ZUR ENTWICKLUNG ITERATIVER UND REKURSIVER STRUKTUREN

KRISTINA HAUSSMANN UND MATTHIAS REISS

Einleitung

Programmieren lernen bedeutet zu einem wesentlichen Teil, die Fähigkeit für einem sachgerechten Umgang mit Kontrollstrukturen erwerben. Jedes mit dem Computer bearbeitbare Problem kann nämlich als Programm geschrieben werden mit Hilfe von *Sequenzen* einzelner Schritte, durch die Definition *bedingter Handlungen*, die nur unter gewissen Voraussetzungen ausgeführt werden, und über die Zusammenfassung dieser Grundelemente in Form von *Wiederholungen*. Diese drei Typen von Kontrollstrukturen, also die Sequenz, die Bedingung und die Wiederholung, sind notwendig und reichen auch gleichzeitig aus, um jede Problemlösung als Computerprogramm darzustellen. Die Implementationen von Sequenzen und Bedingungen in den verschiedenen Programmiersprachen unterscheiden sich mehr oder weniger nur im Grad des Komforts, der für den Programmierer bereigestellt wird. Bei der Wiederholung hingegen gibt es zwei strukturell verschiedene Arten der Realisierung. Diese beiden Formen sind die Iteration und die Rekursion.[1]

Benutzt man die Iteration als Kontrollstruktur für eine Wiederholung, so bedeutet das, daß ein Prozeß in gleicher Art und Weise mehrmals hintereinander ausgeführt wird. Es kann dabei explizit vorgegeben sein, wie oft eine Handlung zu wiederholen ist. Man spricht dann von einer gezählten Wiederholung. Es kann aber auch angegeben sein, daß eine Handlung so oft wiederholt wird, bis eine bestimmte Situation erreicht ist. In beiden

[1]Die sprachliche Regelung ist hier nicht einheitlich. Wir wählen Wiederholung als Oberbegriff über die beiden Begriffe Iteration und Rekursion. Häufig werden auch Iteration und Wiederholung gleichgesetzt und von der Rekursion als Verschachtelung abgesetzt.

Fällen können sich Variablen im Laufe des Prozesses ändern, denn das ist bei einer Iteration selbstverständlich möglich.

Entscheidend für die Kennzeichnung eines Prozesses als iterativ ist vielmehr zum einen die Wiederholung selbst und zum anderen die Tatsache, daß ein einzelner Schritt mit dem Durchlaufen der Wiederholung abgeschlossen ist und ein endgültiges Ergebnis hat. Es gibt eventuell Zwischenergebnisse, die sofort weiterverarbeitet werden, es gibt aber keine Zwischenergebnisse, die in irgendeiner Form gespeichert werden müssen, weil zu einem späteren Zeitpunkt auf sie zurückgegriffen wird.

Die Rekursion als Mittel der Realisierung von Wiederholungen in einem Programm unterscheidet sich von der Iteration insbesondere in diesem letztgenannten Punkt. Bei einer rekursiven Betrachtungsweise wird ein Problem nicht direkt gelöst, sondern auf ein einfacheres strukturgleiches Problem zurückgeführt. Die Lösung dieses einfacheren Problems geht in die Lösung des eigentlichen Problems ein. Falls auch das einfachere Problem nicht direkt lösbar ist, setzt man den Gedankengang in gleicher Weise fort. Wiederum wird ein strukturgleiches, noch einfacheres Problem gesucht, auf das die momentan zu lösende Aufgabe reduziert werden kann. Es ergibt sich so eine Folge von Teilproblemen, von denen jedes etwas einfacher ist als sein Vorgänger. Es gibt in dieser Kette ein einfachstes Problem, das eine offensichtliche, direkte Lösung hat. Insbesondere wird auf diese Weise eine Hierarchie von Teilproblemen aufgebaut. Die einzelnen Schritte stehen nicht gleichberechtigt nebeneinander, sondern sind in verschiedenen Ebenen angeordnet. Durch Backtracking vom trivialen Fall auf der untersten Ebene bis zurück zur höchsten Ebene bekommt man eine Lösung des eigentlichen Problems. Dabei ist ein einzelner Schritt in der Problemlösung nicht zu dem Zeitpunkt erledigt, in dem er ausgeführt ist. Vielmehr bedeutet die Verwendung der Rekursion das Arbeiten in einer Hierarchie von Einzelproblemen und die Übergabe von Werten zwischen den Ebenen (REISS, 1984).

In den verschiedenen Programmiersprachen sind Wiederholungen sehr unterschiedlich realisiert. Es gibt Sprachen wie FORTRAN oder ältere Versionen von BASIC, die nur iterative Kontrollstrukturen kennen. Viele neuere Sprachen verfügen sowohl über iterative als auch über rekursive Kontrollstrukturen. Beispiele sind etwa PASCAL und LISP, zwei Sprachen, die sich ansonsten sehr stark voneinander unterscheiden. Es gibt aber auch Sprachen wie PROLOG, in denen Wiederholungen immer über einen rekursiven Aufruf realisiert werden müssen. Daraus ergeben sich für das Erstellen eines Programmes keine

prinzipiellen Schwierigkeiten. Vom mathematischen Standpunkt aus betrachtet sind Iteration und Rekursion äquivalent. Eine Problemlöung, die iterativ formuliert ist, kann genauso rekursiv formuliert werden und umgekehrt. Betrachtet man allerdings Iteration und Rekursion als Problemlösemethoden, dann kann die eine oder andere Art der Formulierung für ein Problem, zumindest subjektiv vom Problemlösenden aus gesehen, angemessener sein. Beide Problemlösemethoden sind entsprechend mit verschiedenen Denkmodi verbunden, die wir als iteratives und rekursives Denken bezeichnen (HAUSSMANN & REISS, 1989).
Wir wollen im folgenden Beispiele aus zwei verschiedenen Problemlösebereichen für diese beiden unterschiedlichen Denkmodi geben. Dabei verwenden wir zum einen von Schülern erstellte LOGO-Programme und zum anderen ihre Lösungen für das Problem des TURM VON HANOI.

Rekursion und rekursives Denken

Der Begriff der Rekursion und die kognitive Repräsentation rekursiver Prozesse sind durch die Bedeutung von Programmiersprachen in letzter Zeit zum Gegenstand von Untersuchungen geworden (SPOHRER, SOLOWAY & POPE, 1985; MÖBUS, SCHRÖDER & COLONIUS, 1986; VORBERG, GRÜNER, HAHN, HEIM, SCHULZE & WAGNER, 1986; SCHMALHOFER & KÜHN, 1986; WALOSZEK, WEBER & WENDER, 1986). Die Rekursion ist ein ursprünglich mathematisches Konzept, das Eingang in die Informatik gefunden hat. In beiden Disziplinen spielt sie eine wichtige Rolle. Die Verwendung rekursiver Kontrollstrukturen oder auch nur ein elementares Verständnis dieser Strukturen bereitet allerdings sowohl jugendlichen als auch erwachsenen Programmieranfängern erhebiche Schwierigkeiten (KURLAND & PEA, 1983; HAUSSMANN, 1985; HAUSSMANN, 1986), die nicht durch die schematische Vorgabe einzelner Schritte zu lösen sind. Es ist daher wichtig, die kognitiven Prozesse zu untersuchen, die stattfinden, bevor die rekursive Problemlösung erreicht ist.
Es gibt trotz der Bedeutung der Rekursion keine Definition des Begriffs, die allgemein akzeptiert wäre (HAUSSMANN & REISS, 1986). Was unter Rekursion zu verstehen ist, wird in sehr unterschiedlicher Art und Weise gesehen, wobei sich einzelne Definitionen vor allem darin unterscheiden, in welchem Kontext sie stehen und wie eng oder weit sie gefaßt sind (BARFURTH, 1987). In der Mathematik sind etwa rekursive Definitionen von

besonderem Interesse, in der Informatik steht die Betrachtung und Formulierung rekursiver Algorithmen im Vordergrund.
Eine sehr allgemeine Auffassung der Rekursion findet man bei HOFSTADTER (1985). Er bezeichnet damit Verschachtelungen und Varianten der Verschachtelung. Rekursiv in diesem Sinne sind etwa Geschichten innerhalb von Geschichten, rekursiv sind Filme innerhalb von Filmen, rekursiv ist auch die russische Puppe Matrioschka, in der sich kleinere Kopien ihrer selbst finden. Insbesondere ist Rekursion ein Alltagsphänomen, mit dem man intuitiv Erfahrungen sammelt. Das Wesen der Rekursion liegt darin, daß man ein Objekt nicht explizit, sondern durch einfachere Versionen seiner selbst beschreibt (HOFSTADTER, 1985, S.165). Als wesentliches Charakteristikum des rekursiven Prozesses stellt HOFSTADTER das Arbeiten auf verschiedenen Ebenen heraus. Ein Hauptbegriff ist dabei der Begriff des Stapels. Man hat auf einen Stapel nur Zugriff von einer Richtung, man kann also oben etwas wegnehmen oder aber etwas hinzufügen. Soll ein bestimmtes Element aus dem Stapel genommen werden, so müssen eventuelle Hindernisse vorher eines nach dem anderen abgeräumt werden. In ähnlicher Weise werden bei einem rekursiven Prozeß Stapel von Teilaufgaben aufgebaut, deren Bearbeitung jeweils zugunsten einer anderen Teilaufgabe verschoben wurde. Es wird auf verschiedenen Ebenen gearbeitet, zwischen denen nicht beliebig gewechselt werden kann. Von einer bestimmten Ebene kommt man auf direktem Wege nur in die folgende oder die vorhergehende Ebene. Dieses Verständnis der Rekursion wird übrigens auch von PAPERT (1980) geteilt.
Viele Definitionen der Rekursion lassen deutlich erkennen, daß sie aus der Informatik kommen. Das trifft auch für die Definition von ROBERTS (1986) zu, der rekursive Algorithmen betrachtet und den Begriff damit in den Kontext des Problemlösens stellt. Für ihn ist Rekursion "the process of solving a large problem by reducing it to one or more subproblems which are (1) identical in structure to the original problem and (2) somewhat simpler to solve" (ROBERTS, 1986, S. 1). Bei HOFSTADTER sind auch der unendliche Prozeß und vor allem das auch von PAPERT (1980) so betonte Spielen mit der Unendlichkeit prinzipiell möglich. Bei ROBERTS hingegen ist das Ziel eine Lösung des Problems und nicht nur seine sukzessive Vereinfachung.
In vielen Fällen wird aber von Rekursion erst dann gesprochen, wenn die folgenden drei hauptsächlichen Merkmale in einem Prozeß erfüllt sind :

1. Ein Problem wird auf ein strukturgleiches, einfacheres Problem reduziert.
2. Es entsteht eine Folge solcher einfacheren Probleme, von denen eines identfiziert werden kann, das unmit telbar eine Lösung hat.
3. Durch stufenweises Zurückgehen werden die Lösungen einer Ebene erzeugt und in der jeweils höheren Ebene verwendet.

Da die rekursive Lösung eines Problems immer nur eine von mehreren möglichen ist, kann rekursives Denken nicht über eine Aufgabe, sondern nur über den Lösungsweg beschrieben werden. Man kann davon ausgehen, daß eine Person rekursiv denken, wenn in einer Problemlösung die oben angegebenen drei Schritte zu identifizieren sind, wenn also ein Problem reduziert wird auf strukturgleiche Unterprobleme, ein einfachster Fall identifiziert und direkt gelöst wird und die Lösung der jeweils einfacheren Fälle in die Lösung der Fälle auf einer höheren Ebene eingeht. Doch spielen diese Schritte nicht nur in ihrer Gesamtheit, sondern auch einzeln eine Rolle. Sie sind jeweils für sich genommen Indikatoren für ein Arbeiten in Richtung auf eine rekursive Lösung. Rekursives Denken hat hingegen nichts damit zu tun, ob eine Person ganz bewußt Lösungsschritte in einer maschinenähnlichen Art und Weise überlegt. Es geht hier insbesondere auch nicht darum, einzelne Stufen des rekursiven Prozesses explizit zu antizipieren oder zu verfolgen. Rekursives Denken umfaßt vielmehr das Erkennen der grundlegenden Struktur eines Problems und die adäquate Arbeit mit seinen strukturellen Gegebenheiten.

Programmieren und rekursives Denken

Die Identifizierung und Beschreibung der Stufen und Vorstufen beim Verständnis der Rekursion war das Ziel einer Untersuchung mit 25 Schülern der Klassen 7 und 8 verschiedener Schularten im Alter zwischen 12 und 15 Jahren. Die Schüler nahmen knapp neun Monate an einer freiwilligen Arbeitsgemeinschaft teil, in der sie Grundzüge des Programmierens in der rekursiven Programmiersprache LOGO lernten. LOGO ist eine Sprache, die zu dem Zweck entwickelt wurde, Kindern den Zugang zum Computer zu ermöglichen. Wir haben uns für die Arbeit mit LOGO entschieden, da diese Programmiersprache durch die Implementation einer Graphik-Komponente, der sogenannten

Schildkröten-Graphik, erlaubt, recht subtile Programmideen auf einfache und auch für Schüler verständliche Art und Weise zu realisieren. Insbesondere gibt es in LOGO die Möglichkeit, sowohl iterative als auch rekursive Algorithmen zu implementieren. Wir vermuten, daß durch die Arbeit mit dieser Programmiersprache ein Bindeglied geschaffen werden kann zwischen dem eher intuitiven Rekursionsbegriff, wie er im Alltag verwendet wird, und rekursiven Strukturen, wie sie in der Mathematik vorkommen. Insbesondere gilt aber auch, daß das Lernen einer Programmiersprache eine komplexe kognitive Tätigkeit ist. Eine Beobachtung dieses Prozesses scheint Verallgemeinerungen zuzulassen (ANDERSON, FARRELL & SAUERS, 1984).

Bei der Einführung in LOGO wurden Probleme aus der Igel-Graphik genauso wie Anwendungsaufgaben zu den Rechenmöglichkeiten und Beispiele für den Umgang mit Listen und Wörtern berücksichtigt. Die Schüler bekamen von uns kleinere und größere Programmieraufgaben vorgegeben, sie wurden aber auch immer wieder ermutigt, sich stattdessen oder zusätzlich eigene Probleme zu stellen. Das Hauptgewicht unseres Interesses lag nicht bei bestimmten Aufgaben, sondern vielmehr darauf, wie die Schüler sie lösten und welche Kontrollstrukturen Verwendung fanden. Während des LOGO-Kurses machten wir zweimal Einzelinterviews mit allen Schülern der Gruppe. Das erste dieser Interviews wurde nach etwa zehn Unterrichtsstunden durchgeführt. Zu diesem Zeitpunkt hatten die Schüler Grundkenntnisse der Syntax von LOGO erworben. Sie waren mit den wichtigsten Kommandos zur Schildkröten-Graphik vertraut und hatten auch bereits kleinere Programme geschrieben. Dabei waren vor allem ein sequentieller Programmablauf und die wiederholte Ausführung bestimmter Programmschritte mit Hilfe des REPEAT-Kommandos oder über einen rekursiven Aufruf verwendet worden. Wir meinen, daß nach einer solchen kurzen Einführung der Schüler in eine Programmiersprache bei einem Interview eher ein intuitives Wissen aufgedeckt werden kann, als ein systematisches Wissen, das im Verlauf des Kurses erworben wurde. Ein zweites Mal wurden die Schüler im Anschluß an den Kurs interviewt. Zu diesem Zeitpunkt war selbstverständlich der Wortschatz in der Programmiersprache größer geworden. Die Schüler waren aber auch intensiver mit Techniken vertraut gemacht worden, wie Programme geschrieben werden können. Insbesondere hatten sie mit rekursiven Programmen verschiedenen Schwierigkeitsgrades intensiv gearbeitet.

In den Interviews wurden fünf Teilaufgaben gestellt. Davon bezogen sich vier auf die Grundprobleme des Programmierens, nämlich die Planung eines Programms, die Inter-

pretation eines vorliegenden Programms, das Erstellen eines Programms bei vorgegebenem Problem und die Fehlersuche in einem Programm (vgl. PEA & KURLAND, 1983). Eine weitere Teilaufgabe war die Lösung des Problems TURM VON HANOI, also einer prinzipiell rekursiven Problemstellung unabhängig vom Programmierwissen. Über diese Einzelinterviews hinaus protokollierten wir auch die Arbeit der Schüler in den verschiedenen LOGO-Sitzungen. Teilweise geschah das mit Hilfe von Dribble-Files, bei denen jede mit Return bestätigte Eingabezeile in einem externen Speicher festgehalten wird, teilweise analysierten wir aber auch nur die von den Schülern während der Sitzungen erstellten Programme.

Programmbeispiele aus einer LOGO-Arbeitsgemeinschaft

Im Zusammenhang mit dem Programmieren wird häufig eine einfache Definition des Rekursionsbegriffs benutzt. Man spricht von einem rekursiven Algorithmus, wenn sich der Algorithmus während seines Ablaufs selbst aufruft (BAUER & GOOS, 1982[3]; DÖRFLER, 1984). Es gibt verschiedene Formen, die sich in der Komplexität unterschieden, wobei die Grundstufe die von BAUER & GOOS *repetitive Rekursion* genannte Form ist. Dabei gibt es einen rekursiven Aufruf in der letzten Programmzeile, doch werden keine Werte übergeben, so daß mit dem Erreichen einer neuen Stufe die vorhergehende Stufe im Grunde abgeschlossen. Es handelt sich um eine Rekursion, die einer Iteration sehr ähnlich ist, sich aber in der Notation davon unterscheidet. Durch den rekursiven Aufruf in der letzten Programmzeile wird mit jedem neuen Schritt jedesmal die Aktionsebene gewechselt. Doch gehen keine Werte in einem Rücklauf in die Gesamtlösung ein. Man kann sich vorstellen, daß das Programm beim Erreichen der untersten Ebene abbricht. Tatsächlich ist diese Art der Rekursion in vielen Sprachen, unter anderem auch in einigen LOGO-Dialekten, maschinenintern als Iteration realisiert.
Nach dem Prinzip der repetetiven Rekursion wird etwa die Bestimmung des größten gemeinsamen Teilers zweier Zahlen mit Hilfe des Euklidischen Algorithmus ausgeführt (vgl. HAUSSMANN, 1987, S. 143). Es ist ggT(a,b) = ggT(b,amodb) für b > 0 und ggT(a,b) = a für b = 0. Der größte gemeinsame Teiler wird bei dieser Bestimmungsmethode sukzessive vereinfacht bis zu einem trivialen Fall. Dann allerdings ist die Lösung direkt ablesbar. Komplexer sind hingegen Formen, bei denen im Rücklauf von der untersten Ebene auf die ursprüngliche Ebene Werte übergeben werden. Ein Beispiel ist

die Berechnung von n! für eine natürliche Zahl n (vgl. HAUSSMANN, 1987, S. 146). Es gilt n! = n*(n-1)! für n > 1 und 1! = 1. Eine direkte Berrechnung ist hier nicht möglich, sondern nach dem Erreichen des trivialen Falls n = 1 muß man zum Ausgangspunkt zurückgehen. Während bei der Bestimmung des größten gemeinsamen Teilers das Problem auf ein strukturgleiches reduziert wird und ein einfachster Fall angegeben werden kann, geht bei der Berechnung von n! das Ergebnis jeder Stufe in die Gesamtlösung ein. Zu beiden Aufgaben kann man LOGO-Programme schreiben, die einen rekursiven Aufruf in der letzten Zeile haben. Doch verbergen sich dahinter unterschiedliche Schwierigkeiten bei der Programmerstellung (vgl. auch DUPUIS, EGRET & GUIN, 1985).

Die folgenden Aufgaben, die im LOGO-Kurs den Schülern zur Bearbeitung vorgegeben wurden, haben in rekursiver Notation ganz entsprechende Lösungen. Die erste Aufgabe entspricht der Berechnung des größten gemeinsamen Teilers. Eine mögliche LOGO-Prozedur könnte einen Selbstaufruf in der letzten Zeile enthalten und auf der untersten Ausführungsebene abbrechen. Bei der zweiten und dritten Aufgabe hingegen sind Lösungen denkbar, die strukturell der Bestimmung von n! gleichen und somit alle Merkmale eines rekursiven Prozesses aufweisen. Doch sind bei der Lösung aller Aufgaben auch Prozeduren möglich, die den in LOGO implementierten REPEAT-Befehl verwenden.

Aufgabe 1

Es soll eine Prozedur TRICHTER1 geschrieben werden, die ein Wort als Eingabe hat. Der Aufruf von TRICHTER1 "ANTON soll den folgenden Ausdruck auf dem Bildschirm erzeugen:

```
ANTON
ANTO
ANT
AN
A
```

Aufgabe 2

Gesucht ist eine Prozedur TRICHTER2, die den Trichter aus Aufgabe 1 von unten nach oben auf den Bildschirm bringt.

```
TRICHTER2 "OSKAR
```

R
AR
KAR
SKAR
OSKAR

Aufgabe 3

Schreibe eine Prozedur, die ein Eingabewort umkehrt, die also aus dem REGAL ein LAGER macht und aus dem ESEL eine LESE.

Bei der Umsetzung dieser Probleme in Programme sind Entscheidungen über die Programmstruktur und über den Umgang mit Variablen zu treffen. Dabei sind hinsichtlich der Struktur sowohl rein sequentielle Programme als auch Programme mit iterativen oder rekursiven Kontrollstrukturen denkbar. Sicher wird schon von einem Programmiernovizen erkannt, daß die einzelnen Ausgabezeilen über prinzipiell ähnliche Programmzeilen realisiert werden können. Doch während sich dieser Sachverhalt bei einem rein sequentiellen Programm nur implizit in der Ähnlichkeit aufeinanderfolgender Zeilen ausdrückt, werden bei der Verwendung von Iteration oder Rekursion die Möglichkeiten einer Sprache explizit benutzt. Teile einer Problemlösung müssen dazu als strukturgleich identifiziert werden. Eine zweite Entscheidung vor dem Erstellen des Programms beinhaltet den Umgang mit Variablen. Es ist entscheidend, welche Variablen eingeführt werden und wie diese Variablen im Verlauf des Programms geändert werden. Welche Lösungen Schüler für die Probleme finden, illustrieren die folgenden Beipiele. Man sieht insbesondere, daß sehr verschiedene Methoden beim Umgang mit Problemen, die rekursiv gelöst werden könnten, Verwendung finden. Die Fallbeispiele lassen erkennen, daß es Teilaspekte und Vorstufen rekursiven Denkens gibt, die im Hinblick auf ein Verständnis rekursiver Strukturen produktiv genutzt werden können.

Klaudia (13)

KLAUDIA ist eine Schülerin der Klasse 8 eines Gymnasiums. Sie geht bei der Lösung der ersten Aufgabe von der Überlegung aus, daß sich bei der Anwendung einer Prozedur TRICHTER1 zwei aufeinanderfolgende Wörter im letzten Buchstaben unterschei-

den. Dieser Buchstabe wird im jeweils folgenden Wort nicht mehr ausgedruckt. Der Druckbefehl in LOGO ist PRINT, mit BUTLAST hat man Zugriff auf ein Wort ohne seinen letzten Buchstaben.
So findet sie sehr schnell die folgende Lösung:

```
TO TRICHTER1 :WORT
        PRINT :WORT
        PRINT BUTLAST :WORT
        PRINT BUTLAST BUTLAST :WORT
        PRINT BUTLAST BUTLAST BUTLAST :WORT
END
```

Diese Lösung ist sequentiell, doch enthält sie auch ohne die explizite Verwendung einer REPEAT-Anweisung ganz offensichtlich ein iteratives Element. Die Schülerin hat erkannt, wie sich die einzelnen Komponenten voneinander unterscheiden und hat dies in recht ähnlichen Zeilen berücksichtigt. Natürlich ist diese Prozedur wenig flexibel. Sie liefert nur dann eine richtige Lösung, wenn das Eingabewort aus genau vier Buchstaben besteht. Die Eingabe des Beispielwortes ANTON führt also nicht zu der gewünschten Ausgabe. Doch diese ausführliche Schreibweise enthält alle wesentlichen Elemente der Lösung und leitet hin zur Idee einer rekursiven Notation. KLAUDIA kommt nach einigen Überlegungen zu folgender Prozedur:

```
TO TRICHTER1 :WORT
        IF :WORT = '' [STOP]
        PRINT :WORT
        TRICHTER1 BUTLAST :WORT
END
```

Bei dieser Lösung wird das Problem auf ein strukturgleiches mit vereinfachter Eingabe zurückgeführt. Außerdem ist ein trivialer Fall angegeben, der offensichtlich lösbar ist. Bei Eingabe eines Wortes ohne Buchstaben wird der STOP-Befehl ausgeführt. STOP bedeutet in LOGO nicht den Abbruch einer Prozedur, sondern die Rückgabe der Kontrolle an die aufrufende Prozedur. Der Befehl zum Abbruch wäre THROW "TOPLEVEL, das ist die sofortige Rückgabe der Kontrolle an die oberste Ebene. Da in diesem speziellen Fall allerdings in der aufrufenden Prozedur keine Befehle mehr zu bearbeiten sind,

ist das Ergebnis auf dem Bildschirm in beiden Fällen gleich. Rückfragen bei KLAUDIA ergeben, daß ihre Interpretation von STOP die des Prozedurabbruchs ist.

Stefan (14)

Auch STEFAN besucht die 8. Klasse eines Gymnasiums. Er gehört zu den Schülern, die vor dem LOGO-Kurs bereits Programmiererfahrungen in BASIC hatten. Dementsprechend geht er recht souverän an die Aufgaben heran. Die erste Aufgabe ist ihm offensichtlich zu einfach. Er bearbeitetet sie nicht und kommt für die zweite Aufgabe sofort zu einer allgemein anwendbaren Lösung.

```
TO TRICHTER2 :WORT
        MAKE "R COUNT :WORT MAKE "A "
        REPEAT :R [MAKE "A WORD LAST :WORT :A
                MAKE "WORT BUTLAST :WORT PRINT :A]
END
```

Im Gegensatz zum ersten Versuch von KLAUDIA ist diese Prozedur allgemeingültig, sie ist also für Wörter beliebiger Länge zu verwenden. Der Aufbau scheint auf den ersten Blick überraschend und etwas unübersichtlich. Zunächst wird Hilfe des LOGO-Grundwortes COUNT gezählt, wie viele Buchstaben die Eingabe enthält. Im Anschluß daran wird mit MAKE eine Hilfsvariable A definiert. Sie nimmt jeweils die Buchstabenfolge auf, die in einer bestimmten Zeile gedruckt werden soll, und wird durch die LOGO-Befehle WORD, LAST und BUTLAST manipuliert. WORD faßt dabei zwei Eingaben zu einem Wort zusammen, LAST erlaubt den Zugriff auf das letzte Zeichen eines Wortes. Entscheidend ist hier die Anweisung REPEAT. Die notwendigen Änderungen der Variablen werden so oft ausgeführt wie durch die Länge des Eingabewortes bestimmt ist.
Auch diese Lösung läßt erkennen, daß die prinzipielle Gleichartigkeit der aufeinanderfolgenden Schritte berücksichtigt wird. Doch sind andere Charakteristika eines rekursiven Prozesses nicht zu erkennen. Das Hauptgewicht liegt auf der Änderung der Eingabevariablen. Diese Änderung wird aber ganz offensichtlich nicht als Vereinfachung angesehen. STEFAN verwendet rekursive Aufrufe auch bei anderen Aufgaben selten. Wenn es möglich ist, mit einer gezählten Wiederholung zu arbeiten, zieht er diese Methode vor. Dies zeigt sich auch bei seiner Lösung der Aufgabe, eine Buchstabenfolge umzukehren.

Sie unterscheidet sich strukturell nicht von der Lösung der TRICHTER-Aufgabe und kann auch als iterativ angesehen werden.

```
TO UMKEHR :WORT
        MAKE "R COUNT :WORT MAKE "A "
        REPEAT :R [MAKE "A WORD :A LAST :WORT
                MAKE "WORT BUTLAST :WORT]
        PRINT :A
END
```

Sebastian (14)

Es gibt Schüler, die mit rekursiven Prozeduren sehr souverän umgehen. Die folgenden Programme von SEBASTIAN sind ein Beleg. Er ist fast 14 Jahre alt und ebenfalls in der 8. Klasse. SEBASTIAN ändert das gegebene Problem leicht ab, so daß beachtet werden muß, daß sich seine Prozedur TRICHTER zwar in der Wirkung, doch nicht in der prinzipiellen Struktur von der in Aufgabe 1 verlangten Prozedur TRICHTER1 unterscheidet.

```
TO TRICHTER :WORT
        PRINT :WORT
        IF :WORT = " [THROW "TOPLEVEL]
        TRICHTER BUTFIRST :WORT
END

TO TRICHTER2 :WORT
        IF :WORT = " [STOP]
        TRICHTER2 BUTLAST :WORT
        PRINT :WORT
END
```

Die Verwendung von THROW "TOPLEVEL in der ersten Prozedur und STOP in der zweiten Prozedur verdeutlicht, daß diesen Lösungen ein elaboriertes Verständnis rekursiver Prozesse und der Programmiersprache LOGO zugrunde liegt. Darüber hinaus ist das zweite Problem mit Hilfe eines eingebetteten rekursiven Aufrufs gelöst, so daß alle Komponenten eines rekursiven Prozesses zu identifizieren sind. Die Prozedur wird durch den Selbstaufruf auf eine strukturell gleiche Prozedur zurückgeführt, mit jedem

neuen Aufruf, also jeder neuen Ebene, wird die Eingabe weniger komplex und es ist ein einfachster Fall identifiziert, da für das leere Wort die Anweisung STOP gegeben ist. In einem Rücklauf durch alle Ebenen druckt das System den jeweils aktuellen Parameter aus. Interessant ist auch SEBASTIANs Behandlung der dritten Aufgabe, da hier die Manipulation der Variablen und der rekursive Aufruf getrennt werden.

```
TO UMKEHR :WORT
        IF :WORT = '' [PRINT [] THROW "TOPLEVEL]
        MAKE "A LAST :WORT
        TYPE :A
        UMKEHR BUTLAST :WORT
END
```

Im Prozeduraufruf selbst wird die Eingabevariable geändert, der Ausdruck wird hingegen über eine mit Hilfe von MAKE definierte Hilfsvariable gesteuert.

Diese Beispiele der Problemlösungen von Schülern spiegeln die unterschiedlichen Arten des Umgangs mit den gegebenen Problemen wider. Im Beispiel von KLAUDIA, die zu einer rekursiven Notation kommt, kann man eine Entwicklung von einer iterativen zu einer rekursiven Betrachtungsweise feststellen. Für sie ist von Anfang an nicht ein bestimmter Teil des Prozesses herausgehoben, sondern sie versucht die strukturelle Ähnlichkeit der Teile für die Lösung zu nutzen. Dieses Verhalten war bei vielen Schülern zu beobachten. Es führte allerdings nicht immer ohne weitere Anleitungen zur Formulierung rekursiver Prozeduren. Die Methode von STEFAN deutet auf eine eindeutig iterative Sichtweise des Problems hin. Das Augenmerk des Problemlösers wird hier auf die Identifizierung und Belegung geeigneter Variablen gerichtet. Jeder Teilprozeß ist gleichberechtigt und gleichrangig mit den anderen. Sebastian hingegen kann genauso eindeutig einer rekursiven Betrachtungsweise zugeordnet werden. Seine Vorgehensweise läßt erkennen, daß die strukturelle Ähnlichkeit der Teilprozesse vorrangig in die Lösung eingeht und das Problem mit Hilfe des rekursiven Aufrufs sukzessive vereinfacht wird.

Der TURM VON HANOI als rekursives Problem

Betrachtet man die Lösungen von Programmieraufgaben, so entsteht leicht der Eindruck, daß nicht immer durch ein Programm auch der Denkstil des Programmieres wiedergegeben wird. Oftmals ist es so, daß ein Programmieranfänger zwar elaborierte Ideen entwickelt, sie aber nicht in einer Programmiersprache formulieren kann. Wird also zu einem Problem kein Programm oder nur ein unzureichendes Programm erstellt, so heißt das noch nicht, daß der Problemlösende keine Struktur einer Lösung erkennt. Insbesondere werden rekursive Strategien von uns als ein Netz ineinander verschachtelter mentaler Operationen begriffen, die sich in der Regel nicht direkt in beobachtbaren Handlungen niederschlagen. Wir haben deshalb als eine weitere Aufgabe für die Schüler den TURM VON HANOI hinzugenommen. Der TURM VON HANOI ist deswegen eine besonders interessante Problemstellung, weil die optimale Lösung eine rekursive Struktur hat und dabei einzelne Schritte des Denkprozesses handlungsorientiert beobachtet werden können. Die Aufgabe ist in der kognitiven Psychologie oft beschrieben worden (ANZAI & SIMON, 1979; ANDERSON, 1983). Sie kann als rekursives Transformationsproblem gekennzeichnet werden, was allerdings nicht heißt, daß jeder einzelne Zug in diesem Problem einem Schritt im rekursiven Denkprozeß entsprechen muß. Es handelt sich um eine Aufgabe, die ohne große technische Hilfsmittel realisiert werden kann. Man braucht eine Anzahl von Scheiben unterschiedlicher Größe und drei Felder, die mindestens den Durchmesser der größten Scheibe haben. Häufig sind auf den Feldern Stäbe angebracht, und in den Scheiben befindet sich jeweils ein Loch, damit jeder Schritt eindeutig vollzogen ist, wenn die Scheibe auf einen Stab gesteckt wurde.

Die Aufgabe der Versuchsperson ist es, alle Scheiben von A nach C zu bewegen und dort denselben Turm aufzubauen. Maßgeblich sind dabei die folgenden Regeln: es darf jeweils lediglich eine Scheibe pro Zug bewegt werden; befinden sich mehrere Scheiben auf einem Feld, so darf nur die oberste bewegt werden; eine Scheibe darf nie auf eine kleinere gelegt werden; Feld B kann als Hilfsfeld dienen.

Alle Zugmöglichkeiten können in einem Diagramm als Suchraum dargestellt werden (vgl. Abb. 1 nach KLIX, 1971). Dabei wird ein freier Zug und der darauf folgende obligatorische Zug zu einem einzigen Knoten im Suchraum zusammengefaßt. Man kommt so zu einem verschachtelten Dreieck, das die rekursive Struktur des Problems zeigt. Der Knoten an der Spitze des Dreiecks steht für die Ausgangsstellung (alle drei Scheiben

liegen noch auf dem linken Feld), der Knoten in der rechten unteren Ecke für die Zielstellung (alle drei Scheiben befinden sich nun auf dem rechten Feld). Der Knoten in der linken unteren Ecke steht für eine Lösung, bei der alle Scheiben auf das mittlere Feld gelegt wurden. Dazwischen befinden sich alle möglichen Scheibenpositionen, die unter korrekter Anwendung der Regeln herauskommen können. Bei n Scheiben sind 3^n Positionen möglich, im Fall n=3 sind das 27 verschiedene Spielstellungen. Die folgende Abbildung (Abb. 1) beschreibt den Suchraum für das Drei-Scheiben-Problem. Dabei bedeutet "3 2 1", daß die größte Scheibe 3 auf Feld A liegt, die mittlere Scheibe 2 auf Feld B und die kleine Scheibe 1 auf Feld C. Ein Strich weist darauf hin, daß sich keine Scheibe auf dem betreffenden Feld befindet.

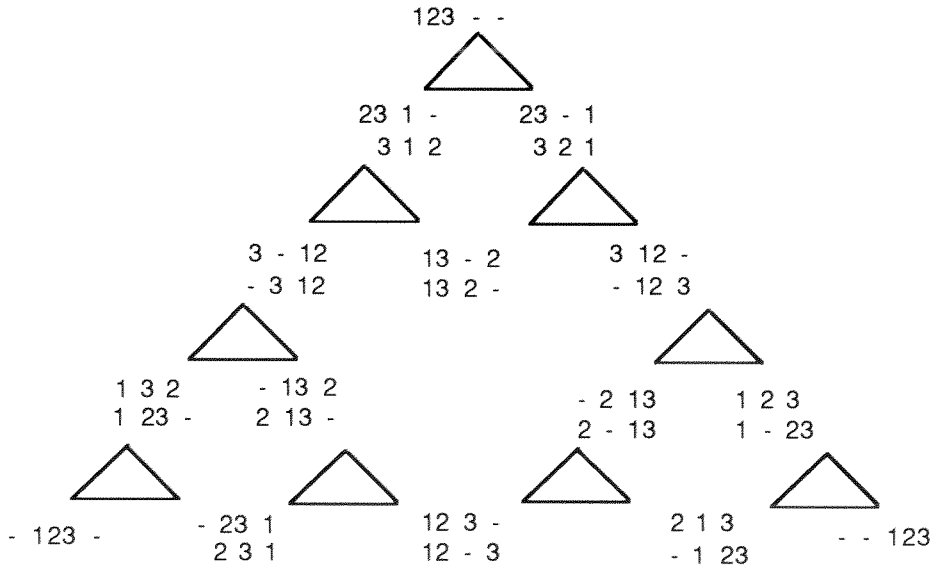

Abbildung 1: Suchraum für den TURM VON HANOI mit drei Scheiben

In diesem Diagramm läßt sich jeder Zug und jede Zugfolge lokalisieren und somit kann man jede Lösung des Problems als einen bestimmten Weg in diesem Diagramm beschreiben. Betrachtet man den optimalen Lösungsweg, also die rechte Kante des großen Dreiecks, so sieht man, daß der wichtigste Zug der ist, bei dem die größte Scheibe von

der Ausgangsposition auf die Zielposition gelegt und damit das Drei-Scheiben-Problem auf zwei Zwei-Scheiben-Probleme reduziert wird. Man befindet sich an diesem Punkt genau auf der Hälfte der rechten Kante des Dreiecks. Dieser Punkt stellt gleichzeitig die rechte untere Ecke eines kleineren Dreiecks und die obere Ecke eines weiteren ebenso großen Dreiecks dar. Bedient man sich der Metapher des Turmbauens, so kann man sagen, daß zweimal ein Zweier-Turm bewegt werden muß, damit man einen Dreier-Turm bauen kann, und daß zusätzlich die größte Scheibe zu bewegen ist. Horizontale oder diagonale Verbindungen zwischen den Positionen sind gleichrangig, wichtig ist nur, ob ein Zug den Abstand zur Zielposition verkleinert (vgl. die Definition der Problemlösegüte bei FUNKE, 1988).

Unser Interesse gilt jedoch nicht nur der optimalen Lösung. Jeder Knoten im Suchraum ist eine gültige Position, und jeder Weg von der oberen Ecke zur rechten unteren Ecke stellt eine Lösung des Problems dar. Berücksichtigt man allerdings bei der Lösung die rekursive Struktur der Aufgabe, so wird man sicher nicht jede Position gleich bewerten, also nicht im gesamten Bereich des Diagramms suchen. Zunächst wird man dann nur Züge wählen, die die Position der unten liegenden größten Scheibe unverändert lassen. Dies wird man so lange fortsetzen, bis sie sich als einzige verbleibende Scheibe auf diesem Feld befindet und das Zielfeld frei ist. Das Teilziel, sie auf das Zielstab zu bewegen, ist somit nur erreichbar, wenn der um eine Scheibe verkleinerte Turm auf dem Hilfsfeld aufgebaut wurde. Bis zu diesem Punkt wird man sich also nur für Züge interessieren, die sich im oberen Teil des Dreiecks bewegen. Umgekehrt ist es nicht sinnvoll, Züge in diesem Teildreieck auszuführen, wenn die größte Scheibe bereits auf ihrer Zielposition liegt. Diese Überlegung läßt sich für das weitere Vorgehen verallgemeinern. Betrachtet man den neuen Zielturm, so kann man auf das Hilfsfeld nur dann gelangen, wenn seine größte Scheibe, das ist die mittlere im Gesamtproblem, vorübergehend auf das andere Feld gelegt und dann die untenliegende Scheibe bewegt wird. Diese weitere Einschränkung des Suchraums bedeutet, daß nur solche Züge sinnvoll sind, die im kleineren oberen Teildreieck des Dreiecks zu lokalisieren sind. Man kommt auf diese Weise zum optimalen Pfad entlang einer der beiden äußeren Kanten des Diagramms. Diese Wege repräsentieren eine Strategie, bei der der Suchraum erfolgreich immer weiter reduziert wird. Eine solche Lösung soll im folgenden als rekursive Lösung des Problems bezeichnet werden.

Ein entsprechender Suchraum läßt sich für jede Anzahl n von Scheiben aufstellen. Dabei kann man die einzelnen Diagramme für den Suchraum bei n und bei n-1 Scheiben ineinander überführen. So läßt sich der Suchraum für das Vier-Scheiben-Problem aus drei Diagrammen für das Drei-Scheiben-Problem zusammensetzen. Im oberen Dreieck findet man alle Züge, die möglich sind, bis die unterste Scheibe auf ein anderes Feld bewegt wird. Die beiden unteren Dreiecke stehen jeweils für die Bewegung des kleineren Drei-Scheiben-Turms vom Hilfsfeld zum Zielfeld B oder C.

Über die Lokalisierung der Züge im Suchraum konnten die Lösungen aller Schüler zu dieser Aufgabe vollständig beschrieben werden. Wir wollen hier exemplarisch die Lösungen zweier Schüler analysieren. Ihre Züge deuten auf typische Strategien hin, die Programmieranfänger bei der Aufgabe des TURM VON HANOI verwendeten. Dabei stand es den Schülern frei, Feld B oder Feld C als Zielposition zu betrachten. Wichtig war lediglich, daß der Turm auf einem anderen als dem Ausgangsfeld aufgebaut wurde.

Tom (13)

TOM besucht die 8. Klasse einer Hauptschule. Sein Engagement für den Computer und das Programmieren ist deutlich ausgeprägt. TOM hat selbst einen Rechner zu Hause und konnte bereits vor Beginn des LOGO-Kurses einfache Probleme in BASIC programmieren. Er kannte einige Kontrollstrukturen so zum Beispiel die gezählte Wiederholung. Am eigenen Computer hatte er die Möglichkeit, sich mit den im Unterricht gestellten Aufgaben zu beschäftigen. Seine Kenntnisse der LOGO-Grundwörter und ihrer Verwendung waren deutlich besser als die der anderen Schüler.

TOM kooperierte zwar mit anderen Schülern, war jedoch immer bestrebt zu dominieren, indem er den Platz an der Tastatur einnahm.

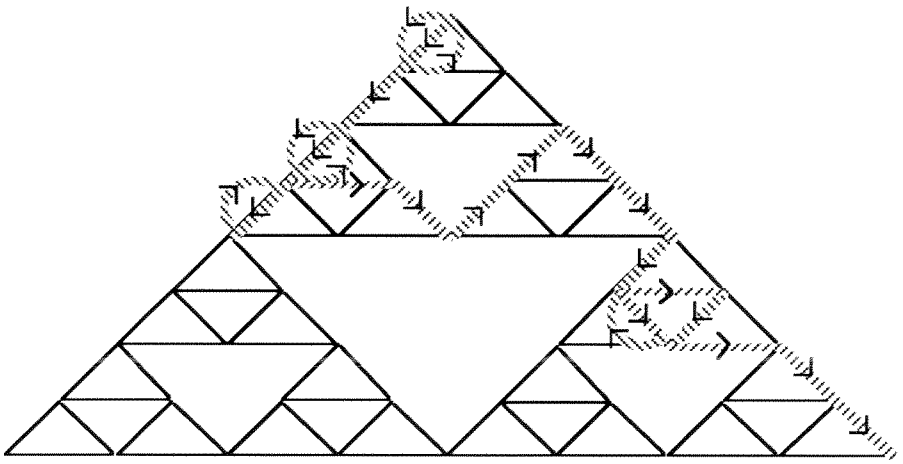

Abbildung 2: Iterative Lösung des Vier-Scheiben-Problems

Abb. 2 zeigt TOMs Lösungsweg für den TURM VON HANOI mit vier Scheiben. Der Suchraum wird im oberen Teil dieses Diagramms nahezu vollständig abgedeckt. TOM bewegt zunächst recht willkürlich die Scheiben hin und her, bis er die größte Scheibe auf das mittlere Feld legen kann. Analysiert man die weiteren Bewegungen, so wird deutlich, daß TOM sich nun mit seinen Zügen in einem kleineren Teildreieck bewegt. Dieses Teildreieck ist eine Untermenge der Züge, bei der lediglich die kleinste und die zweitkleinste Scheibe bewegt werden. Die zweitgrößte bleibt unverändert in ihrer Position auf dem Hilfsfeld.

Nachdem diese auf das Ausgangsfeld der Problemstellung gelegt worden ist, findet TOM sofort den optimalen Weg zur Zielposition.

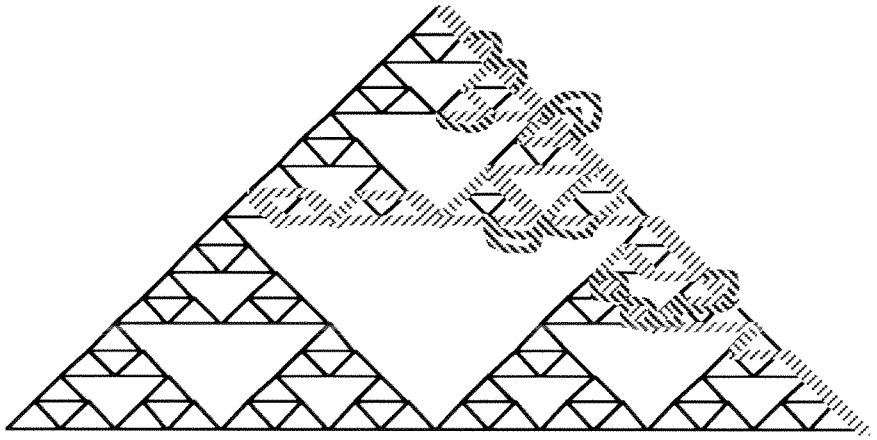

Abbildung 3: Iterative Lösung des Fünf-Scheiben-Problems

Die Analyse des Suchraums zeigt, daß TOM über eine lokale Strategie verfügt. Er verfolgt in jeder Phase der Problemlösung das Teilziel, die größte Scheibe auf den zu errichtenden Stapel zu bringen. Er überträgt diesen Gedanken jedoch nicht auf die dann verbleibenden Scheiben. So ist bei den Zügen, in denen mit diesen Scheiben operiert wird, keine einheitliche Strategie zu erkennen. Die Züge lassen sich innerhalb gewisser Grenzen durch die Methode von Versuch und Irrtum beschreiben. Offensichtlich zu erkennen ist dies im oberen Dreieck, in dem TOM die Scheiben relativ ziellos von einem Feld auf das andere legt. Er konzentriert sich nicht mehr auf sein globales Ziel, den ganzen Turm von einem Feld auf ein anderes zu bewegen. Nachdem er dann die größte Scheibe verschoben hat, setzt er sich ein neues Teilziel. Dieses Teilziel verfolgend macht er nur Züge, die sich im kleineren Dreieck unterhalb des großen oberen Dreiecks bewegen, also Züge, die seine neue größte Scheibe unverändert lassen. Es wäre jedoch falsch, TOMs Züge lediglich als vom Zufall geleitet zu beschreiben. Vermutlich hat er

einzelne Positionen als Schlüsselreize im Langzeitgedächtnis gespeichert, die Initialpositionen für eine ganze Folge von Zügen darstellen (vgl. REISS & HAUSSMANN, 1988). Diese Chunks sind automatisierte mentale Operationen, die den Prozeß der Problemlösung wesentlich schneller ablaufen lassen, aber auch unflexibler gestalten. Die Automatisierung des Denkprozesses läßt sich mit der begrenzten Kapazität des Kurzzeitgedächtnisses erklären. Durch Chunking wird eine ganze Reihe von Einheiten zu einer Einheit zusammengefaßt und so eine überschaubare Anzahl von übergeordneten Einheiten gebildet.

Auch beim zweiten Interview läßt sich diese Strategie bei TOM beobachten. Zwar löst er dieses Mal die Vier-Scheiben-Aufgabe eleganter, verwendet aber bei der wesentlich schwierigeren Fünf-Scheiben-Aufgabe eine vergleichbare Strategie (vgl. Abb. 3). Man kann erkennen, wie TOM in jedem Teildreieck fast den gesamten Suchraum abdeckt. Er verlegt seine gesamten Anstrengungen auf das Teilziel, die unterste Scheibe freizulegen. Die strukturellen Gegebenheiten des Problems im Hinblick auf eine rekursive Lösung bleiben dabei außerhalb seines Blickfeldes.

Jan (14)

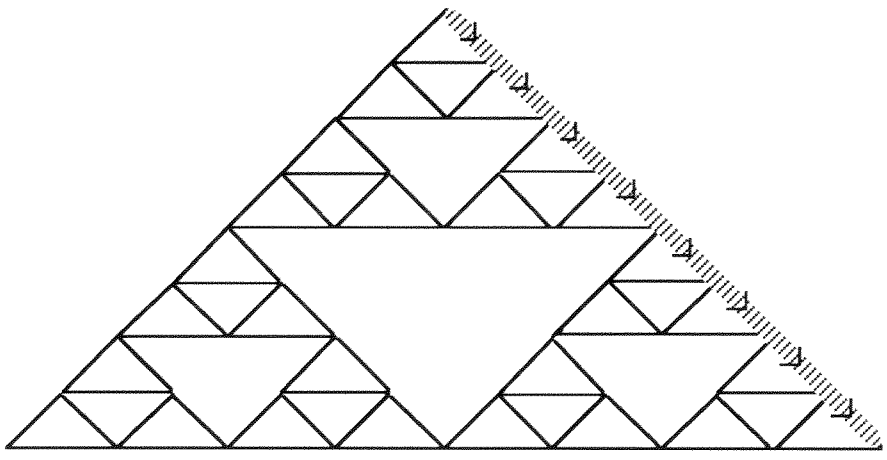

Abbildung 4: Rekursive Lösung des Vier-Scheiben-Problems

JAN besucht zum Zeitpunkt der Untersuchung die 7. Klasse einer Hauptschule. Er hat ein ausgeprägtes Interesse für Computer, aber außerhalb des Unterrichts keine Möglichkeit, einen Rechner zu benutzen. Dennoch zeigt JAN ein außerordentliches Geschick beim Programmieren. Er überblickt recht schnell, wie sich ein Problem in ein entsprechendes LOGO-Programm umsetzen läßt. Auch er beschäftigt sich, genauso wie STEFAN, nicht nur mit den gestellten Aufgaben, sondern ändert und erweitert nach der Lösung einer Aufgabe die Problemstellung, fragt auch gegebenenfalls nach weiteren LOGO-Grundwörtern, die für das Erreichen seiner Ziele erforderlich sind. Auf diese Weise schreibt er im allgemeinen anspruchsvolle Programme, die weit über dem durchschnittlichen Niveau der übrigen Schüler lagen. JAN zieht oft die individuelle Arbeit am Computer der Kooperation mit anderen vor, er hilft ihnen aber bei der Lösung ihrer Aufgaben oder erklärt einzelne Schritte, die zur Problemlösung erforderlich sind.

Abb. 4 zeigt JANs Lösung der Vier-Scheiben-Aufgabe im ersten Interview. Sie verläuft genau auf dem optimalen Pfad. Er findet diese Lösung auch recht schnell. Die einzelnen Scheiben werden rasch bewegt, und er arbeitet zumindest mit einer intuitiven Strategie. Eine kleine Pause ist lediglich zu beobachten, nachdem er die größte Scheibe auf die Zielposition gebracht hat. Es ist anzunehmen, daß er in dieser Situation das Problem neu strukturiert, das heißt, er paßt die Strategie den aktuellen Erfordernissen an und überprüft sie auf ihre Angemessenheit (vgl. COHORS-FRESENBORG, 1985).

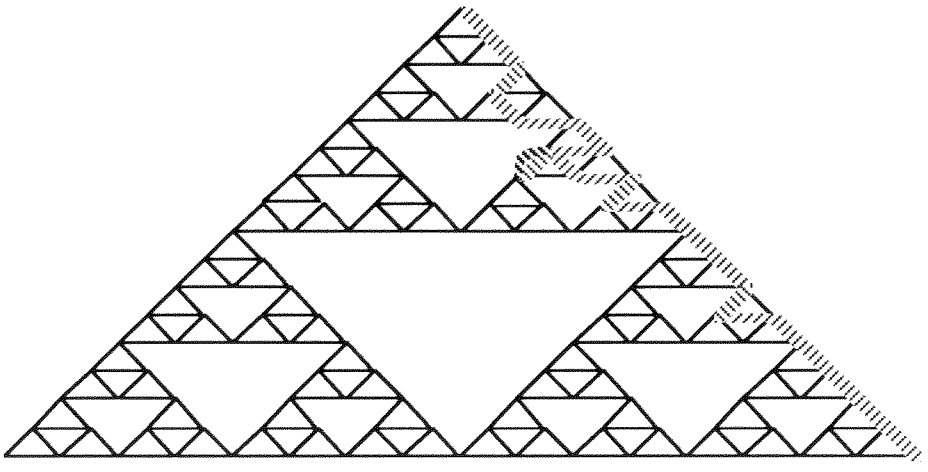

Abbildung 5 : Rekursive Lösung des Fünf-Scheiben-Problems

Wie TOM löst auch JAN im zweiten Interview sowohl das Vier-Scheiben-Problem als auch das Fünf-Scheiben-Problem. Zwar findet er beim Vier-Scheiben-Problem erneut die optimale Lösung, kommt aber beim Turm aus fünf Scheiben zu einer Lösung mit kleinen Umwegen (Abb. 5). Dennoch ist die Abweichung von der optimalen Lösung nicht bedeutsam. Die Hypothese, daß JAN intuitiv rekursiv vorgeht, läßt sich beibehalten.

Diskussion der Ergebnisse

Ist ein rekursives Problem auch als solches kognitiv repräsentiert, so folgt daraus noch nicht, daß auch die Handlungsschritte zur Lösung des Problems bekannt sind. Die Planung dieser Schritte setzt voraus, daß Hierarchien von Teilzielen entwickelt werden, die wie Module bearbeitbar sind. Man kann dies vielleicht am besten als eine Interaktion von Kurz- und Langzeitgedächtnis sehen. Die Hierarchie der Teilziele wird im Langzeitgedächtnis gespeichert. Aus dem Kurzzeitgedächtnis werden einzelne Teilziele abgerufen und dort in Handlungsschritte überführt (HAUSSMANN & REISS, 1988).
Die Untersuchungen zum TURM VON HANOI wurden an einem überschaubaren, leicht formalisierbaren Problem durchgeführt. Sie geben jedoch Aufschluß über rekursive Denkprozesse, wie sie auch beim Programmieren vorkommen. SIMON (1975) unterschied im Rahmen seiner Arbeit am General Problem Solver vier Strategien beim TURM VON HANOI:

- die vom Ziel her rekursive Strategie
- die differenzierte wahrnehmungsgeleitete Strategie
- die einfache wahrnehmungsgeleitete Strategie
- die Strategie der gelernten Zugmuster

Eine rekursive Strategie im strengen Sinne ist nur die erste, da sie ohne Umwege zu einer Lösung führt. Wenn man etwa eine Pyramide (k) mit k Scheiben hat, läßt sich die Strategie folgendermaßen als einfaches Produktionssystem darstellen:

Um Pyramide (k) von A nach C zu bewegen
bewege Pyramide (k-1) von A nach B,
bewege Scheibe (k) von A nach C,
bewege Pyramide (k-1) von B nach C.

Sowohl bei der einfachen als auch bei der differenzierten von der Wahrnehmung geleiteten Strategie ist das oberste Ziel das Umlegen der jeweils größten Scheibe, die noch auf dem Ursprungsfeld liegt. Damit sie bewegt werden kann, ist es zum einen erforderlich, alle Scheiben zu entfernen, die sich über ihr befinden. Zum andern muß aber auch das Zielfeld von Scheiben befreit werden, die kleiner sind als diese Scheibe. Sollten nur diese beiden Regeln in der genannten Reihenfolge zur Anwendung kommen, so sind unendliche Schleifen möglich. Die optimale Problemlösung wird in diesem Fall unwahrscheinlich, die Rückkehr zu bereits durchlaufenen Positionen ist möglich. SIMON bezeichnet diese Strategie als eine einfache von der Wahrnehmung geleitete.
Die differenzierte wahrnehmungsgeleitete Strategie ist im wesentlichen gleichzusetzen mit der eben geschilderten. Der einzige bedeutende Unterschied besteht darin, daß der Problemlöser sein Augenmerk darauf richtet, ob zum einen über der zu bewegenden Scheibe ein Hindernis liegt und zum anderen ob auf dem Zielfeld ein Hindernis liegt. Die eben dargestellte Strategie wird im folgenden erneut auf dieses Hindernis angewendet. Der Problemlöser greift sich nicht mechanisch eine jeweils mögliche Regel heraus, sondern arbeitet einen Stapel von Zielen ab, die im Kurzzeitgedächtnis gespeichert sind. Auch hier handelt es sich um eine Strategie, die rekursive Anteile hat, sie unterscheidet sich aber trotzdem von der ersten Strategie. Die erste Strategie geht von der Zielhierarchie aus, die jeweils erforderlichen Schritte werden nur an der wahrgenommenen Situation überprüft. Die letzte Strategie orientiert sich zwar an der Zielhierarchie, geht aber von der wahrgenommenen Situation aus.
Faßt man diese beiden Strategien zusammen, so läßt sich als Gemeinsamkeit herausstellen: Ziele werden in Teilziele aufgespalten und Zielhierarchien abgearbeitet; insofern sind diese Strategien schlußfolgernd. Es ist natürlich aber auch möglich, daß Züge geübt und Zugsequenzen ohne Einsicht in den rekursiven Prozeß auswendig behalten werden oder daß Tricks statt Problemlösestrategien zur Anwendung kommen. Die folgende Sequenz von Regeln stammt wieder von SIMON (1975) und kann als Beispiel für ein solches Vorgehen gelten:

- Zähle die Anzahl der bisher ausgeführten Züge.
- Ist die Anzahl der bisher ausgeführten Züge ungerade, dann bewege die kleinste Scheibe, sonst bewege die nächstgrößere Scheibe, die auf einem der Stäbe oben liegt.
- Wird die kleinste Scheibe bewegt und ist die Anzahl der Scheiben im Spiel gerade, dann bewege sie zyklisch vom Ursprungsfeld über das Zwischenfeld auf das Zielfeld und zurück zum Ursprungsfeld.
- Wird die kleinste Scheibe bewegt und ist die Anzahl der Scheiben im Spiel ungerade, dann bewege sie zyklisch vom Ursprungsfeld über das Zielfeld auf das Zwischenfeld und zurück zum Ursprungsfeld.

Diese Strategie kann man sich rasch aneignen, sie läßt sich im Langzeitgedächtnis speichern und ist weder eine Belastung für das Kurzzeitgedächtnis noch stellt sie größere Anforderungen an die Mustererkennung. Hier wird zwar in effektiver Weise jeweils neu ein Regelsystem auf die Situation angewendet, aber keine Zielhierarchie abgearbeitet.
Wie unterscheiden sich nun diese Strategien in der Problemlösegüte? Die rekursive Strategie führt auf dem kürzesten Wege zur Zielposition. Bei der einfachen wahrnehmungsgeleiteten Strategie kann es leicht zu zirkulären Bewegungen im Problemraum kommen, die differenziertere wahrnehmungsgeleitete Strategie kann vom optimalen Lösungsweg abweichen, jedoch ist die Wahrscheinlichkeit recht hoch, daß sie irgendwann zur Lösung führt. SIMONs Ansatz, Problemlösen mit Hilfe von Produktionssystemen zu erklären, läßt sich gut mit den Überlegungen von KLIX (1971) verbinden. Die Klassifikation der Strategien, wie sie KLIX (1971) vornimmt, hat als Kriterium den Suchraum, in dem sich die Züge bewegen. SIMON (1975) geht dagegen von Regeln eines Produktionssystems aus, das er dem Problemlöseverhalten seiner Versuchspersonen zugrunde legt. Bei Anwendung der Regeln ergeben sich die Züge automatisch. Es ist bisher auch in neueren Arbeiten (KARAT, 1982; HUSSY, 1987) noch nicht versucht worden, beide Ansätze zusammenzubringen: Der Suchraum ist in der Tat ein gutes Kriterium zur Unterscheidung verschiedener Strategien, es gilt aber, die Regeln herauszufinden, die beim Verfolgen dieser Strategien angewandt wurden, und darzustellen, zu welchen Bewegungen im Suchraum diese Regeln führen. In einem weiteren Schritt haben wir diese Regeln für den iterativen Fall in einem Produktionssystem modelliert

(HAUSSMANN & REISS, in Druck). Diese Form der Simulation trägt dazu bei festzustellen, in welchem Ausmaße sich einzelne Regeln auf die Anzahl der Züge bis zur Problemlösung auswirken. Außerdem kann man mit Hilfe einer Simulation herausfinden, ob eine Strategie auch bei Problemen größerer Komplexität zu einer Lösung führt.
Folgt man KLIXs struktureller Darstellung, so gibt es den idealen Problemlöser, der seine Züge im Lösungsdiagramm entlang der rechten Kante des großen Dreiecks sucht, aber auch den von uns iterativ genannten Problemlöser, der sukzessiv den Suchraum einschränkt (vgl. Abb. 6).

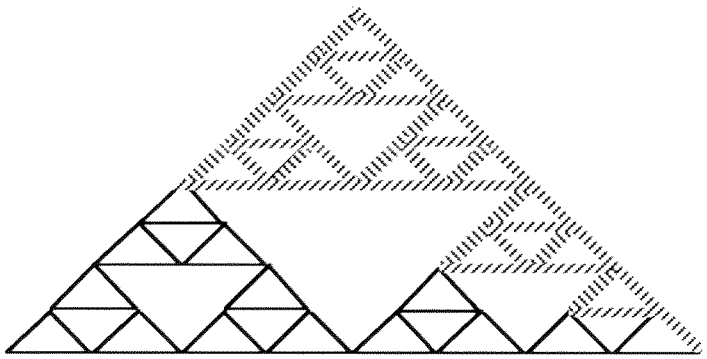

Abbildung 6 : Iterative Lösung des Vier-Scheiben-Problems

In unserer Untersuchung steht JAN für die rekursive Strategie. Seine Lösung ist als Pfad an der rechten Kante des großen Dreiecks zu beschreiben. Ob es sich dabei um eine rekursive Strategie im engeren Sinne oder doch nur um eine differenzierte wahrnehmungsgeleitete Strategie nach SIMON handelt, kann allein aus der Lokalisierung der Zügen im Suchraum nicht geschlossen werden. Die verbalen Protokolle enthalten Anzeichen dafür, daß JAN den Turm von Hanoi als rekursives Problem versteht. TOM dagegen macht Züge, die mit dem von ihm gesetzten Unterziel konsistent sind, die größte noch verbleibende Scheibe auf das Zielfeld zu bewegen. Das Problems wird nicht als rekursive Aufgabenstellung gesehen, es gibt keine abgestufte Zielhierarchie. SIMON würde dies vielleicht als eine perzeptive Strategie bezeichnen.

Das Ziel dieser Untersuchung war es zum einen, Ansätze zur Lösung eines Problems zu beschreiben, das in optimaler Weise rekursiv lösbar ist. Es konnten dabei verschiedene

Strategien identifiziert werden, die die Schüler bei der Lösung des Problems anwendeten. In Anlehnung an KLIX wurden unterschiedliche Strategien beschrieben, die sich in ihrer Nähe zu einer rekursiven Problemlösung unterscheiden.
Zum andern zeigen die anfangs aufgeführten Beispiele zum Programmieren, daß auch dort zwei Gruppen von Problemlösern unterschieden werden können. Es gibt Personen, die ganz eindeutig iterative Strageien bei der Lösung rekursiver Probleme verfolgen. Es gibt auf der anderen Seite aber auch Personen, die ein mehr oder weniger elaboriertes Verständnis einer rekursiven Problemlösemethode zeigen. STEFAN kann sicherlich der ersten Gruppe zugeordnet werden. Dabei fällt auf, daß er genauso wie TOM Vorerfahrungen in BASIC besitzt. Wir möchten allerdings keine Hypothese darüber aufstellen, was hier die Ursache und was die Wirkung ist. Es ist möglich, daß ein Programmieren in BASIC einen iterativen Denkstil fördert, es ist aber genauso möglich, daß eher iterativ orientierte Sprachen von Personen vorgezogen werden, die einem iterativen Denkstil zuzuordnen sind.

SEBASTIAN läßt sich als rekursiver Problemlöser charakterisieren. Er erkennt, daß die gestellten Probleme besonders gut mit Hilfe eines rekursiven Aufrufs im Programmtext zu realisieren sind. Man sieht aber auch, daß SEBASTIAN ein guter Programmierer ist, der über einige Routine beim Erstellen von Programmen verfügt. Bei KLAUDIA kann man erkennen, daß sie den rekursiven Aufruf aus einer iterativen Darstellung ableitet. Sie lernt während des Problemlöseprozesses, daß eine rekursive Notation für die Lösung des gegebenen Problems eine angemessene Darstellung ist. Das Beispiel ihres Programmes macht besonders deutlich, daß es bei der Entwicklung rekursiver Strukturen Zwischenschritte gibt und welcher Art diese Zwischenschritte sein können. Obwohl natürlich eine rekursive Strategie als optimal betrachtet werden muß, können iterative Ansätze als Möglichkeit der Hinführung zur Rekursion genutzt werden.

Im Rahmen unseres Projekts zum rekursiven Denken werden die Ergebnisse zum Problemlösen beim TURM VON HANOI in Beziehung gesetzt zur Kompetenz beim Programmieren, genauer zu den Leistungen bei der Planung rekursiver Programme (vgl. HAUSSMANN & REISS, 1987), bei ihrem Verständnis (HAUSSMANN & REISS, 1989), beim Schreiben und bei der Fehlersuche ("debugging") in solchen Programmen. Der TURM VON HANOI hat sich dabei als ein interessanter Indikator rekursiven Denkens er-

wiesen, weil diese Problemlöseleistung unabhängig ist von spezifischem Wissen über die Syntax und Semantik einer Programmiersprache bzw. vom mathematischen Notationswissen. Er ist durch die KLIXsche Darstellung in seiner Struktur formalisierbar und durch Produktionssysteme als Problemlösungsprozeß beschreibbar. Über Spezifika der kognitiven Prozesse geben die Interviews Aufschluß.

Literatur

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

ANDERSON, J.R., FARRELL R. & SAUERS R. (1984). Learning to program in LISP. *Cognitive Science 8*, 87-129.

ANZAI, Y. & SIMON, H. (1979). The theory of learning by doing. *Psychological Review, 86,* 124-140.

BARFURTH, M. (1987). *Recursion? What is it?* Fribourg: Institut de Psychologie, Université de Fribourg.

BAUER, F.L. & GOOS, G. (1982[3]). *Informatik. Eine einführende Übersicht. Erster Teil.* Berlin: Springer.

COHORS-FRESENBORG, E. (1985). Verschiedene Repräsentationen algorithmischer Begriffe. *Journal für Mathematikdidaktik 6,* 187-209.

DÖRFLER, W. (1984). Fundamentale Ideen der Informatik und Mathematikunterricht. In Österreichische Mathematische Gesellschaft (Hrsg.), Vorträge Symposium über Schulmathematik am 29.9.1983 in Salzburg (S. 19-40). *ÖMG Didaktik-Reihe, 10/2.* Wien.

DUPUIS, C., EGRET, M.A. & GUIN, D. (1985). *Récursivité et LOGO. 1. Préexpérimentation.* Strasbourg: Université Louis Pasteur. IREM.

FUNKE, J. (1988). Using simulation to study complex problem solving. *Simulation and Games, 19,* 277-303.

HAUSSMANN, K. (1985). Iterative and recursive modes of thinking in mathematical problem solving processes. In L. STREEFLAND (Ed.), *Proceedings of the Ninth International Conference for the Psychology of Mathematics Education, Vol. 1* (pp. 18-23). Utrecht: State University.

HAUSSMANN, K. (1986). Iteratives vs. rekursives Denken beim Problemlösen im Mathematikunterricht. *Mathematica didactica 9(2),* 61-74.

HAUSSMANN, K. (1987). *LOGO? LOGO! Ein Programmierbuch.* Braunschweig: Vieweg.

HAUSSMANN, K. & REISS, M. (1986). Rekursive Strukturen und ihre Rolle im Mathematikunterricht. *Karlsruher Pädagogische Beiträge 13,* 70-90.

HAUSSMANN, K. & REISS, M. (1987). LOGO beginners problems with goal merging. J. Hillel (Ed.) *Proceedings of the Third International Conference for LOGO and Mathematics Education* (pp. 156-163). Montréal.

HAUSSMANN, K. & REISS, M. (1988). KASIMIR: An investigation of iterative solution strategies for the TOWER OF HANOI problem. *Proceedings of the Bilateral German-Italian Symposium on Didactics of Mathematics.* Pavia.

HAUSSMANN, K. & REISS, M. (1989). Strategien bei Problem mit rekursiver Lösung - eine prozeßorientierte Analyse rekursiven Denkens. *Journal für Mathematikdidaktik, 10*, 39-61.

HAUSSMANN, K. & REISS, M. (in Druck). KASIMIR - die Modellierung einer iterativen Strategie beim Lösen eines rekursiven Problems. In K. HAUSSMANN & M. REISS (Hrg.), *Mathematische Lehr-Lern-Denkprozesse.* Göttingen: Hogrefe.

HOFSTADTER, D.R. (1985). *Gödel, Escher, Bach. Ein Endloses Geflochtenes Band.* Stuttgart: Klett-Cotta.

HUSSY, W. (1987). Zur Steuerfunktion der Sprache beim Problemlösen. *Sprache und Kognition 1,* 14-22.

KARAT, J. (1982). A model of problem solving with incomplete constraint knowledge. *Cognitive Psychology, 14,* 538-559.

KLIX F. (1971). *Information und Verhalten.* Bern: Huber.

KURLAND, D.M. & PEA, R.D. (1983). *Children's mental models of recursive LOGO programs.* Technical Report No. 10. New York: Center for Children and Technology. Bank Street College of Education.

MÖBUS, C., SCHRÖDER, O. & COLONIUS, H. (1986). Programmieren mit mentalen Modellen?! In M. AMELANG (Hrsg.), *Bericht über den 35. Kongreß der Deutschen Gesellschaft für Psychologie, Band 1* (S. 199). Göttingen: Hogrefe.

PAPERT, S. (1980). *Mindstorms. Children, Computers, and Powerful Ideas.* New York: Basic Books.

PEA,R.D. & KURLAND, D.M. (1983). *On the cognitive prerequisites of learning computer programming.* Technical Report No. 18. New York: Center for Children and Technology. Bank Street College of Education.

REISS, M. & HAUSSMANN, K. (1988). KASIMIR: eine Analyse iterativer Strategien zur Lösung eines rekursiven Problems mit Hilfe eines Produktionssystems. In K. EYFFERTH (Hrsg.), *Bericht über den 36. Kongreß der Deutschen Gesellschaft für Psychologie.* Göttingen: Hogrefe.

REISS, M. (1984). Vom Programmieren zum mathematischen Verallgemeinern. In W. ARLT & K. HAEFNER (Hrsg.), *Informatik als Herausforderung an Schule und Ausbildung* (S. 229-233). Heidelberg: Springer.

ROBERTS, E.S. (1986). *Thinking recursively.* New York: Wiley.

SCHMALHOFER, F. & KÜHN, O. (1986). Die erste Stunde beim Erwerb von Programmierkenntnissen - eine Computermodellierung im Ansatz. In M. AMELANG (Hrsg.), *Bericht über den 35. Kongreß der Deutschen Gesellschaft für Psychologie, Band 1* (S. 199). Göttingen: Hogrefe.

SIMON, H.A. (1975). The functional equivalence of problem solving skills. *Cognitive Psychology 7,* 268-288.

SPOHRER, J. C., SOLOWAY, E. & POPE, E. (1985). A goal/plan analysis of buggy Pascal programs. *Human Computer Interactions, 1(2),* 163-207.

VORBERG, D., GRUNER, E., HAHN, K., HEIM D., SCHULZE, H.H. & WAGNER, K.U. (1986). Entwicklung von Programmierwissen und seine Anwendung beim Problemlösen. M. AMELANG (Hrsg.), *Bericht über den 35. Kongreß der Deutschen Gesellschaft für Psychologie. Band 1*, (S.198). Göttingen: Hogrefe.

WALOSZEK, G., WEBER, G. & WENDER, K.F. (1986). Diagnose von Wissen - Entwicklung der Diagnosekomponente eines intelligenten LISP-Tutors. In M. AMELANG (Hrsg.), *Bericht über den 35. Kongreß der Deutschen Gesellschaft für Psychologie, Band 1* (S. 196). Göttingen: Hogrefe.