung der Beispielähnlichkeit auf den Wissenserwerb könnte sein, daß die Beispiele nur passiv rezipiert wurden und die Probanden nicht dazu aufgefordert waren, die Beispielprobleme zunächst selbständig zu lösen (vgl. Needham & Begg 1991). In Untersuchungen, bei denen verschiedene Lehrmaterialien – häufig Beispiele und Erklärungen/Regeln – miteinander verglichen wurden (Cheng, Holyoak, Nisbett & Oliver 1986; Fong, Krantz & Nisbett 1986; Schmalhofer, Boschert & Kühn 1990), konnte jedoch nicht nachgewiesen werden, daß verschiedene Materialien zu verschiedenen Ergebnissen beim Wissenserwerb führen. Während Cheng et al. und Fong et al. diese Ergebnisse verwenden, um zu belegen, daß Beispiele ein geeignetes Lehrmaterial darstellen, postulieren Schmalhofer et al., daß die von ihnen verwendeten Materialien informationsäquivalent sind und zum Aufbau gleicher allgemeiner Schemata führen. Andererseits kann man annehmen, daß die Lösung der Aufgaben selbst den Wissenserwerbsprozeß dominiert, und somit der Einfluß eines als Hilfestellung gegebenen Materials von den Auswirkungen des learning by doing (Anderson, Conrad & Corbett 1989) überlagert wird. Um diese Fragen zu klären, sind weitere Untersuchungen, bei denen die Beispielähnlichkeit variiert wird, notwendig.

Die unterschiedlichen Auswirkungen des Faktors Mapping auf die beiden Maße zur Erfassung des Wissenserwerbs scheinen im Widerspruch zu der Annahme zu stehen, daß beim analogen Lernen eine einheitliche Wissensstruktur aufgebaut wird. Geht man jedoch im Sinne eines Rahmenmodells von Holland, Holyoak, Nisbett & Thagard (1987, Kap. 2) davon aus, daß beim induktiven Wissenserwerb sowohl Regeln zur Problemklassifikation als auch Regeln zur Problemlösung erworben werden, ist das Ergebnis interpretierbar. Holland et al. postulieren, daß Lernprozesse solange stattfinden, bis Problemklassifikation und Problemlösung hinreichend gut gelingen und aneinander angepaßt sind. Bei einem so kurzen

Lernzeitraum, wie in der berichteten Untersuchung, kann nicht davon ausgegangen werden, daß der Lernprozeß für den Erwerb rekursiver Progammierfertigkeiten bereits abgeschlossen wäre. Insofern scheint eine Vorgabe des Mapping-Prozesses eine schnellere Inferenz der Anwendungsbedingungen rekursiver Programmiertechniken zu ermöglichen, der Aufbau einer sinnvollen Merkmalsmenge zur Klassifikation rekursiver Probleme scheint dagegen erstens ein längerfristiger Prozeß zu sein und zweitens auch eher das Resultat selbständiger Vergleichsprozesse.

Das Ergebnis, daß Probanden, die das für Experten relevante Merkmal "Rekursionstyp" zur Klassifikation von Funktionen verwenden, signifikant höhere Transferleistungen zeigen, belegt die von Vorberg und Goebel (1991) postulierte These, wonach dieses Merkmal in der aufgebauten Schemahierarchie notwendig und hinreichend zur Lösung rekursiver Programmieraufgaben ist.

Das von den Probanden in wenigen Tagen erworbene Wissen über die Programmierung rekursiver Funktionen ist trotz der nur eingeschränkten Unterstützung des Lernprozesses erstaunlich hoch. Viele Untersuchungen zum Erwerb von Programmierwissen verzichten entweder ganz darauf, Programmieranfänger mit tatsächlichen Programmieraufgaben zu konfrontieren (z.B. Kahney 1989; Kurland & Pea 1983) oder berichten über so geringe Erfolgsquoten, daß statistische Analysen nicht möglich sind (z.B. Schmalhofer, Boschert & Kühn 1990). Damit scheint sowohl die verwendete Programmiersprache (Schmid 1994), als auch das Konzept des Lernens aus Beispielen geeignet für die Vermittlung einer so komplexen Problemlösetechnik wie das Programmieren rekursiver Funktionen.

Literatur

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition, 9* (4), 422-433.

Anderson, J. R., Conrad, F. G. & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13*, 467-505.

Anderson, J.R. & Thompson, R. (1989). Use of analogy in a production system architecture. In S. Vosniadou and A. Ortony (Ed.), *Similarity and Analogical Reasoning* (pp. 267-297). Cambridge: Cambridge University Press.

Cheng, P.W., Holyoak, K.J., Nisbett, R.E. & Oliver, L.M. (1986). Pragmatic versus syntactic approaches to training deductiv reasoning. *Cognitive Psychology, 18*, 293-328.

Fong, G.T., Krantz, D.H. & Nisbett, R.E. (1986). The effects of statistical training on thinking about everyday problems. *Cognitive Psychology, 18*, 253-292.

Kahney, H. (1989). What do novice programmers know about recursion? In E. Soloway a. J.C. Spohrer (Ed.), *Studying the Novice Programmer* (pp. 209-228) Lawrence Erlbaum.

Kurland, D.M., & Pea, R.D. (1983). Mental models of recursive Logo programs. Proceedings of the Fifth Annual Meeting of the Cognitive Science Society, 1-5.

Needham, D.R. & Begg, I.M. (1991). Problem-oriented training promotes spontaneous analogical transfer: Memory-oriented training promotes memory for training. *Memory and Cognition, 19*, (6), 543-557.

Novick, L.R. (1988). Analogical transfer, problem similarity, and expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 14*, 510-520.

Novick, L.R. & Holyoak, K.J. (1991). Mathematical problem solving by analogy. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17* (3), 398-415.

Reed, S.K. & Bolstad, C.A. (1991). Use of examples and procedures in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17* (4), 753-766.

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

Schmid, U. (1994). Programmierenlernen: Der Einfluß von Beispielfunktionen und erklärenden Texten auf den Erwerb rekursiver Programmiertechniken. *Kognitionswissenschaft*, im Druck.

Anhang A: Rekursion, Syntax der funktionalen Sprache und Lernumgebung

Die in der Untersuchung verwendete Lernumgebung LEAR wurde eigens für die Vermittlung rekursiver Programmiertechniken im Rahmen funktionaler Programmierung entwickelt. Kern der Lernumgebung ist ein Interpreter für eine einfache funktionale Sprache. Diese Programmiersprache enthält wesentliche Charakteristika funktionaler Sprachen (LISP, LOGO, ML). Auf eine formale Darstellung der Sprache wird hier verzichtet. Stattdessen werden im folgenden am Beispiel der Fakultätsfunktion sowohl das Konzept der Rekursion als auch die wesentlichen Merkmale der Sprache erläutert.

Die Fakultät einer Zahl n (geschrieben $n!$) ist das Produkt der Zahl n mit all ihren Vorgängern (bis 1), wobei $0! = 1$ gesetzt wird. Diese Vorschrift kann funktional wie folgt angegeben werden:

$$n! = \begin{cases} 1 & \text{wenn } n = 0 \\ n \cdot (n-1)! & \text{sonst.} \end{cases}$$

Für $n = 4$ errechnet sich $4! = 4 \cdot 3 \cdot 2 \cdot 1$ nach dieser Vorschrift folgendermaßen:

$4! = 4 \cdot 3!$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = 4 \cdot 6 = 24$

$3! = 3 \cdot 2!$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = 3 \cdot 2 = 6$

$2! = 2 \cdot 1!$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = 2 \cdot 1 = 2$

$1! = 1 \cdot 0!$ $\qquad\qquad\qquad\qquad\qquad = 1 \cdot 1 = 1$

rekursiver Abstieg ↘ $\qquad\qquad 0! = 1 \qquad\qquad$ ↗ rekursiver Aufstieg

Die Funktion ! wurde auf der zu definierenden Seite ($n!$) und auf der definierenden Seite ($n \cdot (n-1)!$) verwendet. Dies ist das Grundprinzip der Rekursion. Wichtig bei rekursiven Funktionen ist, daß es mindestens einen Fall gibt, bei dem das Ergebnis der Funktion direkt angebbar ist (hier $n = 0$). Nur dann kann der rekursive Abstieg beendet werden. Anderenfalls würde die Berechnungsvorschrift "endlos" angewendet (Nichttermination). Die mathematisch angegebene Fakultätsfunktion kann folgendermaßen in der Syntax von LEAR programmiert werden:

```
FUN faku (n:NAT):NAT
IF EQUAL (n,0)
THEN 1
ELSE MULT(n, faku(MINUS(n,1)))
FIN
```

In der ersten Zeile der Funktion (Funktionskopf) wird nach dem Schlüsselwort FUN der Name der Funktion, die Parameter und der Ergebnistyp angegeben. Die Funktion mit Namen faku besitzt einen Parameter, der mit n bezeichnet wird und der vom Datentyp NAT (natürliche Zahl) ist. Das Ergebnis der Funktion ist ebenfalls eine natürliche Zahl. Die folgenden drei Funktionszeilen (Funktionsrumpf) sind eine bedingte Anweisung (IF-THEN-ELSE). Es wird geprüft, ob eine Zahl n gleich 0 ist, ist dies der Fall, wird die Zahl 1 geliefert, sonst wird n mit dem Ergebnis der rekursiven Aufrufs faku(MINUS(n,1)) multipliziert. Die Funktion wird durch das Schlüsselwort FIN beendet. Neben dem Datentyp NAT verfügt LEAR über die Datentypen "Listen von Zahlen" (NATLIST) und Wahrheitswerte (BOOL). Als arithmetische Operationen stehen PLUS, MINUS und MULT zur Verfügung. Bei Listen kann das erste Element einer Liste (HEAD) ausgegeben werden, die Liste um das erste Element reduziert werden (TAIL), sowie ein neues Element in eine Liste eingefügt werden (CONS). Neben der Prüfung auf Gleichheit (EQUAL), kann geprüft werden, ob eine Zahl größer als eine andere ist (GREATER) und ob eine Liste leer ist (EMPTY). Alle Operationen werden in sogenannter Präfix-Notation verwendet, das heißt, daß zuerst der Name der Operation (z.B. MINUS) und dann in Klammern die Argumente angegeben werden.

Die Lernumgebung für diese einfache Sprache besteht aus folgenden Fenstern und Funktionen:

- Aufgabenfenster: Vorgabe von Aufgabenstellungen und Vorschlägen für Werten zum Testen der Funktionen
- Editorfenster: einfacher Editor zum Erstellen der Funktionen
- Interpreterfenster: Kommandozeilen zur Ausführung von Funktionen
- Meldungsfenster: Meldung syntaktischer und semantischer Fehler in den Funktionen (zusätzlich durch Curserposition im Editorfenster lokalisiert)
- Lernhilfenfenster: Vorgabe eines Beispiels pro Aufgabe.

Nach Vorgabe einer Aufgabe befindet man sich im Editorfenster. Durch Tastendruck kann von dort eine Prüfung der erstellten Funktion auf Fehler angefordert werden. Bei syntaktisch und semantisch korrekten Lösungen gelangt man automatisch ins Interpreterfenster, ansonsten wird der Fehler gemeldet. Im Interpreterfenster kann die erstellte Funktion mit den vorgeschlagenen oder selbstgewählten Werten ausgeführt werden. Im Experiment war die Lösungszeit pro Aufgabe auf 25 Minuten begrenzt.

Anhang B: Konstruktion der Beispiele

Exemplarisch werden die Beispiele zur ersten Aufgabe der 4. Lektion angegeben.

Aufgabe: Schreibe eine rekursive Funktion addz, die den Summenwert einer Zahl x mit all ihren Vorgängern bis 0 berechnet. Beispiel: $x = 5$ ergibt $5 + 4 + 3 + 2 + 1 + 0 = 15$	FUN addz (x:NAT):NAT IF EQUAL(x,0) THEN 0 ELSE PLUS(x,addz(MINUS(x,1))) FIN
Beispiel I: Änderung von Konstanten oder vordefinierten Operationen Die Funktion bsp berechnet die Summe einer Zahl mit all ihren Vorgängern ohne 2 und 1. Beispiel: $x = 5$ ergibt $5 + 4 + 3 = 12$	**FUN bsp (x:NAT):NAT** **IF EQUAL(x,2)** **THEN 0** **ELSE PLUS(x,bsp(MINUS(x,1)))** **FIN**
Beispiel II: Erweiterung von Abbruch- bedingung oder direktem Fall Die Funktion bsp berechnet die Summe einer Zahl mit all ihren Vorgängern ohne 1. Beispiel: $x = 5$ ergibt $5 + 4 + 3 + 2 = 14$	**FUN bsp (x:NAT):NAT** ***IF EQUAL(MINUS(x,1),0)*** **THEN 0** **ELSE PLUS(x,bsp(MINUS(x,1)))** **FIN**
Beispiel III: Erweiterung des rekursiven Aufrufs Die Funktion bsp berechnet die Summe einer quadrierten Zahl mit all ihren quadrierten Vorgängern. Beispiel: $x = 5$ ergibt $5 \cdot 5 + 4 \cdot 4 + 3 \cdot 3 + 2 \cdot 2 + 1 \cdot 1 + 0$ $= 25 + 16 + 9 + 4 + 1 + 0 = 55$	**FUN bsp (x:NAT):NAT** **IF EQUAL(x,0)** **THEN 0** **ELSE PLUS(MULT(x,x),bsp(MINUS(x,1)))** **FIN**
Beispiel IV: Erweiterung von Abbruchbedingung oder direk- tem Fall und zusätzlich des rekursiven Aufrufs Die Funktion bsp berechnet die Summe einer quadrierten Zahl mit all ihren quadrierten Vorgängern ohne 1. Beispiel: $x = 5$ ergibt $5 \cdot 5 + 4 \cdot 4 + 3 \cdot 3 + 2 \cdot 2$ $= 25 + 16 + 9 + 4 = 54$	**FUN bsp (x:NAT):NAT** **IF EQUAL(MINUS(x,1),0)** **THEN 0** **ELSE PLUS(MULT(x,x),bsp(MINUS(x,1)))** **FIN**

Beispiel V: Erweiterung von Abbruchbedingung, direktem Fall und rekursivem Aufruf

Die Funktion bsp berechnet die Summe einer quadrierten Zahl mit all ihren quadrierten Vorgängern ohne 1.

Beispiel: $x = 5$ ergibt
$5 \cdot 5 + 4 \cdot 4 + 3 \cdot 3 + 2 \cdot 2$
$= 25 + 16 + 9 + 4 = 54$

```
FUN bsp (x:NAT):NAT
IF EQUAL(MINUS(x,1),0)
THEN MINUS(x,1)
ELSE PLUS(MULT(x,x), bsp(MINUS(x,1)))
FIN
```

Anhang C: Rekursive Aufgaben und Lösungen

Lektion 3: Endrekursive Funktionen

Schreibe eine rekursive Funktion dubx,
die eine Zahl z x-mal verdoppelt.
erster Parameter:
dubx(plus(z,z),minus(x,1))
zu verdoppelnde Zahl z,
zweiter Parameter:
Anzahl der Verdoppelungen x.
Beispiel: $z = 3$, $x = 2$
dubx(3,2) = dubx(6,1) = dubx(12,0)= 12

```
FUN dubx(z:NAT,x:NAT):NAT
IF EQUAL(x,0)
THEN z
ELSE dubx(PLUS(z,z),MINUS(x,1))
FIN
```

Schreibe eine rekursive Funktion last, die
das letzte Element einer Liste l, die
mindestens ein Element enthält, ausgibt.

```
FUN last (l:natlist):nat
IF empty (tail(l))
THEN head (l)
ELSE last (tail (l))
FIN
```

Schreibe eine rekursive Funktion member, die
1 liefert, wenn Element n in Liste l enthalten ist und 0, wenn n nicht in l ist.
erster Parameter: Zahl n,
zweiter Parameter: Liste l.

```
FUN member(n:NAT,l:NATLIST):NAT
IF empty (l)
THEN 0
ELSE IF equal (head(l),n)
THEN 1
ELSE member (n,tail(l))
FIN
```

Lektion 4: Teil-Rest-RekursiveFunktionen

Schreibe eine rekursive Funktion addz, die
den Summenwert einer Zahl x mit all ihren
Vorgängern bis 0 berechnet.
Beispiel: $x = 5$ ergibt
5 + 4 + 3 + 2 + 1 + 0 = 15

```
FUN addz (x:nat): nat
IF equal (x,0)
THEN 0
ELSE plus(x,addz(minus(x,1)))
FIN
```

Schreibe eine rekursive Funktion genlist, die eine
Liste absteigender Zahlen von n bis 1 ausgibt.
Beispiel: $n = 5$ ergibt
(5,4,3,2,1,nil)

```
FUN genlist(n:nat):natlist
IF equal (x,0)
THEN NIL
ELSE cons(n,genlist(minus(n,1)))
FIN
```

Schreibe eine rekursive Funktion suml, die eine Liste ausgibt, die die Summe zwischen aufeinanderfolgenden Listenelementen der Liste l ausgibt.
Beispiel: $l = (6,4,1)$ ergibt
(2,3,nil)

```
FUN suml(l:natlist): natlist
IF empty (tail(l))
THEN nil
ELSE cons (minus(head(l),head(tail(l))),
   suml(tail(l)))
FIN
```