Peter Bender, Kassel

Zum didaktischen Wert der Rekursions-Technik

(Beitrag zum deutsch-französischen mathematikdidaktischen Symposium im November 1986 in Marseille)

Für einen extensiven Einsatz des Computers in der Schule (über dessen Möglichkeiten als Medium und Werkzeug hinaus) gibt es im großen zwei Begründungsstränge: einen von der Informatik und einen von der Erkenntnistheorie kommend (genauer: von deren durch Computer-Fachleute bestimmten Richtung). Trotz unterschiedlicher Argumentationszusammenhänge wird bei beiden die Technik des "strukturierten Programmierens" als Bildungsinhalt, d.h. potentieller Träger von Bildungsgehalt, propagiert.

Hier und im folgenden interessiert nicht eine etwaige Spezialdefinition dieser Programmier-Technik aus der Disziplin 'Informatik', sondern die Begrifflichkeit, wie sie sich in der didaktischen Diskussion darstellt. Auch will ich jetzt einmal die Einwände dagegen beiseite lassen, obwohl ich sie für bedeutsam halte (gegen das pragmatische Argument: fast kein Mensch muß jetzt oder in Zukunft programmieren können, auch nicht in der Schule, es sei denn im Informatik-Unterricht; gegen das didaktische Argument: die Kognitionsforschung hat keine Hinweise darauf, daß da quasi automatisch ein Transfer von Denk"prozeduren" aus einem Bereich in andere stattfindet, zumal wenn einer der Bereiche vom Computer geprägt ist und die anderen nicht).

Mit "strukturiertem Programmieren" ist im wesentlichen der konsequente Einsatz der Unterprogramm-Technik gemeint. Diese wiederum ist Voraussetzung für rekursives Programmieren, d.h. für den Aufruf von Unterprogrammen durch sich selbst, eine Technik, die in gewissen Programmiersprachen möglich ist, z.B. in Logo. Da diese Programmiersprache mit dem Ziel entwickelt wurde, die Verbreitung des Computers in den Schulen zu befördern, und die Propaganda für dieses Ziel in Form der sog. Logo-Philisophie bei nicht wenigen Pädagogen in der

westlichen Welt gut ankam, ist auch die Rekursions-Technik als Spielart der allgemeinen Idee der Iteration in die didaktische Diskussion gelangt.

Iteration (i.w.S.) ist ein mathematischer (bzw. informatischer) Begriff, dem genügend Weite (logische Allgemeinheit), Fülle (vielfältige Anwendbarkeit und Relevanz) und Sinn (Verankerung im Alltagsdenken, lebensweltliche Bedeutung) zukommt, so daß er im Sinne Schreibers (1983) eine universelle Idee der Mathematik (gewiß auch der Informatik) ist. Die Frage, ob der Iterations-Gedanke durch die geläufigen Beispiele aus der Mathematik wie Multiplikation als fortgesetzte Addition, Zinseszinsrechnung, das Messen, Infinitesimalrechnung usw. im Schulunterricht bereits angemessen repräsentiert ist oder nicht noch eine Darstellung in entsprechenden Computer-Programmen erforderlich ist, diese Frage ist keineswegs abschließend geklärt. Für das Folgende unterstelle ich aber einmal eine Entscheidung für das Programmieren in der Schule, und zwar selbstredend in einer höheren Programmiersprache. Was spricht dann dafür, die Idee der Iteration gerade in der Rekursions-Technik zu verkörpern, wie sie für Logo charakteristisch ist, und nicht etwa in Iterations-Schleifen, wie sie in vielen geläufigen Programmiersprachen vorkommen? Diese beiden Alternativen sollen nun diskutiert werden, wobei ich die Prädikate 'rekursiv' und 'iterativ' entsprechend verwende, obwohl diese als mathematische Begriffe eigentlich inkompatibel sind. Angesichts vieler grundsätzlicher pädagogischer Fragen bei der Computerisierung der Schule erscheint diese Alternativen-Abwägung eigentlich als technisches Problem minderer Bedeutung; aber Logo-Anhänger heben immer wieder den besonderen didaktischen Wert der Rekursion hervor (Papert 1980:71, 74, Hoppe 1984:102, 135, 137, Ziegenbalg 1984, Löthe 1985b:198 u.v.a.).

Bestimmt legt das Erreichen einer gewissen Stufe der Programmier-Fertigkeit in Verbindung mit dem Übergang zu bestimmten Problem-Klassen die Verwendung der Rekursion nahe. Zwar läßt sich jede Rekursion auch als Iteration programmieren (umgekehrt sowieso), aber schon die Abarbeitung einer nicht allzu primitiven Baumstruktur mit Hilfe von Iterations-Schleifen kann sehr aufwendig, fehleranfällig und debugging-feindlich werden. Von der Logo-Philosophie her mit ihrer Orientierung an der AI-Forschung ist die Rekursion wohl eine für den fortgeschrittenen Programmierer geeignete Technik, jedenfalls bei Zugrundelegung einer dynamischen Programmiersprachen-Konzeption, wie sie bis in die siebziger Jahre dominierte. Für einen Schulunterricht, der eher an herkömmlichen Ausbildungsvorstellungen ausgerichtet ist, schätze ich, auch bei stärkerer Einbeziehung des Computers, die Rekursion als weniger relevant ein:

Für die Beispiele, in denen sie einen echten Programmier-Vorteil darstellt, fehlt es an einigermaßen überzeugenden didaktischen Analysen, aus denen sich eine Rechtfertigung zur Aufnahme in den Stoffkanon der allgemeinbildenden Schule ableiten ließe. Daß das Programmieren solcher Beispiele (Turm von Hanoi, Zeichnung eines Baumdiagramms usw.) eine Form von Problemlösen ist, an der man die Rekursions-Technik lernen kann, ist ein partiell zirkelhaftes Argument und reicht m.E. nicht für die breite Mehrheit der Schüler eines Jahrgangs.

Für Hoppe (1984:102) und Löthe (1985:198 u.a.) ist rekursives Programmieren "mathematiknah" und trägt zur Überwindung des "von Neumann'schen Arbeitsprinzips" der herkömmlichen Computer mit ihrem linear-sequentiellen Durchlaufen der Befehlsfolgen bei. Dieser Einschätzung liegt wohl eine Art Komfortabilitäts-Skala für Programmiersprachen zugrunde: Auf der einen Seite vielleicht Assembler-Sprachen, wo eine elementare Rechenoperation aus mehreren Einzelbefehlen zusammengesetzt werden muß, bis hin auf der anderen Seite vielleicht zu Lisp, wo ein rekursiver Ausdruck direkt hingeschrieben werden kann. Diese Skala läßt sich allerdings verlängern bis hin zu Programmiersprachen(-Systemen), mit denen z.B. Polynomgleichungen nach Aufstellung automatisch gelöst werden, die also in diesem Sinn noch viel "mathematiknäher" sind. Wieso soll gerade Logo auf dieser Skala als Schulsprache, auch noch als erste (und einzige), ausgewählt werden?

Keine Programmiersprache scheint mir besser als andere geeignet zu sein, der psychischen Gefahr zu begegnen, die vom Computer ausgeht, wenn er unfehlbar, von keinen Zweifeln geplagt, von keinen Emotionen gerührt, Befehl nach Befehl ausführt und die Denkweise des Programmierers ursurpiert und ihn zu "computerhaftem, linearem" Denken verleitet. Der mögliche Beitrag der Rekursions-Technik zur Abwendung dieser Gefahr wirkt doch sehr bescheiden, zumal - und das gilt für alle diese Programmier-Feinheiten - was man vielleicht an Programmier-Aufwand spart, muß man als Mühe beim Durchschauen wieder einsetzen. Für den versierten Programmierer mag diese Mühe gering und sowieso überflüssig sein, für den Anfänger ist sie lange Zeit wichtig, z.B. ganz pragmatisch im Hinblick auf das Debugging, aber auch aus grundsätzlichen Erwägungen. Ohne diesen Anspruch des Durchschauens ließe sich nicht rechtfertigen, warum Schüler (Lernende) überhaupt programmieren sollen und nicht gleich fertige Programme bzw. Anwender-Software benutzen.

Und auch den Beitrag des Programmierens zur (etwa mathematischen) Begriffsbildung als dritte Phase nach dem verständigen Umgehen mit einem Begriff und seiner Formalisierung halte ich mit einer "mathematiknahen" Programmiersprache für weniger gegeben als gerade mit einer, die den Schüler zwingt, sich die lineare Abarbeitung eines Algorithmus Schritt für Schritt klar zu machen. Anthropomorphisierend und übertrieben ausgedrückt: Mit einer "mathematiknahen" Programmiersprache ist der "tutee" 'Computer' zu intelligent und verlangt dem Schüler zu wenig Erklärens-Leistung ab.

Rekursive Prozedur-Aufrufe können den Rechenablauf verschleiern (so schon Oberschelp 1977:45) - das ist geradezu ihre Funktion; denn sie sollen den Programmierer davon entlasten, diesen Ablauf austüfteln zu müssen. Diese ihre Funktion weist sie insofern als anti-didaktisch aus, jedenfalls in bezug auf den Programmier-Novizen, also in einem weiten Bereich der Schule. Man sollte sich vor Augen führen, daß Logo zwar eigens für Kinder entwickelt wurde, aber nicht etwa aus 'interaktiven' Sprachen heraus durch dezidierten Einbau der Rekursions-Technik, sondern als Vereinfachung der sowieso 'rekursions-haltigen' Programmiersprache 'Lisp'.

Es wird wohl von niemanden bestritten, daß die Rekursions-Auffassung eines Terms schwieriger zu durchschauen ist als die Iterations-Auffassung, wobei die Auffassungen in folgendem Sinn gemeint sind: Es sei der Ausdruck $a_n = f(a_{n-1})$ (mit $n \in \mathbb{N}$) gegeben. Ihn iterativ auffassen heißt, sich den Gesamtaufbau so vorstellen: a_0 , $a_1 = f(a_0)$, $a_2 = f(a_1)$, ..., $a_n = f(a_{n-1})$ ... Dagegen bedeutet die rekursive Auffassung: Für jedes n gilt $a_n = f(a_{n-1}) = f^2(a_{n-2}) = \ldots = f^n(a_0)$, und dann $a_1 = f(a_0)$, $a_2 = f(a_1)$, ... , $a_n = f(a_{n-1})$. Iterativ vorgehen heißt also, von einem bekannten Wert ausgehend mit der Operation f einen zweiten Wert, danach einen dritten usw. ermitteln; rekursiv vorgehen bedeutet, einen gesuchten, unbekannten Wert als dadurch gegeben zu betrachten, daß er aus einem anderen, allerdings ebenfalls unbekannten Wert durch die Operation f hervorgeht. Während sich für den Iterierer - mit Recht - gar nicht erst die Frage stellt, ob er an sein Ziel kommt, wundert sich der Rekurrierer, jedenfalls wenn er mathematisch weniger versiert ist, wieso ein unbekannter Wert durch Rückführung auf einen anderen, ebenfalls unbekannten, nun (im Prinzip) bekannt sein soll. Er muß in der Vorstellung Schicht für Schicht diese Rückführung weitertreiben und befindet sich dauernd auf scheinbar schwankendem Boden. Während der Iterierer nach jedem Schritt die alten Ergebnisse vergessen kann, muß sich der Rekurrierer die ganze Rückführungs-Kette merken, da er, sobald er auf einen wirklich bekannten Wert stößt, von diesem aus wieder vorwärts die anderen bestimmen muß.

Genau so sieht die Befehlsfolge aus, in die der Interpreter eine rekursive Kontrollstruktur verwandelt. Trivialerweise kommt man an der linear-sequentiellen Arbeitsweise des Computers doch nicht vorbei, wenn man die Rekursions-Technik durchschauen will. Außerdem ist immer noch eine Iteration involviert; diese wird aber nicht als Schleife abgearbeitet, sondern intern ist ja jeder Befehl extra aufgeführt, was übrigens bei einer großen Zahl von Aufrufen zu einem immensen Speicherbedarf führt. Die endständige Rekursion (LL; "last line"; der Selbstaufruf der Prozedur erfolgt als ihr letzter Befehl) erfordert allerdings gar nicht die rekursive Auffassung in diesem Sinn; denn bei ihr werden ja alle Befehle in der Prozedur direkt durchlaufen, ehe diese, i.a. mit anderen Parametern, neu aufgerufen wird. Man braucht sich also keinen zusätzlichen Aufbau einer Befehlssequenz mit anschließender Abarbeitung vorzustellen, sondern für eine solche Prozedur ist die iterative Auffassung angemessen.

Es ist jedoch nicht alles endständige Rekursion, was so aussieht: sie liegt z.B. immer dann nicht vor, wenn in der Anweisung, in der der Selbstaufruf der Prozedur erfolgt, nach diesem noch eine weitere Operation durchgeführt wird, z.B. die Multiplikation in 'RÜCKGABE :N * FAK :N-1' bei der Berechnung der Fakultäten mit einer Prozedur 'FAK'.

Es wird immer wieder berichtet, daß Lernende (zumindest) mit der (endständigen) Rekursion umgehen können (Hanninger 1982:35, Lavallade 1985:59, Treadaway 1985:47 u.a.), während Kurland/Pea (1983, nach Haussmann 1985:148) bei ihren Untersuchungen mit 8- bis 12-Jährigen zu dem Ergebnis kommen, daß diese das Rekursionsprinzip nicht durchschauen, auch wenn sie es benutzen. Und von Erfolgen bei nicht-endständiger Rekursion hört man gar nichts. Diese Befunde sprechen eher für die Kraft der iterativen Auffassung, zumal Anzai/Uesato (1982; nach Haussmann 1985:147) feststellen, daß Lernende rekursive Formulierungen besser verstehen, wenn sie mit iterativen Strukturen vertraut sind. Insgesamt halte ich die statistische Basis jedoch für viel zu schmal, und es müßte überprüft werden, was die jeweiligen Autoren überhaupt unter Erfolg bzw. Mißerfolg verstehen. Aber was spricht eigentlich nach dieser Schwierigkeitsanalyse und den empirischen 'Befunden' dagegen, in der allgemeinbildenden Schule, wenn überhaupt, mit einer iterativen Programmiersprache zu arbeiten, womit mathematischen und informatischen Ansprüchen an den Unterricht m.E.

vollauf Genüge getan wäre, oder, wenn man unbedingt essentielle Rekursion behandeln will, diese erst später einsetzen läßt (dagegen: Löthe 1985:198)?

Logo-Anhänger, auch gemäßigtere, befürchten - mit Recht -, daß der Übergang zu ihrer Programmiersprache dann häufig genug ganz entfallen würde. In der Tat: mit fortschreitender Schulzeit werden die Inhalte, die das rekursive Programmieren nahelegen, wegen ihrer Austauschbarkeit, Isoliertheit und Unverbindlichkeit bei Lehrern und Schülern zunehmend fragwürdig. Im Bereich der Grundschule lassen sich solche Inhalte etwa mit dem Ziel des Erwerbs kognitiver Strategien leichter rechtfertigen. Etwas anderes wäre es, wenn die kulturelle Bedeutung der Mathematik und der Informatik als Hauptquellen von mit Hilfe von Logo-Programmen zu lösenden Probleme erheblich gesteigert würde. Diese Idylle einer Mathematik und Informatik treibenden Jugend halte ich für eine irreale Projektion der Vorlieben von manchen Mathematikdidaktikern und 'Computer-im-Unterricht'-Anhängern. Aber sie wäre eine notwendige Voraussetzung für ein gewisses Niveau, auf dem sich Unterschiede in Programmier-Stilen überhaupt erst auf die Inhalte (Auswahl, Art der Behandlung ohne Vorgaben durch den Lehrer!) oder gar Denkstile spürbar auswirken können.

So sehen Fischer (1977:73) und Hoppe (1984:116f) die Rekursion als Spezialfall der allgemeinen Strategie, ein Problem auf einfachere Teilprobleme zurückzuführen und diese zu lösen (Reduktions-Strategie), wobei sie zwar akzeptieren, daß das Problem als solches nach dem Rekursions-Schritt dasselbe ist, sie es aber einfacher nennen, weil es der Abbruch-Bedingung näher ist. Tatsächlich spielt die Einfachheit des Teilproblems aber keine Rolle, da es ja gar nicht gelöst, sondern im Gegenteil wohlweislich als "black box" (Hoppe 1984:117) behandelt wird. Vielmehr ist es die Beziehung des Teilproblems zum Ausgangsproblem, die die Lösung im wesentlichen liefert. Nicht Problem-Reduktion, sondern Einführung einer Relation mit Hilfe des Rekursions-Schritts ist die Strategie, vergleichbar mit folgender Herleitung der Formel für die endliche geometrische Reihe: Gegeben ist die Reihe $1+q+...+q^n$ mit $q\neq1$, $n\in\mathbb{N}$; man multipliziere sie mit q und bilde die Differenz zwischen der neuen und der alten Reihe: $(q-1)(1+q+...+q^n) = q+q^2+...+q^{n+1}-1-q-...-q^n = q^{n+1}-1$; woraus sich die gesuchte Formel $1+q+...+q^n = (q^{n+1}-1)/(q-1)$ ergibt.

Also: Es trifft zwar zu, daß mit jedem Rekursions-Schritt das Ende der Ausrechnung näher rückt, aber die Ausführung eines Algorithmus, auf die sich Fischer und Hoppe da beziehen, ist doch das Rechenhandwerk, das (evtl. durch den

Computer) nach der prinzipiellen Problemlösung ausgeübt wird. Für das Problemlösen selbst gilt aber: Im Vergleich zu den Schwierigkeiten bei der Reduktions-Strategie, die ich keineswegs verharmlosen will (das Finden und/oder Lösen von Teilproblemen eines Problems ist häufig genug selbst sehr problemhaltig), stellt die Rekursions-Strategie prinzipiell höhere Anforderungen. Gegenüber der Struktur der Teilprobleme hat die rekursive Relation eher externen Charakter, und sie anzugeben liefert die Problemlösung auf der Meta-Ebene. Die Barrieren, die sich vor diesem Verständnis der Problemlösung auftun, sind jedem Oberstufenlehrer und jedem Mathematikdidaktiker an der Hochschule bekannt, der schon einmal versucht hat, das Prinzip der vollständigen Induktion zu unterrichten.

Die Logo-Philosophie sieht die Logo-Rekursion nicht nur als Programmier-Technik und Problemlöse-Strategie, sie schreibt ihr auch Verwurzelung in der Lebenswelt zu: So benutzt Papert die Scherzaufgabe, was man sich wohl als zweiten Wunsch aussucht, wenn man zwei Wünsche frei hat: Natürlich zwei weitere Wünsche (Papert 1980:74). Das ist eine Rekursion ohne Abbruchbedingung, und man darf sich nicht als ersten von jeweils zwei Wünschen zwei weitere wünschen, weil der Wunsch-Gewährer dann nie mit dem Aufbau der Folge offener Wünsche fertig wird, jedenfalls wenn er nicht zwei Wünsche zugleich bearbeiten kann (Hoyles/Noss 1985:174). Andere Bezüge zum Alltagsleben der Kinder: Irgendwie jede Wiederholung menschlicher Handlung und viele Beispiele in Liedern, Gedichten und Geschichten (Lavallade 1985:59). In der Tat: Hier zeigt sich die lebensweltliche Verankerung der universellen Idee der Iteration, aber beileibe nicht in der Spezialform der Rekursions-Technik.

Für Hoppe (1984:104) ist - (für mich) nicht nachvollziehbar - die rekursive Formulierung (eines Programms, das eine Spirale zeichnet) anschaulicher als die iterative. Man kann jedoch seiner These (Hoppe 1984:137) einschränkend zustimmen, daß "die rekursive Beschreibung geeignet" ist, gewisse "Algorithmen und informatische Verfahren leichter durchschaubar, eleganter und kürzer darstellbar zu machen." Allerdings kommt dieser Vorzug m.E. erst an einer späten Stelle des Curriculums der allgemeinbildenden Schule mit weit fortgeschrittener Fähigkeit zum Formalisieren, wenn überhaupt, zum Tragen und ist mit den oben diskutierten Schwierigkeiten verbunden, so daß ich die didaktische Eignung der Logo-Rekursion auf den ganzen Bereich der allgemeinbildenden Schule eher verneine.

Literatur:

Anzai, Y. u. Y. Uesato: Learning recursive procedures by middle-school children. In: Proceedings of the Fourth Annual Conference of the Cognitive Science Society, Ann Arbour, MI 1982

Fischer, G.: Das Lösen komplexer Problemaufgaben durch naive Benutzer mit Hilfe des interaktiven Programmierens. Dissertation Hamburg 1977

Hanninger, J.: Ein Unterrichtsversuch zu geometrischen Erfahrungen in der Grundschule mit Computerhilfe. Bericht Nr.18 aus dem Fach Mathematik, PH Eßlingen (1982)

Haussmann, K.: Iteratives und rekursives Denken beim Lösen mathematischer Probleme. In: Beiträge zum Mathematikunterricht 1985, Bad Salzdetfurth: Franzbecker 1985, S.146-149

Hoppe, H.U.: Logo im Mathematikunterricht - ein Beitrag zur Didaktik des interaktiven Programmierens. Vaterstetten: IWT-Verlag 1984

Hoyles C. u. R. Noss: Proceedings of the Logo and Mathematics Education Conference 1985, London: University 1985

Kurland, D.M. u. R.D. Pea: Children's Mental Models of Recursive LOGO Programs. In: Proceedings of the Fifth Annual Conference of the Cognitive Science Society, Rochester N.Y. 1983

Lavallade, D.: In search of recursion. In: Hoyles/Noss 1985, S.57-60

Löthe, H.: Mathematik, Informatik, Computeranwendungen - Probleme und Chancen einer Integration. In: Beiträge zum Mathematikunterricht 1985, Bad Salzdetfurth: Franzbecker 1985, S.195-198

Oberschelp, W.: Zum Verhältnis von Mathematik, Informatik und Philosophie. In: Schriftenreihe des IDM 16, 35-61 (1977)

Papert, S.: Mindstorms. Children, Computers, and Powerful Ideas. New York: Basic Books 1980 (Dt. Basel: Birkhäuser 1982)

Schreiber, A.: Bemerkungen zur Rolle universeller Ideen im mathematischen Denken. In: mathematica didactica 5, 65-76 (1983)

Treadaway, M.: Logo-Developments in Suffolk. In: Hoyles/Noss 1985, S.45-54

Ziegenbalg, J.: Programmiersprachen als Träger von Grundideen der Informatik. In: Der mathematische und naturwissenschaftliche Unterricht 37, 404-415 (1984)