DIE ENTWICKLUNG ZUM PROGRAMMIEREXPERTEN DURCH DAS PROBLEMLÖSEN MIT AUTOMATEN

CLAUS MÖBUS
Universität Oldenburg

ZUSAMMENFASSUNG

Wir wollen hier einige Ideen vorstellen, die uns seit einiger Zeit in Oldenburg beschäftigen und die wir im Rahmen eines DFG-Projektes in naher Zukunft konkreter erforschen wollen. Das Forschungsvorhaben reiht sich in die Projekte ein, die unter dem Stichwort "intelligente Tutorsysteme" den Computer zu einem neuen Medium der Wissensvermittlung im explorativen Sinne machen wollen (vgl. a. Abelson & diSessa, 1981).
Dabei sollen sich "intelligente" Tutorsysteme von ihren "nichtintelligenten" Geschwistern des computerunterstützten Unterrichts der 60er und 70er Jahre durch adaptive Wissens- und Fehlererklärungskomponenten unterscheiden. Dadurch soll die Effizienz menschlicher Tutoren im Einzelunterricht erreicht, Über- oder Unterforderung des Lernenden vermieden und dem Lernenden Rückmeldung über Kenntnisstand und Fehlkonzepte gegeben werden.
Der von uns dafür ausgewählte Diskursbereich "Problemlösen mit Automaten" ist klar definiert und abgrenzbar. Fehler durch den Lernenden lassen sich als solche einfach und eindeutig diagnostizieren. Kompetenz und fehlerfreies Wissen dagegen kann nur durch intensive Übung erreicht werden. Damit eignet sich der Stoff besonders gut, über einen Computertutor vermittelt zu werden.

1. EINLEITUNG

Der Stand der Forschung zum Thema "Programmierwissen" kann am besten mit einem Zitat von Barstow (1979) umrissen werden:
"(Menschliche) Programmierer wissen eine Menge über das Programmieren ... (Ein) Großteil dieses Wissens kann durch Fakten und Regeln so präzise formalisiert werden, daß es effizient von einer Maschine für die Ausführung von Programmieraufgaben verwendet werden kann. Bedauerlicherweise sind (aber) die meisten derzeit verfügbaren Quellen für Programmierwissen (z.B. Bücher und Artikel) nicht präzise genug, um dieses Kriterium zu erfüllen. Der Grund dafür ist einfach: diese Bücher und Artikel hatten den menschlichen Benutzer (als Adressaten) intendiert. (Für diesen) waren informelle Beschreibungen hinreichend, konnten Details weggelassen werden, konnten Annahmen vage formuliert bleiben. Menschliche Leser können mit einem informellen Stil fertig werden, die Details, wenn nötig, inferieren; normalerweise teilen sie (mit dem Autor) den gleichen Hintergrund an Annahmen. Um die notwendige Präzision zu erreichen, müssen Details nachgetragen und implizite Annahmen explizit gemacht werden."

Obwohl Barstow keine kognitionspsychologische Grundlagenforschung im Sinne hat, blieben ihm doch Schwachstellen im Forschungsbereich "Programmierwissen" nicht verborgen. Trotz der Fülle der Arbeiten (das Sammelreferat von Pea & Kurland (1984)

nennt allein 400 angelsächsische Arbeiten neueren Datums) bleibt die Berechtigung der Barstowschen Einschätzung nach wie vor bestehen.
Zum einen liegt das an der ergonomischen Ausrichtung vieler Autoren, die breitere Problemgebiete untersuchen wollen (Stichwort z.B.: "Humanisierung der Arbeitswelt"), zum anderen daran, daß die Autoren keine Prozeß- und Wissensmodellierung vornehmen. Dadurch bleibt selbst ihnen meist die Oberflächlichkeit ihres Ansatzes verborgen.

Als beispielhaft dagegen sind besonders die Arbeiten von Brooks (1977), Green (1980, 1983) und Anderson, Farrell und Sauers (1984) zu nennen. Jedoch nehmen nur Anderson et al. die Modellierung des Wissens und Wissenserwerbs ernst. Sie versuchen mit dem Modellansatz der Produktionssysteme (eine spezielle Sorte von MARKOValgorithmen), menschliche Planung beim Abfassen von LISP-Programmen zu simulieren.
Wir glauben, eine Reihe von grundlagenorientierten psychologischen und didaktischen Gründen anführen zu können, warum man solche Untersuchungen nicht innerhalb einer bestimmten Programmiersprache (LISP, LOGO, PASCAL, C, ALGOL, MODULA, BASIC, FORTRAN, PL/I, ADA, SIMULA, ELAN, PROLOG, PLANNER, ...) sondern in einem syntaktisch einfacheren, für die Programmiersprachen aber grundlegenden Bereich: "Problemlösen mit Automaten" durchführen sollte.

2. Programmieren von Computern und abstrakten Maschinen

Kaum ein anderes Fach hat in den letzten Jahren einen derartigen Boom erlebt wie die Informatik. Neben einem personellen Zuwachs an Studenten, Lehrern, Forschern und Anwendern war auch ein Zuwachs an Wissen in den physikalischen Grundlagen, der Theorie und den Anwendungen zu beobachten. Dabei nimmt das Programmieren von real existierenden Rechenanlagen oder Symbolprozessoren einen breiten,aber nicht alles umfassenden Raum ein. Neben sachlichen Gründen, die das Programmieren und das Wissen darüber wichtig werden lassen (z.B. die Algorithmisierung von Problemlösungen), gibt es auch andere, weniger positive Ursachen, die dem Programmieren stärkere Publizität verleihen. Diese ist durch die babylonische Sprachverwirrung im Bereich der Programmiersprachen bedingt.
So werden in der Schule verschiedene BASIC-Dialekte, PASCAL, ELAN, COMAL und LOGO "gesprochen". Im Hochschulbereich gesellen sich dann FORTRAN, ALGOL-68, PL/I, LISP und PROLOG hinzu. In Forschungsinstitutionen werden dazu noch laufend mit erheblichem finanziellen Aufwand neue Sprachen in der Hoffnung auf eine "Marktlücke" entwickelt.
Für den Bildungsbereich sind hier in der Reihenfolge ihrer Verfügbarkeit SMALLTALK-80 von XEROX, CIP von F.L. Bauer (Sonderforschungsbereich 49, TU München) und BOXER von diSessa (MIT, Cambridge USA) zu nennen. Im Gegensatz zu den beiden letztgenannten kann man zwar SMALLTALK auf kommerziellen Maschinen erwerben (Hoffmann, 1985), jedoch sind Anwendungen im Bildungswesen bisher nicht bekannt.
Während SMALLTALK wohl eher nichtnumerische Anwendungen im Auge hat, wurde an der TU München unter der Leitung von F.L. Bauer eine Breitbandsprache entwickelt. Das sogenannte CIP-Projekt (Computer-aided, Intuition-guided Programming) ist im DFG-geförderten Sonderforschungsbereich "Programmiertechnik" beheimatet und hatte von Anfang an didaktische und wissenspsychologische Ziele (Möller, 1984). Die Forschergruppe

um Bauer entwickelte dabei eine hypothetische "Gedankenmaschine", die auch als "Formularmaschine" oder *calculation sheet machine* bezeichnet wird. Diese Maschine leitet sich aus Konzeptionen der Schuldidaktik her ("Rechenbäume") und soll nach dem Willen ihrer Väter das Verständnis real existierender und zukünftiger Rechnerarchitekturen fördern und das Erlernen alter wie neuer Programmiersprachen erleichtern. Damit erfüllt die "Formularmaschine" genau die Kriterien, die die Kognitionspsychologie an mentale oder Surrogatmodelle stellt (Young, 1981; Gentner & Stevens, 1982).

Am weitesten in den Bildungsbereich will diSessa mit seiner interaktiven Umgebung BOXER stoßen. Der Computer soll nicht nur ein Medium sein, das Lernen des Programmierens zu fördern, sondern generell ein Instrument explorativen Lernens in weitgefächerten Wissensgebieten. BOXER versteht sich als Fortführung von LOGO. Einerseits bestand hier der Wunsch, auf der Lehrerseite eine "erwachsenere" Sprache als LOGO anbieten zu können. Zum anderen verläßt BOXER die noch in LOGO verwendete und von LISP übernommene eindimensionale Listenstruktur zugunsten einer Zweidimensionalität der Programmiersprache. BOXEN (d.h. Kästen) sind als zweidimensionale Erweiterung der eindimensionalen Listen anzusehen. Ebenso wie LISP besitzt die explorative Umgebung BOXER eine Syntax, die ihrer eigenen Datenstruktur entspricht, was leichte Erlernbarkeit garantiert. Entwicklungsarbeiten laufen z.Zt. auf vollen Touren (nämlich auf dem "Flaggschiff" der KI-Forschung, der LISP-Maschine von Symbolics).

Definiert man "Programmieren" als die Niederschrift der Kontrollstrukturen und des Datenflusses in einem Computer, kann man die semantische Komponente des Programmierwissens ganz generell als Wissen über den Ablauf von Kontrolle (Kontrollwissen) und des Datenflusses (Datenflußwissen) bezeichnen. Für ein konkretes Problem muß dieses semantische Wissen strukturiert, reorganisiert und zu einem Gedankenalgorithmus in umgangssprachlicher oder mathematischer Form kombiniert werden. Wir befinden uns in der Phase der prealgorithmischen Problemspezifikation (Dosch, 1983).
Die Kommunikation mit dem Computer kann aber zur Zeit noch nicht in natürlicher Sprache erfolgen (s. aber Miller, 1981). Daher muß die vorcomputersprachliche Form des Algorithmus in einer Programmiersprache eingebettet werden. Dazu muß der Programmierer Wissen über Syntax und Semantik der gerade ausgewählten Programmiersprache besitzen. Dosch (1983) unterscheidet hier zwischen der Designphase des Algorithmus mit applikativen bzw. funktionalen Sprachkonstrukten (wie Funktionsanwendung, Funktionskomposition, funktionaler Abstraktion, bedingten Aufrufen und Rekursion als alleiniger Kontrollstruktur) und der Implementationsphase des Algorithmus auf einer konventionellen von-Neumann-Maschine mit prozeduralen bzw. maschinenorientierten Sprachkonstrukten. Zu den prozeduralen Termen werden Anweisungen, Variablen, bedingte Zuweisungen, Prozeduren und Schleifen gezählt. Als maschinennahe Konstruktionen gelten Sprünge, Marken und Zeiger.
Zwar wird in den curricularen Empfehlungen sowie Studien- und Prüfungsordnungen versucht, die Ausbildung von funktionalem, prozeduralem, objektorientiertem, deklarativem oder maschinennahem Programmierwissen und Programmiersprachenwissen (BASIC, C, PASCAL etc.) zu orthogonalisieren, jedoch gelingt dieses faktisch nie. Lehrer haben zum Beispiel oft nur Programmiersprachenwissen aber kein Programmiermetasprachenwissen. Sie können nur innerhalb einer Sprache unterrichten. Oft läßt die technische

Ausstattung auch nur eine Sprache (meist BASIC) als Programmiermedium zu.
Das Resultat ist die frühzeitige Verquickung von metasprachlichem Programmierwissen mit Programmiersprachenwissen. Das führt dazu, daß zum Beispiel BASIC und FORTRAN-Programmierer rekursive Problemlösungen und dynamische Datenstrukturen so gut wie nie kennenlernen und daß PASCAL-Programmierer gezwungen sind, gleichzeitig maschinennah und maschinenfern zu programmieren, wenn sie mit dynamischen Datenstrukturen (d.h. *pointern*) arbeiten. Beide Schwierigkeiten lernt z.B. ein LISP-Programmierer nie kennen. Ebenso wird durch die Verwendung des prozeduralen Programmierstils in den klassischen Sprachen (ALGOL, FORTRAN, PASCAL etc.) ein maschinennahes Registermaschinenmodell als mentales Modell impliziert, ohne daß sich der Sprachverwender dessen bewußt wird. Dadurch wird aber das Erlernen anderer Sprachen eher behindert.

Ein derartiges - implizit erworbenes - mentales Modell als einziges Surrogat zu "besitzen", erscheint im Lichte neuerer Konzeptionen (funktionaler Ansatz, Datenflußsprachen) immer fragwürdiger (Davis & Keller, 1982; Gurd, Kirkham & Watson, 1985; Paseman, 1985; Pountain, 1985).
Üblicherweise kann der Lernende nur durch den Erwerb einer weiteren, der ersten gegenüber unähnlichen Sprache dazu gebracht werden, das metasprachliche Programmierwissen durch Abstraktion von den in konkreten Programmiersprachen abgefaßten Programmen zu entwickeln.
Erst die Abkopplung von semantischem und syntaktischem Wissen ermöglicht nach unseren Hypothesen die Entwicklung von Expertenwissen. Unter Expertenwissen zählen wir Invarianz- und Transformationswissen. Dieses hat Ähnlichkeit mit den *higher-order-rules* nach Scandura (1977). Ein Experte sollte in der Lage sein, aufgrund seines metasprachlichen Wissens, das ihn zu diesen Invarianz- und Transformationsleistungen befähigt, schneller neue Anforderungen zu bestehen. Er sollte insbesondere in der Lage sein, anders als ein Novize und auch anders als ein Spezialist (der z.B. nur eine Programmiersprache gut kennt), schnell neue Sprachen zu lernen, wenn die Bedürfnisse es erfordern, und auch leicht Algorithmen von funktional rekursiver Form in die prozedural iterative zu übersetzen.
Da die notwendige Abstraktion durch das Erlernen zweier (z.B. PASCAL*, LISP**) oder dreier unähnlicher (z.B. PASCAL, LISP, FORTRAN) Sprachen (vgl. Abbildung 1) ein aufwendiges mehrjähriges Unternehmen ist, stellt sich die Frage, ob es eine Möglichkeit gibt, das metasprachliche Expertenwissen direkt zu lehren, um danach Programmiersprachenwissen "vertikal" in das so entstandene Wissensnetz einzuhängen.
Idealerweise sollte das Lehrmaterial repräsentationsnah bzw. repräsentationsunterstützend sein (so hat z.B. Anderson zum Studium des FAN-Effektes den Aufbau semantischer Netze durch das Auswendiglernen kurzer propositionaler Sätze gefördert. Diese kurzen Propositionen in Aktivform werden von Kognitionspsychologen als repräsentationsnäher angesehen als komplizierte Passivkonstruktionen (Anderson, 1980, S. 176 ff).

*PASCAL	**LISP	*PASCAL	**LISP
meistens kompiliert	meistens interpretiert	reservierte Wörter	keine reservierten Wörter
starke Typisierung	typfrei	komplizierte Wörter	einfache Syntax
syntaktische Verschiedenheit von Programmen und Daten	syntaktische Einheit von Programmen und Daten	Zwang zu maschinennaher Programmierung bei Verwendung von Zeigern	verdeckte Zeigerprogrammierung
keine automatische Verwaltung des Speichers bei dynamischen Speicher- bzw. Datenstrukturen	automatische Verwaltung (garbage collection)		

Abb. 1: Gegenüberstellung der differenzierenden Eigenschaften zweier unähnlicher Programmiersprachen.

Welches Wissen sollte als metasprachliches Expertenwissen durch das Lehrmaterial aufgebaut werden? Hier sollte uns der Buchtitel eines Buches von Wirth (1976) Hinweise geben:

ALGORITHMS + DATA STRUCTURES = PROGRAMS

Programmierwissen konstituiert sich nach Wirth aus Programmierwissen über Algorithmen und Datenstrukturen. Für unsere Betrachtungen wollen wir die Datenstrukturen einmal ausschließen und uns auf die Algorithmen beschränken. Intuitiv gesehen ist ein Algorithmus ein allgemeines, effektiv (oder "mechanisch") ausführbares Verfahren (Albert & Ottmann, 1983, S. 197). Seine Präzisierung erfährt der Algorithmenbegriff (a) mit Hilfe von Maschinen, (b) über Klassen von Funktionen (c) über Worte verarbeitende Kalküle.

Da Algorithmen rein mechanisch ausführbar sind, können sie "im Prinzip" auch von Maschinen ausgeführt werden (Albert & Ottmann, 1983, S. 197). Als mathematische Modelle solcher Maschinen wurden eine Reihe von Automaten vorgeschlagen. Diese lassen sich nach Mächtigkeit aufsteigend ordnen: endliche Automaten, Keller-Automaten, TURING-Maschinen bzw. Registermaschinen.

Eine andere Präzisierung des Algorithmenkonzepts erfolgt so über die Angabe von Klassen von Funktionen: "Es werden aus der Mathematik Erfahrungen eingebracht, die beim Auswerten (Ausrechnen) von Funktionen gesammelt wurden. Dabei beschränkt man sich üblicherweise auf Funktionen mit Argumenten und Werten im Bereich natürlicher Zahlen (sog. "zahlentheoretische Funktionen"). Dies ist keine wesentliche Einschränkung, weil andere durch Algorithmen manipulierte Objekte sich meistens auf einfache Art durch natürliche Zahlen verschlüsseln lassen. Statt den Algorithmenbegriff zu präzisieren, definiert man den Begriff der berechenbaren zahlentheoretischen Funktion. Beispiele für derartige Definitionen sind: μ-rekursive, allgemein rekursive, partiell rekursive, λ-definierbare Funktionen" (Albert & Ottmann, 1983, S. 199f).

Die dritte Präzisierung von Funktionen erlebt zur Zeit unter dem Stichwort "Produktionssysteme" eine Blüte in der kognitiven Psychologie (s.a. Anderson, 1983; Sleeman,

1984). "Eine Analyse des 'schriftlichen' Rechnens und auch etwa im Bereich (formaler) Sprachen gemachte Beobachtungen führten dazu, Eingabe- und Ausgabedaten von Algorithmen stets als Worte (Zeichenreihen) über einem jeweils passenden endlichen Alphabet auffassen und einen Algorithmus als eine Vorschrift (einen Kalkül) zur schrittweisen Veränderung von Zeichenreihen zu definieren. Beispiele für solche Definitionen sind MARKOV-Algorithmen, POSTsche normale Systeme, Grammatiken (CHOMSKY-Sprachen) aber auch die bereits genannten TURING-Maschinen" (Albert & Ottmann, 1983, S. 200). Interessant für Forscher im Bereich der kognitiven Wissenschaften ist die Analyse von Anderson (1976). In seinem Buch *Language, Memory and Thought* zeigt er im Kapitel 3, das den Titel *Models of procedural knowledge* trägt, welche Beziehungen zwischen der *Stimulus Sampling Theory* und der Theorie endlicher Automaten bestehen. Zudem wird die Äquivalenz von MARKOV-Algorithmen Newellscher Prägung (den sogenannten Produktionssystemen) und von TURING-Maschinen an einem Beispiel demonstriert. Ferner zeigt Anderson im Kapitel 11 (*Language comprehension and generation*), daß eine spezielle Form von Automaten, die in der Sprachverarbeitung (*computational linguistics*) ein Standardmodell sind (*Augmented Transition Networks*) äquivalent zu TURING-Maschinen sind (Anderson, 1976, S. 453, 459).

Zur Äquivalenz von Präzisierungen des Algorithmusbegriffs über mathematische Maschinen, Klassen von Funktionen und wortersetzende Kalküle (d.h. Produktionssysteme) wollen wir wieder Albert und Ottmann (1983) zitieren: "Faßt man die Speicherinhalte eines Rechners als Darstellungen von natürlichen Zahlen auf, so läuft die durch einen Rechner verursachte Änderung der Speicherinhalte auf die Berechnung einer Funktion mit Argumenten und Werten im Bereich der natürlichen Zahlen hinaus. Schließlich ist auch die Manipulation von Worten nur eine andere Möglichkeit, die Tätigkeit eines Rechners zu sehen: Sowohl die Eingabe als auch die Ausgabe wird meistens in Form von 'Worten' über eine Tastatur ein- bzw. über einen Drucker o.ä. ausgegeben" (Albert & Ottmann, 1983, S. 200).

Welche der drei Präzisierungen (mathematische Maschinen, Markovalgorithmen, sowie rekursive Funktionen) eignen sich für den Erwerb mentaler Modelle, die die Programmierung real existierender oder sich in der Entwicklung abzeichnender Computer erleichtern sollen?

Betrachten wir zunächst die mathematischen Maschinen oder Automaten. Auf diesem Gebiet hat besonders Cohors-Fresenborg Anstrengungen unternommen:

"Bei dem Bemühen, Kindern schon im Grundschulalter einen spielerischen Zugang zu Problemen der Automation, der Entwicklung von Schaltnetzen und eine Vorbereitung auf das Programmieren von Computern zu ermöglichen, sahen wir uns veranlaßt, ein Material zu entwickeln, welches die Aspekte von Denken und Ordnen mit Handeln verknüpft. Es sollte weiter ermöglicht werden, daß sich das Ordnen nicht ausschließlich in einem begrifflichen Strukturieren niederschlagen muß, sondern daß es den Kindern auch die Möglichkeit gibt, das Strukturieren als Sequenzialisierung von Handlungen aufzufassen" (Cohors-Fresenborg, 1984, S. 325).

Cohors-Fresenborg begann 1974 damit, einen Baukasten "Dynamische Labyrinthe" zu entwickeln,der in die Hintergründe von Automaten und Computern einführt. "Vorstellungen über Spielzeugeisenbahnnetze werden als Einstieg in das Entwickeln von Algorithmen

benutzt" (Cohors-Fresenborg, 1984, S. 325). Der Autor nimmt auch Bezug auf die Repräsentationsnähe seines Lehrmaterials: "Im Sinne der Repräsentationsstufen von Bruner lassen sich drei Stufen unterscheiden: Bau und Durchfahren der Netze (enaktive), Zeichen (ikonische), Darstellung durch Automatentabellen (symbolische Ebene)" (Cohors-Fresenborg, 1984, S. 237ff).

Nachteilig für unsere schon im Titel dieses Beitrags angedeuteten Ziele ist aber einmal die Beschränkung von Cohors-Fresenborg auf endliche Automaten, die bekanntlich nicht zu allen Berechnungen in der Lage sind, und zum anderen auf die starke Akzentuierung der hardwareorientierten Ebene (FLIP-FLOP-Automaten).

Zur didaktischen Eignung der mächtigeren TURING-Maschinen schreiben Bauer und Goos (1982, S. 54): "Sehr viel detaillierter müssen häufig die Beschreibungen von Algorithmen sein, die auf sogenannten TURING-Maschinen (Turing, 1936) ausgeführt werden sollen. Für Einzelheiten über diese mehr theoretisch relevanten Fragen sei auf Vorlesungen über Algorithmentheorie verwiesen ...".

Ein ähnliches Urteil (bezüglich der *working memory load* des Lernenden) legen die Autoren über die MARKOV-Algorithmen ab: "Die mosaikhafte Beschreibung durch einzelne Ersetzungsoperationen erschwert unnötigerweise die Abfassung eines Algorithmus ebenso wie die Feststellung der Übereinstimmung der Niederschrift mit dem beabsichtigten Algorithmus. Bereits für die schulmäßige Durchführung einer Rechenoperation mit Zahlen in Ziffernschreibweise wird dabei die Beschreibung umfänglich. Das Werkzeug Rechenanlage legt dem Benutzer geradezu nahe, manche von Haus aus komplexen Verarbeitungsschritte als elementar anzusehen" (Bauer & Goos, 1982, S. 55).

Aus dieser Einschätzung heraus haben die Autoren und die Mitarbeiter des SFB 49 "Programmiertechnik" beim Übergang von der 2. zur 3. Auflage des zweibändigen Hochschulstandardwerks "Informatik - eine einführende Übersicht" (3. Auflage, 1982, 31.-34. Tausend) eine Beschreibungsform gewählt, die sich auf den Funktionsbegriff (dritte Präzisierung des Algorithmenbegriffs) stützt:

"Diese Algorithmen - wir werden sie Rechenvorschriften nennen - definieren Abbildungen durch formelhaften Aufbau (Zusammensetzung von Abbildungen durch Hintereinanderausführung) und Fallunterscheidungen, ihre Berechnungsstärke erhalten sie durch Zulassen der Rekursion. Diese Klasse, zuerst von McCarthy (1962) studiert, ist verwandt der Klasse der μ-rekursiven Funktionen der Logik, einer klassischen Präzisierung des Algorithmenbegriffs.

Als Grundlage des Aufbaus werden Objekte ('Daten') und (Rechen-)Operationen über Objekte benutzt, die auch abstrakt, also lediglich durch Angabe ihrer Eigenschaften ('axiomatisch') beschrieben sein können. Solche Einheiten von Objektmengen und zugehörigen (Rechen-)Operationen sollen primitive (Rechen-)Strukturen genannt werden" (Bauer & Goos, 1982, S. 55).

Die Rechenvorschriften werden als Kantorovic-Bäume oder Formulare dargestellt, die von der rekursiven Formularmaschine abgearbeitet werden kann. Die Formularmaschine wird von den Autoren auch als Gedankenmaschine bezeichnet (Bauer & Goos, 1982, S. 111). Bei dieser Namenswahl wird nicht ganz deutlich, ob die Autoren damit ein mentales Maschinen- oder Surrogatmodell anstreben (wie es z.B. Gentner, 1982,darstellt), oder ob sie nur eine idealisierte mathematische Maschine kreieren wollten. Möglich

wäre aber auch, daß sie beide Ziele gleichzeitig im Auge hatten. Jedenfalls werden die Programme für die Formularmaschine (d.h. die Formulare bzw. die Kantorovic-Bäume) Seite an Seite mit ALGOL-68-, CIP- und PASCAL-Programmen abgedruckt. Mit Hilfe der Formularmaschine wird die Berechnung von Formeln, Vorrang- bzw. Berechnungsregeln, rekursiver Funktionen, Iterationen etc. dargestellt.

3. Die Formularmaschine

Die Formularmaschine besteht - wie der Name schon sagt - aus einer Reihe von Formularen (z.B. DIN-A4 Blättern), die eine bestimmte Anordnung besitzen. Die Anordnung der Formulare in der Fläche ergibt den Kontroll- und Datenfluß. Formulare tragen Bezeichnungen für Operatoren, Parameter und eventuell Variablen. In die Formulare können im Laufe der Berechnungen Zwischenergebnisse eingetragen werden, wenn Vorergebnisse bekannt sind. Die Berechnungen können auch parallel (d.h. kollateral) erfolgen. Dabei wechselt die Formularmaschine zwischen datengesteuerter (*data driven*) und zielgesteuerter (*demand driven*) Berechnung, wenn die Situation dieses erforderlich macht. So werden normale Formeln datengesteuert ausgewertet ("strikte Auswertung" nach Bauer & Goos) und Verzweigungen zielorientiert bzw. *demand driven* ("nichtstrikte" Auswertung nach Bauer & Goos).

Weil das Erstellen von Formularanordnungen gleichbedeutend mit der Programmierung der Formularmaschine in einer syntaxarmen aber nicht string- sondern bildorientierten Programmiersprache ist, sollte die Beschäftigung damit als *prototype learning* (Clark & Voogel, 1985) interpretiert werden. Wir erwarten uns einen Transfer für das Erlernen einer "normalen" Programmiersprache. Der Transfer sollte am größten sein, wenn diese Programmiersprache ebenfalls syntaxarm und strukturähnlich zur Formularmaschine ist. Wir erwarten den größten Transfer beim Erlernen von diSessas BOXER Programmiermedium. Wenn wir dagegen Transferhypothesen zu klassischen stringorientierten Programmiersprachen formulieren, dann würden wir den größten Effekt beim Erlernen von SCHEME (einem LISP-Dialekt mit lexikalischer Bindung freier Variablen) postulieren. SCHEME ist unseres Erachtens nach diejenige Programmiersprache, die die einfachste Syntax aller uns bekannten Programmiersprachen besitzt. Interessanterweise wird SCHEME bei der Grundausbildung im Computersciencecurriculum am MIT seit vielen Jahren eingesetzt (Abelson, Sussman & Sussman, 1984). Noch interessanter ist, daß gerade dieser Abelson als Mitautor von diSessa das bekannte Buch über *Turtle Geometry* geschrieben hat (Abelson & diSessa, 1981).

Als wesentlich geringer erwarten wir den positiven Transfer durch das Programmieren der Formularmaschine auf das Erlernen des im Bauer und Goos (1982) verwendeten PASCAL, CIP oder ALGOL-68, was hauptsächlich auf die komplizierte Syntax und Typisierung in diesen Sprachen zurückzuführen ist.

Die Vorläuferkonzeption der "Formularmaschine" findet sich als Rechenbaum für Grundrechenarten im Curriculum Mathematik der Orientierungsstufe (5. Jahrgangsstufe) (Schmitt-Wohlfarth, 1978) (vgl. Abbildung 2). Es gibt Rechenbäume für das Addieren, Subtrahieren, gemischte Operationen und für das Rechnen mit Klammern sowie Merkbäume.

1. Vom Term zum Rechenbaum

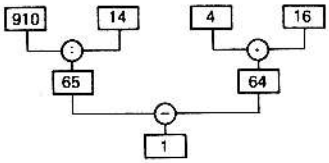
Berechne den Wert des Termes $910 : 14 - 4 \cdot 16$ *und zeichne den Rechenbaum!*

$\underline{910 : 14} - \underline{4 \cdot 16} =$

$= \quad 65 \quad - \quad 64 \quad =$

$= \quad \underline{1}$

* Termname: Differenz



Termberechnung

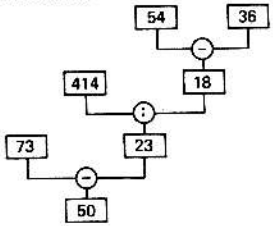
$73 - 414 : (54 - 36) =$

$= 73 - 414 : 18 \quad =$

$= 73 - \quad 23 \quad =$

$= \quad \underline{50}$

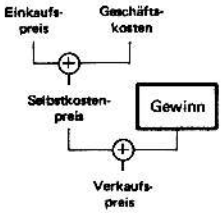
*Rechenbaum

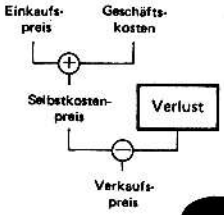


*Termname: Differenz

Einkauf – Verkauf

Merkbäume





1. Der Geschäftsinhaber eines Supermarktes bestellt 45 kg einer Käsesorte, das kg zu 4,20 DM. Es sind ihm zusätzlich Kosten von 45,80 DM entstanden. 1 kg dieser Käsesorte verkauft er um 6,80 DM. Wieviel verdient er an dem Käse insgesamt?
(*Gesamtterm) [10]

Abb. 2: Rechenbäume und Merkbäume.

"Rechen-" und "Merkbäume" stellen bipartite Graphen dar, mit einer Halbordnung der Rechenoperationen "früher-zu-berechnen-als". Während bei Rechenbäumen mit ausgewerteten Argumenten gerechnet wird, trifft man in "Merkbäumen" nur ungebundene Variablen an.

In "Merk/Rechenbäumen" sind alle Berechnungen endlich. Das trifft auf die Programme der Formularmaschine nur teilweise zu. So sind zwar die Programme für die Formularmaschine, die den Merk- bzw. Rechenbäumen in Schmitt-Wohlfarth entsprechen, statisch wie dynamisch endlich. Dennoch sind sie aber nicht für praktische Berechnungen geeignet, weil ihnen rekursive oder iterative Strukturen fehlen. Deshalb sind in der

Formularmaschine Verzweigungen, Rekursion und Iteration möglich, um alle berechenbaren Funktionen auf der Formularmaschine programmieren zu können.
Ein Programm wird dabei als ein formales Objekt angesehen:
"(Das Programm als formales Objekt) komprimiert die syntaktische Repräsentation durch den Programmtext, die durch die Input-Output-Funktion definierte Semantik und das operationale Verhalten des Algorithmus" (Dosch, 1984, S. 153).

Die erste Erweiterung gegenüber den Rechenbäumen betrifft die Festlegung einer Auswertungsregel einer Formel (z.B. "parallel" vs. *leftmost-innermost*). Dazu sind in der Abbildung 3 zwei Beispielprogramme dargestellt.

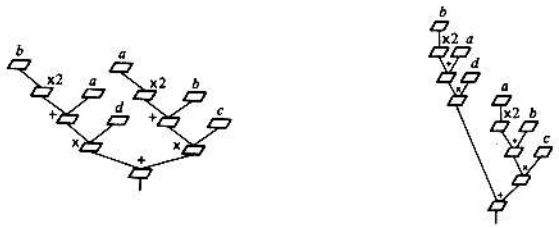


Abb. 3: Parallele Berechnung vs. Berechnung nach dem "Kellerprinzip".

Die nächste Erweiterung erfährt die Formularmaschine durch ihre Fähigkeit, bedingte Auswertungen vorzunehmen. Dazu folgt in der nächsten Abbildung wieder ein Beispiel (s. Abbildung 4).

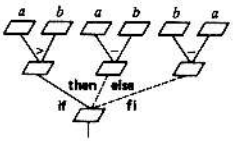


Abb. 4: Bedingte Auswertung durch Fallunterscheidung.

Die Auswertung erfolgt nach der *demand-driven* Strategie: Es ist zuerst die Bedingung einer Alternative auszuwerten und dann, je nach ihrem Ausfall, der nicht benötigte Zweig zu kappen; der verbleibende Zweig ist sodann auszuwerten" (Bauer & Goos, 1982, S. 96).
Für Rechenvorschriften können Parameter und Namen definiert werden. So erhält die Formel

$$(u + v)\ \mathrm{div}\ (u - v)$$

den Namen f, wobei u und v Parameter sind. Das entsprechende Formularmaschinenprogramm findet sich in Abbildung 5.

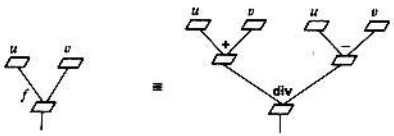


Abb. 5: Rechenvorschrift mit Namen f und Parametern u und v.

Durch die Einführung von benannten Rechenvorschriften können nach dem "Teile und Herrsche"-Prinzip hierarchisierte Systeme von Rechenvorschriften programmiert werden. Man kann z.B. die Formel

(b * 2+a) * d+(a * 2+b) * c

hierarchisieren zu

h(b,a,d) + h(a,b,c)

mit:

h: (a,b,c) ⟼ (a * 2+b) * c

Das entsprechende Programm für die Formularmaschine findet sich in Abbildung 6.

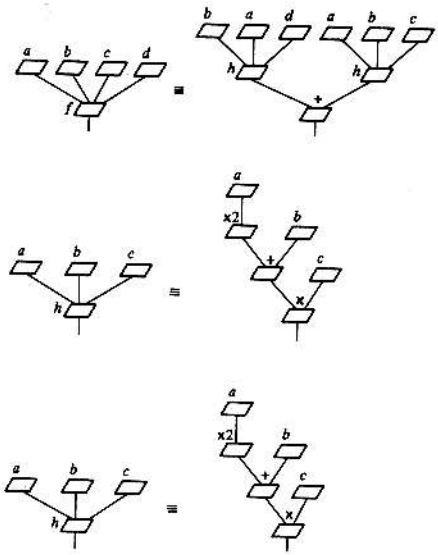


Abb. 6: Hierarchisiertes Formularmaschinenprogramm.

Die Mächtigkeit einer TURING-Maschine erhält die Formularmaschine durch ihre Fähigkeit, rekursive Rechenvorschriften auszuführen.
Die Standardaufgabe "Fakultät von n" kann als 'Drosophila' der Informatik angesehen werden. Daher ist es nicht verwunderlich, wenn Bauer und Goos, die rekursive Formularmaschine und ihre Berechnungsweise gerade an diesem Beispiel ausführlich demonstrieren. Die Funktion der Fakultät ist definiert als

$$n! \quad = \quad \begin{array}{l} \text{Wenn } n = 0, \text{ dann } 1 \\ \text{sonst } n \star (n - 1)\ ! \end{array}$$

Die Formularmaschine findet sich in Abbildung 7.

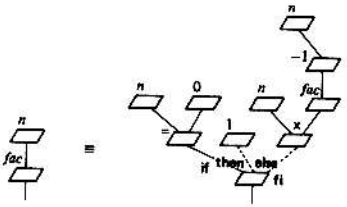


Abb. 7: Programm zur Berechnung von n! auf der rekursiven Formularmaschine.

Der operativen Semantik der rekursiven Formularmaschine widmen Bauer und Goos das Kapitel 2.3.3 (s.a. Abbildung 8).

Die (rekursive) Formularmaschine

Der Gang der Berechnung einer Rechenvorschrift ist bis auf Kollateralität durch ein zugehöriges Formular festgelegt. Kommt im Formular selbst wieder eine Rechenvorschrift als Operation vor[44], so ist ein Formular dieser Rechenvorschrift anzulegen („Aufruf") und deren Ergebnis schließlich rückzuübertragen. Dies gilt auch für eine rekursiv definierte Rechenvorschrift – mit der Besonderheit, daß im Lauf der Berechnung entsprechend den rekursiven Aufrufen weitere Exemplare des Formulars eben dieser Rechenvorschrift benötigt werden.

Zu jedem Aufruf werden in ein neues Exemplar des Formulars zunächst linksseitig die jeweiligen Argumentwerte eingetragen (*'call by value'*). Man nennt jedes solche Exemplar eine **Inkarnation** der Rechenvorschrift; um die Übersicht zu behalten, kann man die Inkarnationen und die entsprechenden Aufrufe im Verlauf der Berechnung durchnumerieren.

Für den rekursiven Fall ist es nun besonders bedeutsam, daß die Fallunterscheidung eine arbeitssparende Auswahl trifft: nachdem die Parameterbezeichnungen durch die linksseitig festgestellten Argumentwerte ersetzt sind, werden daher auf dem Urformular und allen folgenden Inkarnationen möglichst zuerst die Bedingungen ausgewertet und sodann die unzulässigen Zweige gekappt. Die Rekursion endet mit Inkarnationen, in denen kein Zweig mehr verbleibt, der einen rekursiven Aufruf enthält. Die ganze Berechnung **terminiert** (für einen bestimmten Parametersatz), wenn sie nur endlich viele Inkarnationen benötigt.

Die Tätigkeit eines Menschen, der auf diese Weise mit Formularen arbeitet, kann in einsichtiger Weise auch mechanisiert werden. Man gelangt so zum Begriff einer rekursiven (Gedanken-)Maschine, der **Formularmaschine**, in der die volle Freiheit der Berechnung noch erhalten ist. Man beachte, daß ein neues Exemplar eines Formulars auch dann angelegt wird, wenn die gleichen Argumente schon einmal aufgetreten sind: die Formularmaschine macht (auf der hier geschilderten Stufe) von einer möglichen Mehrfachverwendung eines Ergebnisses keinen Gebrauch.

Das oben erwähnte Kappen von Zweigen ist insbesondere dann ohne weiteres möglich, wenn in den Bedingungen keine rekursiven Aufrufe vorkommen. Noch übersichtlicher ist der Fall der **linearen Rekursion**, bei der außerdem in den einzelnen Zweigen der Fallunterscheidung höchstens ein rekursiver Aufruf vorkommt; dann wird nämlich in jeder Inkarnation höchstens eine neue Inkarnation angestoßen. Fast alle bisher behandelten Beispiele fallen übrigens in diese Klasse.

Für *fac* von 2.3.2 arbeitet eine Formularmaschine wie in Abb. 59 angegeben. Typisch für die Rekursion ist das „Nachklappern" der Berechnung: Erst wenn die Rekursion mit der Inkarnation $fac^{(3)}$ geendet hat, werden die zurückgestellten Berechnungen in $fac^{(2)}$, $fac^{(1)}$ und $fac^{(0)}$ durchführbar und auch durchgeführt[45]; das Urformular $fac^{(0)}$ liefert schließlich das Endergebnis. Das Nachklappern kann in besonders gelagerten Fällen von Aufrufen zu einem bloßen Rückübertragen der Ergebnisse der einzelnen Inkarnationen degenerieren, wie Abb. 60 für das Beispiel *gcd1*(15, 9), vgl. 2.3.2 zeigt. Ein solcher Aufruf heißt **schlicht**. Wenn in linearer Rekursion ausschließlich schlichte Aufrufe vorliegen, spricht man von **repetitiver Rekursion.**

Bei linear rekursiven Rechenvorschriften ist – abgesehen von der sonstigen Kollateralität des Formulars – die Reihenfolge, in der die benötigten Inkarnationen angestoßen werden, eindeutig bestimmt. Dies ist nicht notwendig so im allgemeinen Fall: wenn in einem Zweig mehrere Aufrufe vorkommen, so erlaubt die Kollateralität unter Umständen verschiedene Reihenfolgen und sogar Parallelarbeit.

[44] Für primitive, d.h. den zugrundeliegenden Rechenstrukturen entstammende Operationen ist kein Formular erforderlich.

Abb. 8: Operative Semantik der rekursiven Formuliermaschine.

Der Berechnungsgang für fac(3) ist in Abbildung 9 dargestellt. Schattierte Rauten bleiben unberücksichtigt, weil die Bedingung verhindert, daß dieser Zweig des Formulars ausgewertet wird. Gestrichelte Rauten verweisen auf "nachklappernde" Operationen. "Nachklappernde" Operationen können erst nach Vorliegen des ersten rekursiven Teilergebnisses (hier: fac(o) = 1) in umgekehrter Reihenfolge abgearbeitet werden.

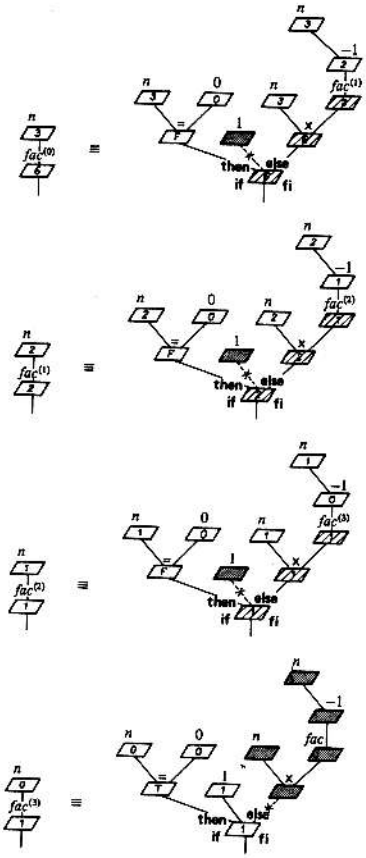


Abb. 9: Berechnungsfolge der Formularmaschine für fac(3) = 3!

Erweiterungen erfuhr die Formularmaschine in Richtung variablenorientierter, prozeduraler, maschinenorientierter Programmierung mit iterativen Kontrollstrukturen. Durch Überdeckung und Abrollen von Formularen lassen sich entsprechende iterative Programme erstellen (Bauer & Goos, 1982, S. 157-164). Da wir diese Erweiterungen im Moment wegen ihrer Komplexität nicht betrachten wollen, verzichten wir hier auf eine entsprechende Darstellung.

4. Forschungsziele

Welche Forschungsziele lassen sich formulieren? Da wäre als oberstes Ziel die Entwicklung eines kompletten "intelligenten" Computertutors auf dem referierten Diskursbereich. Wissenspsychologisch interessant und für die zukünftige Entwicklung der Kognitionspsychologie relevant wäre hierbei die Entwicklung einer adaptiven Wissensdiagnostik- und Fehlererklärungskomponente. Erstere wäre unerläßlich, um eine Über- oder Unterforderung des Lernenden zu vermeiden. Letztere sollte Lernfortschritte akzelerieren. Beide Komponenten müssen zusammenspielen, um die Warnung Einarssons (1977) und die Forderung Anderson et al. (1984) zu berücksichtigen. Einarsson warnte "*Do not use the computer for tasks that can be carried out by simpler means*" und

Anderson forderte, daß der Computertutor dem menschlichen Einzel (oder Nachhilfe) unterricht mindestens ebenbürtig sein sollte.

Ein Computertutor hätte für diesen Diskursbereich gute Chancen, diese Kriterien zu erfüllen, weil der Lehrinhalt überschaubar ist und eine differenzierte Fehleranalyse möglich ist. Es ist kaum Vorwissen notwendig, d.h. dem Lernenden müssen nicht zunächst wichtige Voraussetzungen im Frontalunterricht vermittelt werden, bevor er ein Tutorsystem "Formularmaschine" sinnvoll nutzen könnte.

Ein weiterer Grund für die Schaffung eines Tutors wäre die Beobachtung, daß es für diesen Inhaltsbereich trotz seiner Relevanz (s.a. Dosch, 1984) keine anerkannte didaktische Standardmethode existiert. Daher kommt es, daß der Lernende metasprachliches Programmierwissen über einen langwierigen Abstraktionsprozeß quasi "nebenbei" erwirbt.

Der wichtigste Grund aber liegt im *learning by doing*. Dieses kann durch Frontalunterricht oder Trocken-Übungen im Hochschulbereich nicht ermöglicht werden. Zumal es oft sehr schwer ist, die Konsequenzen bestimmter Programmelemente vorherzusagen. Insbesondere mit der Rekursion haben Lernende Schwierigkeiten (Anzai & Uesato, 1982). Sie erfordert Verständnis für Induktionsschlüsse, Prozesse bzw. Operationen mit Eigendynamik. Gilt die Endrekursion oder *tail recursion* noch als einfache Alternative zu einem Sprung, erwachsen den meisten Programmierern heftigste Schwierigkeiten, wenn rekursive Aufrufe in den Prädikaten von IF- THEN- ELSE-Verzweigungen auftreten.

Deshalb sind Bewegungsabläufe bzw. Animationen von zentraler Bedeutung: Sie können dem Lernenden die Konsequenzen seiner Programme unmittelbar vor Augen führen und erklären. Der Lernende sieht, wo was aus welchen Gründen geschieht, und lernt dabei, Fehlerquellen schnell zu lokalisieren. Hierbei sollte ihn die zu konstruierende Fehlererklärungskomponente unterstützen.

Wissensdiagnostik- und Fehlererklärungskomponente setzen nun ihrerseits andere Modelle voraus. Wir müssen das Experten- und das Novizenwissen modellieren. Aus dem Vergleich beider Wissensbasen müssen Defizite und Fehlkonzepte identifiziert werden. Es muß ein Partnermodell des Lernenden entwickelt werden, das sowohl *facts* und *rules* sowie *malfacts* und *malrules* enthält.

Bei alledem darf die Realisierungsmöglichkeit nicht aus dem Auge verloren werden. Anderson (1984) gibt zum Beispiel 10 Mannjahre für die Entwicklung eines kompletten Tutors an. Diese Zahl deckt sich mit anderen Erfahrungen. Es kann also durchaus sein, daß im Rahmen der DFG ein solches umfassendes Projekt gar nicht finanzierbar ist, so daß man rein fachdidaktische oder rein informatorische Inhalte auskoppeln müßte, um andere Partner für die Finanzierung derartiger Tutoren zu gewinnen.

LITERATUR

Abelson, H., & DiSessa, A. (1981). Turtle Geometry. The Computer as a medium for exploring mathematics. Cambridge, MA: MIT Press.

Abelson, H., Sussman, G., & Sussman, J. (1984). Structure and interpretation of computer programs. Cambridge, MA: MIT Press.

Albert, J., & Ottmann, Th. (1983). Automaten, Sprachen und Maschinen für Anwender. Mannheim: BI-Wissenschaftsverlag.

Anderson, J.R. (1976). Language, memory and thought. Hillsdale, N.J.: Erlbaum.

Anderson, J.R. (1980). Cognitive psychology and its implications. San Francisco: Freeman.

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

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

Anderson, J.R.,Boyle,C.W., Farrell, R., & Reiser, B. (1984). Cognitive principles in the design of computer tutors. Pittsburgh: CMU.

Anzai, Y., & Uesato, Y. (1982). Is recursive computation difficult to learn? Proceedings of the 41th Annual Conference of the Cognitive Science Society. Ann Arbor: University of Michigan.

Barstow, D.R. (1979). Knowledge based program construction. New York: North Holland.

Bauer, F.L., & Goos, G. (1982). Informatik. Eine einführende Übersicht, 1. Teil. Berlin: Springer (3. Auflage).

Brooks, R. (1977). Toward a theory of the cognitive processes in computer programming. International Journal of Man-Machine Studies, 9, 737-751.

Clark, R.E., & Voogel, A. (1985). Transfer of training principles for instructional design. Educational Communication and Technology Journal, 33. (in print)

Cohors-Fresenborg, E. (1984). Dynamische Labyrinthe: Ein Einstieg in die Computerwelt. In W. Arlt & G. Häfner (Hg.), Informatik als Herausforderung an Schule und Ausbildung (S. 235-239). Berlin: Springer.

Davis, A.L., & Keller, R.M. (1982). Data flow program graphs. Computer, 15, 26-41.

diSessa, A. (1985). Design considerations for an integrated computational environment: BOXER. Vortrag auf dem 14. Fachgespräch der Fachgruppe Interaktive Systeme, des German Chapter of the Association for Computing Machinery, 13.-15.5.1985, TH Darmstadt.

Dosch, W. (1983). New prospects of teaching programming language. In F.B. Lovis & E.D. Tag (Eds.), Informatics education for all students at university level (pp. 153-169). Amsterdam: North-Holland.

Einarsson, G. (1977). Computer-assisted instruction in communication engineering. Didakometry, 57.

Gentner, D., & Stevens, A.L. (Eds.) (1982). Mental models. Hillsdale, N.J.: Erlbaum.

Green, T.R.G. (1980). Programming as a cognitive activity.In H.T.Smith & T.R. Green (Eds.), Human interaction with computers (pp. 271-320). New York: Academic Press.

Green, T.R.G. (1983). Learning big and little programming languages. In A. Wilkinson (Ed.), Classroom computers and cognitive science (p.71-93). New York: Academic Press.

Gurd, J.R., Kirkham, C.C., & Watson, I. (1985). The Manchester prototype data flow computer. Communications of the ACM, 28, 34-53.

Hoffmann, H.J. (1985). Interaktive Systeme nach 1984 und SMALLTALK als Beispiel einer interaktiven Arbeitsumgebung. 14. Fachgespräch der FG Interaktive Systeme des German Chapter of the Association for Computing Machinery, 13.-15.5.1985, TH Darmstadt.

Miller, L.A. (1981). Natural-language programming: styles, strategies, and contrasts. Perspectives in Computing, Vol. I, No. 4, 22-33.

Möller, B. (Ed.) (1984). A survey of the project CIP: computer-aided, intuition-guided programming-wide spectrum language and program transformations. München: Technische Universität.

Paseman, W.G. (1985). Applying data flow in the real world. Byte, 5, 201-214.

Pea, R.D., & Kurland, D.M. (1984). On the cognitive prerequisites of learning computer programming. (Unpublished Manuscript).

Pountain, D. (1985). Parallel processing. Byte, 5, 385-395.

Scandura, J.M. (1977). Problem solving. New York: Academic Press.

Schmitt, H., & Wohlfarth, P. (1978). Mathematikbuch 5N. München: Bayerischer Schulbuchverlag.

Sleeman, D. (1984). An attempt to understand students' understanding of basic algebra. Cognitive Science, 8, 387-412.

Wirth, N. (1976). Algorithms + data structures = programs. Englewood Cliffs, N.J.: Prentice Hall.

Young, R.M. (1981). The machine inside the machine: user's models of pocket calculators. International Journal of Man-Machine Studies, 15, 51-85.

Lernen im Dialog mit dem Computer

Herausgegeben von
Heinz Mandl und Peter Michael Fischer

URBAN & SCHWARZENBERG
München – Wien – Baltimore 1985

Anschrift der Herausgeber:

Prof. Dr. Heinz Mandl
Dipl.-Psych. Peter M. Fischer
Deutsches Institut für Fernstudien
an der Universität Tübingen
Hauptbereich Forschung
Bei der Fruchtschranne 6
7400 Tübingen

Anschriften des Wissenschaftlichen Beirates:

Prof. Dr. Dieter Frey, Institut für Psychologie der Universität Kiel,
Olshausenstraße 40/60, 2300 Kiel

Prof. Dr. Siegfried Greif, Universität Osnabrück, FB 8 Psychologie,
Knollstraße 15, 4500 Osnabrück

Prof. Dr. Heiner Keupp, Institut für Psychologie, Sozialpsychologie Universität München,
Kaulbachstraße 93/II, 8000 München 40

Prof. Dr. Ernst-D. Lantermann, Gesamthochschule Kassel, FB 3,
Heinrich-Plett-Straße 40, 3500 Kassel

Prof. Dr. Rainer K. Silbereisen, Institut für Psychologie, FB 2, Technische Universität Berlin,
Dovestraße 1–5, 1000 Berlin 10

Dr. Bernd Weidenmann, Universität der Bundeswehr München, Fachbereich Sozialwissenschaften,
Werner-Heisenberg-Weg 39, 8014 Neubiberg

Lektorat:

Dr. H. Jürgen Kagelmann

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Lernen im Dialog mit dem Computer / hrsg. von Heinz Mandl u. Peter Michael Fischer. – München ; Wien ; Baltimore : Urban und Schwarzenberg, 1985.
(U-&-S-Psychologie)
ISBN 3-541-14261-8

NE: Mandl, Heinz [Hrsg.]

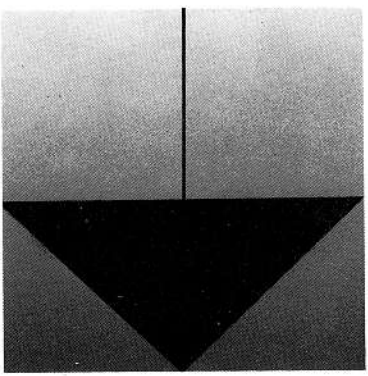
Alle Rechte, auch die des Nachdruckes, der Wiedergabe in jeder Form und der Übersetzung in andere Sprachen behalten sich Urheber und Verleger vor. Es ist ohne schriftliche Genehmigung des Verlages nicht erlaubt, das Buch oder Teile daraus auf fotomechanischem Weg (Fotokopie, Mikrokopie) zu vervielfältigen oder unter Verwendung elektronischer bzw. mechanischer Systeme zu speichern, systematisch auszuwerten oder zu verbreiten (mit Ausnahme der in §§ 53, 54 URG ausdrücklich genannten Sonderfälle).

Umschlagentwurf: Dieter Vollendorf
Druck und Bindung: F. Pustet, Regensburg.
Printed in Germany.
© Urban & Schwarzenberg 1985

ISBN 3-541-14261-8

Mandl/Fischer
(Hg.)

Lernen im Dialog mit dem Computer



Forschung
Psychologie
Urban & Schwarzenberg