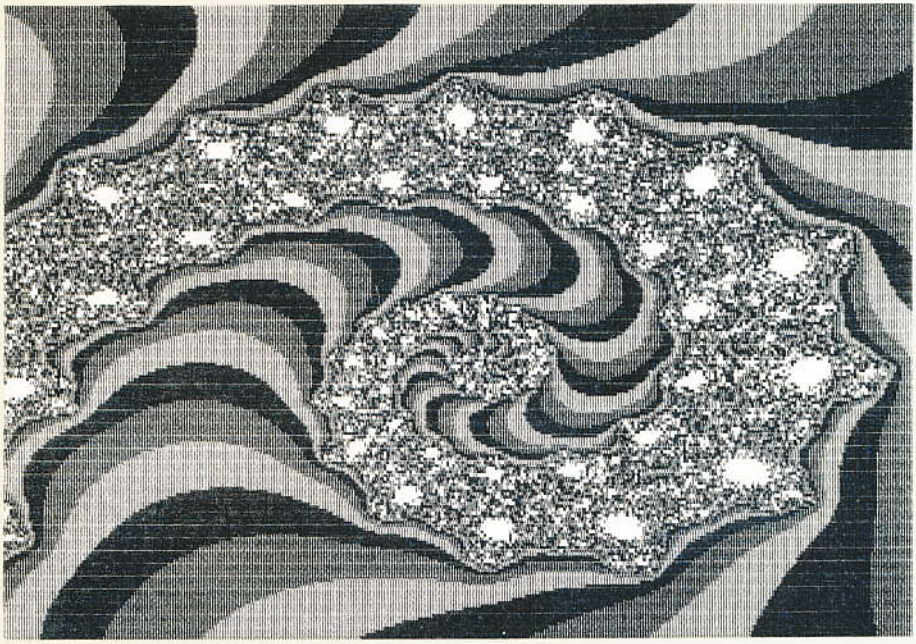
WISSENS PSYCHOLOGIE

Herausgegeben von
Heinz Mandl und Hans Spada



Psychologie Verlags Union

Heinz Mandl/Hans Spada (Hrsg.)

Wissenspsychologie

Mit Beiträgen von

Hans Aebli, Gerhard Dirlich, Dietrich Dörner,
Christian Freksa, Angela D. Friederici, Helmut F. Friedrich,
Ulrich Furbach, Theo Herrmann, Siegfried Hoppe-Graff,
Aemilian Hron, Ludwig J. Issing, Friedhart Klix,
Rainer H. Kluwe, Alan Lesgold, Gerd Lüer, Heinz Mandl,
Claus Möbus, Rolf Oerter, Klaus Opwis, Günther Palm,
Wiebke Putz-Osterloh, Wolfgang Schnotz, Hans Spada,
Gerhard Steiner, Sigmar-Olaf Tergan, Michael R. Waldmann,
Franz E. Weinert, Karl F. Wender, Friedrich Wilkening

Psychologie Verlags Union
München – Weinheim 1988

Anschriften der Herausgeber

Professor Dr. Heinz Mandl
Deutsches Institut für Fernstudien
Hauptbereich Forschung
Bei der Fruchtschranne 6
7400 Tübingen

Professor Dr. Hans Spada
Lehrstuhl für Allgemeine Psychologie
Psychologisches Institut
Niemensstraße 10
7800 Freiburg i. Br.

Anschriften des Wissenschaftlichen Beirates des Psychologie-Programms

Prof. Dr. Dieter Frey, Institut für Psychologie der Universität Kiel, Olshausenstraße 40/60, 2300 Kiel
Prof. Dr. Siegfried Greif, Fachbereich Psychologie der Universität Osnabrück, Knollstr. 15, 4500 Osnabrück
Prof. Dr. Heiner Keupp, Institut für Psychologie, Sozialpsychologische Abteilung, Universität München, Leopoldstraße 13, 8000 München 40
Prof. Dr. Ernst-D. Lantermann, Gesamthochschule Kassel, FB 3, Heinrich-Plett-Straße 40, 3500 Kassel
Prof. Dr. Rainer K. Silbereisen, Fachbereich Psychologie, Justus-Liebig-Universität Gießen, Otto-Behaghel-Straße 10, 6300 Gießen
Prof. 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

Wissenspsychologie / Heinz Mandl ; Hans Spada (Hrsg.). Mit Beitr. von Hans Aebli ... – München; Weinheim : Psychologie-Verl.-Union, 1988
ISBN 3-621-27016-7
NE: Mandl, Heinz [Hrsg.]; Aebli, Hans [Mitverf.]

Als Titelbild wurde eine am Deutschen Institut für Fernstudien an der Universität Tübingen (DIFF) – Arbeitsbereich Mathematik/Informatik angefertigte Computer-Graphik (Programm ‚Fraktalbild' von W. Knapp) verwendet.

Gesamtherstellung: H. Mühlberger, 8900 Augsburg

ISBN 3-621-27016-7

18 Zur Modellierung kognitiver Prozesse mit daten- bzw. zielorientierten Regelsystemen

Claus Möbus

Einleitung

Mit diesem Beitrag wollen wir drei Ziele verfolgen.

(1) *Wissensrepräsentation mit Regelsystemen:* Wir wollen anhand ausgewählter Beispiele *Repräsentationsmöglichkeiten* für deklaratives und prozedurales Wissen demonstrieren. Die Beispiele stammen aus dem Bereich der Anwendung von Problemlösefertigkeiten („Turm von Hanoi") und der Gedächtnispsychologie („Semantisches Gedächtnis"). Das "Turm"-Problem eignet sich gut für eine einführende Betrachtung, weil zu seiner Lösung relativ wenig Wissen erforderlich ist und daher methodische Prinzipien klarer hervortreten können. Wir werden auf der Basis dieses Beispiels auch einige Bezüge zu dem Artikel von Opwis (in diesem Band) herstellen. Es werden sowohl Parallelen als auch Divergenzen in den Modellansätzen herausgearbeitet.

Den zweiten Bereich „Semantisches Gedächtnis" wählten wir, weil einerseits Klix und Wender in diesem Band dazu Beiträge geliefert haben und andererseits die Modellierung von semantischen Netzen mit formalen Modellen die Unzulänglichkeit verbaler Theorien deutlich belegt. Wir gehen in diesem Artikel von idealisiert vereinfachten informationsverarbeitenden Prozessen aus, um die Regelsysteme übersichtlich zu halten.

(2) *Darstellung vorwärts- bzw. rückwärtsverkettender Regelsysteme als Modelle für daten- bzw. zielgesteuerte Informationsverarbeitung:* Wir wollen einen kurzen Einblick in die zur Zeit diskutierten *Methoden* und *Werkzeuge* geben, die für eine kognitionspsychologische Modellierung im Sinne eines *computational models* (Pylyshyn, 1984) in Frage kommen. Hierzu zählen wir speziell die Regelmodelle („Produktionssysteme"), obwohl Schemaansätze in verschiedenartigem Gewand („Objektorientierte Systeme", *frames*) immer wieder in der Literatur auftauchen. Zur Implementation der Modelle sind verschiedene Softwarewerkzeuge (LISP, LOOPS, PROLOG, SMALLTALK etc.) entwickelt worden. Wir wollen uns hier auf die Regelsysteme und deren Realisierung in LISP und PROLOG beschränken. Der Schemaansatz kann deshalb ausgeklammert werden, weil es bisher an empirischen Belegen mangelt.

(3) *Realisierung von Regelsystemen mit den KI-Sprachen LISP und PROLOG:* Das dritte Ziel besteht darin, den Mangel an Lehrbuchmaterial auf diesem Gebiet etwas auszugleichen und dem Leser Anregung und Selbstvertrauen zu geben, sich

aktiv in die Modellierung mit *computational models* einzuschalten. Die technischen Möglichkeiten sind im Prinzip vorhanden. Jeder gängige Personalcomputer kann zu einem symbolverarbeitenden Informationssystem erweitert werden, wie es bis vor wenigen Jahren nur an den renommiertesten Forschungsinstitutionen (in den USA: Carnegie-Mellon University, MIT, Yale, Stanford, . . .; in Deutschland: Hamburg, Stuttgart) verfügbar war. Zu diesem Zweck haben wir einige Programmbeispiele beigefügt. Die Programme sind in den KI-Sprachen LISP und PROLOG geschrieben und auf handelsüblichen Personalcomputern lauffähig. Die verwendeten LISP- und PROLOG-Dialekte (TLC-LISP, micro PROLOG) sind zudem ausführlich in zwei Lehrbüchern dokumentiert (Stoyan & Görz, 1984; Clark & McCabe, 1984, Kap. 9).

Von mathematischen zu informationsverarbeitenden Modellen

1972 fand in Ann Arbor (USA) eine Konferenz mathematischer Psychologen und Statistiker statt, in der eine Bestandsaufnahme formaler Theorien und Modelle in der Psychologie erarbeitet wurde. Das Ergebnis der Konferenz wurde zwei Jahre später in einem zweibändigen Werk mit den Titeln „Learning, Memory and Thinking" und „Measurement, Psychophysics and Neural Information Processing" (Krantz, Atkinson, Luce & Suppes, 1974) veröffentlicht. Es ist interessant, wenn man fast 13 Jahre später die Inhalte dieser Konferenz Revue passieren läßt. Auf einige Arbeiten, speziell diejenigen, die sich unter dem Blickwinkel einer besonderen mathematischen Methode (z. B. Analyse von Kovarianzmatrizen, multidimensionale Skalierung, Theorie latenter Merkmale, aber auch MARKOV-Prozesse) mit psychologischen Phänomenen befaßt haben, trifft eine Warnung von Lüer (1981, S. 90) zu: »Der Mathematiker und Physiker Lichtenberg (1742–1799) äußerte einmal, daß es viele Phänomene zwischen Himmel und Erde gibt, für die es keine Theorie oder Erklärung in unseren Lehrbüchern gäbe. Ebenso gibt es viele Theorien und Erklärungen in Lehrbüchern, für die es keine Phänomene zwischen Himmel und Erde gibt. Vor der letztgenannten Gefahr sollten wir uns in der kognitiven Psychologie sehr nachhaltig schützen." Andere Arbeiten kann man aus heutiger Sicht als wegweisend nennen. Hierzu gehört speziell der Aufsatz von Simon und Newell über Denkprozesse. Die Autoren referierten in ihm ihre Theorie der menschlichen Informationsverarbeitung (Newell & Simon, 1972) und verglichen diese mit den zu der Zeit unangefochten vorherrschenden mathematisch stochastischen Parametermodellen (Greeno, 1974; Millward, 1974; Wickens, 1974) am Beispiel des Konzepterwerbs. Ihre Theorie und alle anderen *computational models* unterscheiden sich von mathematischen Theorien bzw. Modellen durch einen größeren Detailliertheitsgrad. Statt einer Metabeschreibung eines kognitiven Prozesses, der sich in Übergangs- und Zustandswahrscheinlich-

keiten (z. B. bei MARKOV-Prozessen) niederschlägt, versucht man, den *Mechanismus* des informationsverarbeitenden Prozesses selbst zu modellieren. Dabei stützt man sich auf Einzelfalluntersuchungen, Interpretation introspektiver Daten (z. B. Verbalprotokolle; Ericsson & Simon, 1984), theoretisch auf Enkodierung und Repräsentation von Situationen sowie Reizen und methodisch auf formalisierte, auf dem Computer ablauffähige Regelsysteme („Produktionssysteme"). Sie beschränken sich auf die Informationsverarbeitung *symbolischer* Repräsentationen. Darunter versteht man die Verarbeitung von Informationspaketen (z. B. Zielen, Wörtern, Sätzen, Bildern) auf einer Ebene, die von der konkreten Realisierung im physiologischen neuronalen Substrat abstrahiert (Newell, 1980).

In letzter Zeit sind unter dem Stichwort „Parallelität der informationsverarbeitenden Prozesse" verstärkt Anstrengungen unternommen worden, Prozesse unterhalb der symbolischen Ebene mit einem größeren Auflösungsgrad zu modellieren (Hinton & Anderson, 1981; Rumelhart, McClelland & PDP Research Group, 1986; McClelland, Rumelhart & PDP Research Group, 1986). Beide Beschreibungsebenen und die darauf beruhenden Berechnungsmodelle stehen dabei nicht in einem Widerspruch, sondern sind abhängig von der jeweiligen Betrachtungsposition (Anderson & Hinton, 1981, S. 29 ff.) und dem Feinheitsgrad der Beschreibung.

Wir konzentrieren uns hier auf die symbolische Informationsverarbeitung. Damit einher geht die Entscheidung, ein „intentionales Vokabular" zu benutzen (Phylyshyn, 1984, S. 5). Dieses unterscheidet sich wesentlich von einer neurophysiologischen oder behavioralen Sprachebene durch solche Begriffe wie: Wahrnehmung, Wiedererkennung, Inferenz, Entscheidung, Deduktion, Wissen, Zielsetzung etc.

Ein formales Beschreibungsmittel soll die Regel sein. Mit Regeln können Momentaufnahmen des Wissens einer Person festgehalten werden, und man kann Lernprozesse mit Generalisations- und Diskriminationslernen modellieren (Anderson, Kline & Beasley, 1980) sowie die Prozeduralisierung von Wissen mittels Regelverkettung *(compilation)* nachbilden (Anderson, 1983a,b; Neves & Anderson, 1981). In letzter Zeit geht man in begründeten Fällen von rein deterministischen Regeln ab (Bandler & Kohout, 1985; Schmalhofer & Polson, 1986). Speziell bei „niederen" kognitiven Phänomenen und bei einem hohen Auflösungsgrad der Phänomenbeschreibung *(grain size)* scheinen stochastische Theorien angebracht, wie Card, Moran und Newell (1983, S. 180) an einer Hierarchie unterschiedlich differenzierender Modelle zum gleichen Realitätsausschnitt festgestellt haben.

Der Ursprung der Produktionssysteme: MARKOV-Algorithmen

Produktionssystemmodelle gehen auf die Logiker Post und Thue sowie auf den Mathematiker Markov zurück. Sie sind in ihrer allgemeinsten Form als Ersetzungssysteme (s. u.) bekannt (s. a. Bauer & Goos, 1984, Bd. II, Kap. 7). Mit ihrer Hilfe läßt sich der Begriff des Algorithmus präzisieren. Ihre praktische Bedeutung haben sie zuerst in Syntaxdefinitionen von natürlichen und künstlichen Sprachen erlangt. Der Linguist Chomsky definierte mit ihrer Hilfe eine Sprachhierarchie, die nach Mächtigkeit geordnet ist. Ein Ersetzungssystem besteht aus einem Vokabular von Zeichen, auf dem das System arbeitet, und einer Menge von Produktionen. Das Vokabular zerlegt sich in eine Menge von sogenannten Terminal- und Nichtterminalzeichen (= Hilfszeichen). Letztere dienen hauptsächlich der Kontrolle der Ersetzungen. Gibt es nun eine lineare Ordnung der Regeln und wird eine Ersetzung soweit wie möglich links in der Zeichenkette vorgenommen, hat man einen wichtigen Spezialfall vorliegen: den „MARKOV-Algorithmus" (nicht zu verwechseln mit den probabilistischen MARKOV-Ketten oder MARKOV-Prozessen der mathematischen Psychologie).

Einen derartigen MARKOV-Algorithmus stellen wir an einem Beispiel aus der statistischen Interventionsanalyse von Zeitreihenexperimenten (Möbus & Nagl, 1983; Möbus, Göricke & Kröh, 1983, 1985) vor. Bei der Interventionsanalyse wird eine Zeitreihe daraufhin untersucht, ob sie sich aus interventionsbedingten deterministischen sowie aus stochastischen Anteilen zusammensetzt: modelliert man das Vorhandensein und das Nichtvorhandensein einer Intervention mit einem Stepinput I(t) [keine Intervention: I(t) = 0; Intervention I(t) = 1], kann die Änderung der Intervention $\Delta I(t) = | I(t) - I(t-1) |$ ebenfalls einen Einfluß auf die beobachtbare Zeitreihe haben. Wie berechnet man nun den Pulsinput $\Delta I(t)$ mit Hilfe eines MARKOV-Algorithmus?

Die Zeitreihen des Stepinput I(t) und des Pulsinput $\Delta I(t)$ besitzen die Realisationen:

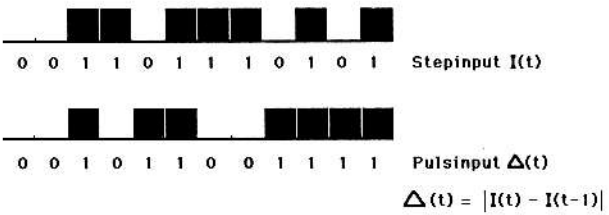


Abb. 18.1.: Zeitreihe I(t) mit deterministischen Realisationen und daraus durch Differenzenbildung abgeleitete Zeitreihe $\Delta(t)$

Das Vokabular der Terminalzeichen besteht also aus (0,1,.). Wie sich gleich zeigen wird, benötigen wir die Hilfszeichen ($, &) und sieben Produktionen nach dem Muster ⟨Bedingungszeichenkette⟩ ===⟩ ⟨Konsequenzzeichenkette⟩:

1. Startproduktion:		===⟩ $
2. Verschieben der Kontrolle	$0	===⟩ 0$
3. Erste Eins nach Nullkette bleibt:	$1	===⟩ 1&
4. Löschen aller weiteren Einsen:	&1	===⟩ 0&
5. Umwandlung erster Null nach Einsen:	&0	===⟩ 1$
6. Endproduktion (Löschen von $):	$	===⟩ .
7. Endproduktion (Löschen von &):	&	===⟩ .

Da bei MARKOV-Algorithmen mit dem Ersetzen von Zeichen in der Zeichenkette am weitesten links angesetzt wird, bedeutet die Anwendung der ersten Produktion ein Vorsetzen des Hilfszeichens $ vor die Ausgangszeichenkette. Danach können die weiteren Produktionen auf die Zeichenkette nach folgender Übersicht angewendet werden:

1. Produktion ist anwendbar auf:	0 0 1 1 0 1 1 1 0 1 0 1
2. Produktion ist anwendbar auf:	$0 0 1 1 0 1 1 1 0 1 0 1
2. Produktion ist anwendbar auf:	0$0 1 1 0 1 1 1 0 1 0 1
3. Produktion ist anwendbar auf:	0 0$1 1 0 1 1 1 0 1 0 1
4. Produktion ist anwendbar auf:	0 0 1&1 0 1 1 1 0 1 0 1
5. Produktion ist anwendbar auf:	0 0 1 0&0 1 1 1 0 1 0 1
3. Produktion ist anwendbar auf:	0 0 1 0 1$1 1 1 0 1 0 1
4. Produktion ist anwendbar auf:	0 0 1 0 1 1&1 1 0 1 0 1
4. Produktion ist anwendbar auf:	0 0 1 0 1 1 0&1 0 1 0 1
5. Produktion ist anwendbar auf:	0 0 1 0 1 1 0 0&0 1 0 1
3. Produktion ist anwendbar auf:	0 0 1 0 1 1 0 0 1$1 0 1
5. Produktion ist anwendbar auf:	0 0 1 0 1 1 0 0 1 1&0 1
3. Produktion ist anwendbar auf:	0 0 1 0 1 1 0 0 1 1 1$1
7. Produktion ist anwendbar auf:	0 0 1 0 1 1 0 0 1 1 1 1&
Der Pulsinput △I(t) ist:	0 0 1 0 1 1 0 0 1 1 1 1.

Damit wird deutlich, daß algorithmisches Wissen (so z. B. die Zeitreihe I(t) zu △I(t) zu differenzieren) durch Regelanwendung realisiert werden kann. Wie Opwis (in diesem Band) schon ausführt, wurden die MARKOV-Algorithmen mit Hilfe einer Neuinterpretation zu Grundformen der Modellierung kognitiver Prozesse („neoklassische" Produktionssysteme in der Terminologie von Anderson, 1983a). Die Zeichenkette, auf der der Algorithmus operiert, kann als Inhalt des Arbeitsgedächtnisses verstanden werden. Die Regeln stellen das prozedurale Wissen dar, das als im Langzeitgedächtnis abgespeichert gelten kann. Die sukzessive Anwendung der Regeln entspricht der Hypothese, daß bei höheren kognitiven Prozessen Parallelverarbeitung zugunsten sequentieller Verarbeitung zurücktritt.

Planen, Probieren, Vorwärts- und Rückwärtsverkettung von Regeln

Bevor wir konkrete Regelsysteme vorstellen, wollen wir einige Begriffe einführen.

Problemlösen kann als Weg bzw. Suche im Problem-Verhaltensgraphen vom Startknoten (Formulierung des Problems) zu einem oder mehreren Zielknoten (Lösungen des Problems) aufgefaßt werden. Diese Sicht wird z. B. von Newell (1980) und Card, Moran und Newell (1983, S. 361 ff.) explizit als *Problemraumhypothese* bezeichnet. Die Knoten des Graphen stellen die jeweiligen *Zustände* des Problemlöseprozesses dar. Die *Übergänge* zwischen den Knoten werden durch die Anwendung von Problemlöseoperatoren vollzogen. Dabei können auf eine bestimmte Problemsituation mehrere *Operatoren* anwendbar sein, so daß algorithmisches oder heuristisches Wissen über das Problem die Auswahl von Operatoren einschränkt. Dieses heuristische Wissen wollen wir als *Kontrollwissen* bezeichnen. Charakteristisch für Novizen ist geringes Kontrollwissen und ausführliche Suche („Probieren") im Problem-Verhaltensgraphen, die sich auch in Irrwegen dokumentiert. Bei zunehmender Prozeduralisierung des Lösungswissens baut sich immer mehr Kontrollwissen auf, so daß der Experte die nicht optimale Anwendung von Operatoren vermeidet.

Wie kann ein Problemlöser zu einem Satz von Operatoren kommen? Eine Möglichkeit, die aber von Novizen meist nicht gewählt wird, liegt im Aufbau einer *Zielhierarchie*. Darunter versteht man die Zerlegung des globalen Ziels in einfachere Subziele. Die Subziele werden dann wieder in weitere Unterziele zerlegt, bis man auf einfache Ziele stößt, denen man Operatoren zuordnen kann. Diese sorgen dann für Bewegungen im Problemlösungsprozeß. Den Aufbau derartiger Zielhierarchien haben Card, Moran und Newell (1983) zu einem halbformalen Beschreibungssystem namens GOMS (*G*oals, *O*perator, *M*ethods, *S*electionrules) ausgeweitet. Der systematische Aufbau einer Zielhierarchie garantiert dem Problemlöser, alle für die Lösung relevanten Operatoren bereitgestellt zu haben. Wir wollen einen derartigen Prozeß als *Planungsprozeß (top-down-Vorgehen)* bezeichnen. Kann der Problemlöser die vollständige Zielhierarchie nicht aufbauen, ist kein globales Kontrollwissen über die zur Lösung führende Operatorsequenz vorhanden. Der Problemlöser ist dann auf lokales Wissen angewiesen. Er reiht die Operatoren aneinander, ohne Garantie, die Lösung zu finden *(Probieren, bottom-up-Vorgehen)*.

Experten und Novizen unterscheiden sich, wenn man in unserer Terminologie bleibt, nicht nur durch den mehr oder minder vollständigen Aufbau einer Zielhierarchie, sondern auch im Feinheitsgrad ihrer Subziele. Beim Experten stoppt der Aufbau einer Zielhierarchie früher als beim Novizen. Die weitere Ausdifferenzierung von Subzielen erübrigt sich, weil der Experte durch früher erfolgte Prozeduralisierung und *chunk*-Bildung spezielle situationsangepaßte Operatoren zur Verfügung hat. Er hat „Wissen über Muster" aufgebaut und führt Teilprobleme auf bekannte Fälle zurück. Rosenbloom und Newell (1986) haben zur *chunk*-Bildung

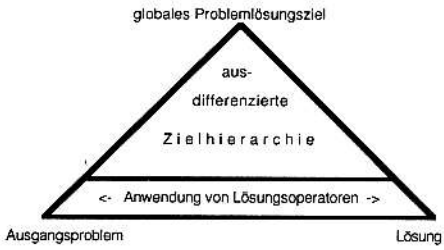


Abb. 18.2.: Schematische Darstellung der Konzepte Planung, Zielhierarchie und Problemlösen als gerichtete Suche im Problem-Verhaltensgraphen

und Prozeduralisierung von Wissen ein detailliertes formales, symbolverarbeitendes Modell entwickelt, das auch empirische Vorhersagen ermöglicht. Den Zielen in der Hierarchie lassen sich nach dem GOMS-Modell Operatoren oder Mengen von Operatoren (= Methoden) zuordnen. Wir wollen hier als Operatoren Regeln wählen. Beispiele für die Ableitung von Regeln aus Zielhierarchien finden sich u. a. bei Polson und Kieras (1984, 1985), Kieras und Polson (1985) und Rosenbloom und Newell (1986).

Die Regeln oder auch Implikationen bestehen aus Bedingungen (Prämissen) und Konsequenzen (Konklusionen). Arbeitet das Regelsystem vorwärts, wird zunächst geprüft, ob Regelbedingungen auf Inhalte im Arbeitsgedächtnis passen. Aus dieser Menge wird dann entsprechend dem Stand des Kontrollwissens eine Regel ausgewählt und angewendet. Es ist auch möglich, daß die Auswahl und Anwendung der Regeln unter Vorbehalt erfolgt und bis zu einem bestimmten Punkt, der von der Begrenztheit des Arbeitsgedächtnisses abhängig ist, wieder rückgängig gemacht werden kann („Probieren"). Die Anwendung einer Regel entspricht dabei im Problem-Verhaltensgraphen einem Übergang von einem Knoten zum Nachbarknoten.

Vorwärtsarbeitende Systeme sind charakteristisch für die Modelle der Carnegie-Mellon-Schule (Newell, Simon, Anderson) (s. a. Opwis, in diesem Band).

Demgegenüber erweisen sich für bestimmte Problembereiche (so z. B. bei Anforderungsspezifikationen zur Lösung eines Problems beim sogenannten *brainstorming* oder bei der Beantwortung von Fragen durch Suchprozesse in semantischen Netzen) rückwärtsverkettende Regelsysteme als adäquatere Beschreibungsmittel. Hierbei geht man vom gelösten Problem und seiner Repräsentation im Problem-Verhaltensgraphen aus und fragt, welche Voraussetzungen hergestellt oder ausgesucht werden müssen, damit diese Endsituation erreicht werden kann.

Man überprüft jetzt die *rechten* Seiten der Produktionen. Alle Regeln, deren rechte Seiten (d. h. Konklusionen der Regeln) auf die lösungsnäheren Knoten (Nachfolgerknoten) passen, liefern uns mit ihren *linken* Seiten Merkmale für Knoten (Vorgängerknoten), die der Lösung ferner und evtl. einer Ausgangssituation näher stehen.

Die Integration des Vorwärts- und Rückwärtsschreitens bei der Entwicklung von kognitiven Fertigkeiten und ihre praktische Bedeutung für die Konzeption von Lehr- und Schulbüchern ist z. B. von Anderson (1983b) und ausführlich bei Dörner (1976, Kap. 3) dargestellt.

Regeln können sich ferner noch darin unterscheiden, ob in ihnen Ziele enthalten sind. Rein *datengesteuerte* Regelbündel (d. h. ohne Ziele in den Bedingungen) werden in der Theorie von Rosenbloom und Newell (1986) nur bei autonomen, parallel arbeitenden Enkodierungs- und Dekodierungsvorgängen eingesetzt. Sie belasten das Arbeitsgedächtnis in keiner Weise und stehen nach Anderson (1983a, S. 126 ff.) am Ende eines langen intensiven Lernprozesses, der im seriellen zielorientierten Vorgehen (mit Kurzzeitspeicherbelastung) seinen Ursprung hat. Alle anderen Prozesse werden mit zielgesteuerten Regeln modelliert.

Wir werden am Beispiel des „Turm von Hanoi" zunächst die Vorwärtsverkettung und dann bei Suchprozessen in semantischen Netzen Rückwärtsverkettungen vorstellen.

Repräsentation von Kontrollwissen in einem vorwärtsverkettenden Produktionssystem: Der Turm von Hanoi

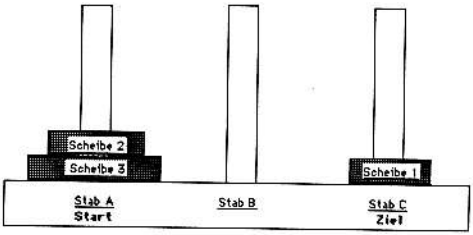
Der „Turm"

Die Denksportaufgabe „Turm von Hanoi" erfreut sich bei Kognitionspsychologen, Informatikern und Mathematikern einer ungebrochenen Beliebtheit. Sie dient wegen ihrer einfachen, aber dennoch nicht-trivialen Struktur als ideales Vehikel, alternative Methoden und Lösungsansätze miteinander zu vergleichen (Bauer & Goos, 1982, S. 145 ff.; Bauer & Wössner, 1984, S. 306; Crowe, 1956; Gardner, 1982).

In der Psychologie wurde der „Turm" gerade wegen der eindeutigen Definiertheit des Problem-Verhaltensgraphen zur Analyse von Denk- und Lernprozessen herangezogen und spiegelt dabei die Wandlungen der Zielsetzung und Methodologie der Kognitionsforschung wider (Anderson, 1983a; Anzai & Simon, 1979; Gediga & Schöttke, 1985; Karat, 1982; Klahr, 1981; Klix, 1971; Piaget, 1976; Simon, 1975; Spada, 1973; Sydow, 1970). Es wurde soviel über den „Turm" publiziert, daß Karat selbstironisch warnte „after all, the study of Tower of Hanoi problems is not the goal of cognitive science" (Karat, 1982, S. 555).

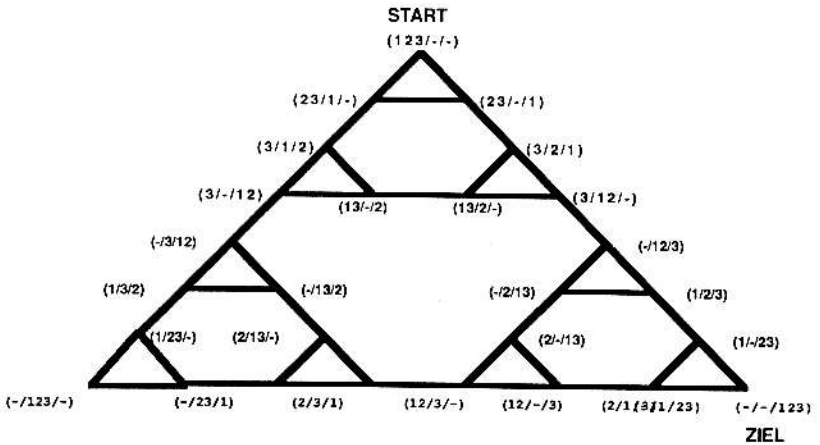
Eine informelle Beschreibung des Turmproblems findet sich in Abb. 18.3.

Der Problem-Verhaltensgraph für ein 3-Scheibenproblem ist in Abb. 18.4 dargestellt.



Informelle Beschreibung:
Drei Scheiben (Scheibe 1, 2 und 3) befinden sich auf einem Spielbrett. Die Numerierung der Scheiben ist nach Größe geordnet. In der Ausgangsposition des Problems befinden sich die Scheiben (zusammengefaßt zu einem sich nach oben verjüngenden Turm) beim Stab A. Die Aufgabe besteht darin, den Turm von Stab A zu Stab B zu schaffen. Dabei darf nur jeweils *eine* Scheibe zur Zeit bewegt werden. Außerdem darf niemals eine größere Scheibe auf einer kleineren zu liegen kommen. Daraus folgt, daß eine Scheibe nur dann bewegt werden kann, wenn sie oben auf dem Turm liegt.

Abb. 18.3.: Problemsituation beim „Turm von Hanoi", die zwischen Ausgangsproblem und Lösung liegt



Jede Liste beschreibt einen Zustand des Problemlöseprozesses. So bedeutet (1/-23), daß sich die große Scheibe 3 und die mittlere Scheibe 2 am Zielstab C befinden. Die kleinste Scheibe 1 befindet sich noch noch beim Ausgangsstab A. Die Situation von Abb. 1 läßt sich demnach mit (23/-/1) beschreiben und kennzeichnet den ersten Zug in direkter Richtung zum Ziel.

Abb. 18.4.: Der Problem-Verhaltensgraph für ein 3-Scheibenproblem des „Turm von Hanoi"

Graphen für Probleme mit mehr als drei Scheiben geben Gediga und Schöttke (1985), Karat (1982) und Spada (1973) an. Der Problemraum besteht aus einer Menge von Wissenszuständen, die mit den Scheibenkonfigurationen übereinstimmen. Die Wissenszustände können durch erfolgreiche Anwendung von (Problemlöse-)Operatoren gewechselt werden. Auf diese Weise kann man sich vom ursprünglichen Problemzustand (= Aufgabe) zum Zielzustand (= Lösung) vorarbeiten.

Wir führen als Operatoren vier voneinander unabhängige Produktionen ein und erweitern das Konzept des Problemraums um eine Zielkomponente, die im Arbeitsspeicher abgelagert ist. Die Unabhängigkeit der Produktionen wird durch sehr spezifisches Kontrollwissen erreicht, das in den Bedingungen und Aktionen der Regeln niedergelegt ist. Dadurch kommt es nie zu einer Konfliktmenge, die zwei oder mehr Regeln enthält. Die Regeln lauten:

P 1: WENN alle Scheiben sich bei Stab C befinden,
DANN ist das Problem gelöst

P 2: WENN es kein gegenwärtiges Ziel gibt, eine Scheibe zu bewegen,
DANN nimm Dir als neues Ziel vor, die größte Scheibe, die sich nicht bei Stab C befindet, dorthin zu schaffen

P 3: WENN es ein Ziel gibt, eine Scheibe zu bewegen
UND
WENN die Scheibe auf den Turm gelegt werden darf, der im Ziel vermerkt ist,
DANN setze das Ziel in eine Aktion um
UND lösche das Ziel

P 4: WENN es ein Ziel gibt, eine Scheibe zu bewegen
UND
WENN diese Scheibe nicht bewegt werden kann,
DANN lösche das Ziel
UND nimm Dir als *neues* Ziel vor, die größte störende Scheibe zu dem Stab zu schaffen, der im *vorherigen* Ziel weder Ausgangs- noch Zielstab war.

Obwohl das System nur vier Produktionen enthält (vgl. a. Hasemer, 1984, S. 47), entspricht es in seiner Struktur einem Produktionssystem von Simon (1975), das mit sieben Produktionen die von Simon sogenannte *sophisticated perceptual strategy* verkörpert. Bei Simon wird zusätzlich noch ein Informationselement im Kurzzeitspeicher gehalten, das als *state* bezeichnet wird. Es signalisiert, ob ein Zug möglich ist. Wir haben hierauf verzichtet, weil dieses Element für unsere Zwecke überflüssig ist. In der dritten und vierten Produktion wird über Kontrollwissen jeweils die Möglichkeit eines Zuges festgestellt.

Die *sophisticated perceptual strategy* zeichnet sich gegenüber den anderen von Simon auf ihre kognitiven Implikationen hin untersuchten Strategien durch eine Reihe von Punkten aus, die sie für einen Problemlöser attraktiv machen. Sie erlaubt es u. a., beliebig komplexe Turmprobleme ohne größere Belastung des

Arbeitsspeichers zu lösen. Auch Fehler behindern nicht grundsätzlich die Fähigkeit, den Weg zur Lösung fortzusetzen. Das beschriebene Produktionssystem muß als Endprodukt einer Prozeduralisierung mit *chunks* höherer Ordnung (im Sinne von Roosenbloom & Newell, 1986) aufgefaßt werden. Wir werden, um dieses deutlicher zu machen, das formale Modell (s. Anhang 1) so formulieren, daß der *chunk* „zulässiger Zug" im Bedingungsteil von P3 im Sinne unseres Zielhierarchiekonzepts weiter spezifiziert wird. Dagegen wird der *chunk* „größte störende Scheibe" in P4 so belassen. In der Implementation wird daher die Auflösung dieses *chunks* nicht mit Regeln, sondern mit primitiven Operatoren unterhalb der Regelebene durchgeführt (s. Anhang 1).

Das Produktionssystem wirft Probleme auf, wenn es als Modell für kognitive Prozesse bei Personen gelten soll. Es beschreibt nur die *fehlerfreie* Informationsverarbeitung von Experten, und diese ist effizienter als das Problemlösen von Novizen. Karat hat daher ein Produktionssystem mit 10 Produktionen vorgeschlagen, das zwei stochastische Regelanwendungen enthält. Die Regeln gliedern sich in drei Gruppen: (a) Generiere Zugvorschläge, (b) bewerte Zugvorschläge und (c) führe zulässige Vorschläge aus.

Da wir hier jedoch einen *einfachen kleinen* in LISP geschriebenen Regelinterpreter mit Programmcode vorstellen wollen, haben wir uns auf unser kleines, deterministisches und effizientes Modell beschränkt. Für eine genaue Beschreibung der Produktionen, des Interpreters und des Codes sei auf Anhang 1 verwiesen.

Modellidentifikation und Modellüberprüfung

Da mathematisch statistische Parametermodelle und Produktionssysteme formale Modelle sind, übertragen sich auch auf die Regelmodelle Probleme, die bei den Parametermodellen hinreichend bekannt sind: Modellidentifikation, Parameterschätzung und Modellüberprüfung.

Das *Problem der Modellidentifikation* bezieht sich auf die Architektur des Systems. So wird hier die Zahl der Regeln und die Form der Bedingungen und Aktionen betrachtet. Regeln, in deren Aktionsteilen empirisch „sichtbare" Operatoren aufgerufen und nicht nur neue Ziele gesetzt werden, sind leichter zu identifizieren. Die Annahme von „nicht sichtbaren" Zielen im Arbeitsgedächtnis sollte daher einhergehen mit Annahmen über den Stand des prozeduralen Lernens (Anderson, 1983a, Kap. 6; Rosenbloom & Newell, 1986). Durch *chunk*-Bildung verschwinden Ziele zugunsten von expliziten autonomen Verknüpfungen zwischen Reiz- und Reaktionssymbolen. Eine Person könnte nach diesen Theorien keine Aussagen über Ziele verbalisieren, die in der durch den *chunk* ersetzten Zielhierarchie existierten.

Das *Problem der Parameterschätzung* wurde in diesem Artikel bisher noch nicht behandelt, weil wir keine stochastischen Regeln benutzt haben. Darunter verste-

hen wir Regeln, die entweder im Bedingungsteil Schwellenparameter oder im Aktionsteil Schalterparameter besitzen. Solche Regeln erlauben es, auf gleichartige Arbeitsspeicherkonfigurationen mit unterschiedlichen Übergängen im Problem-Verhaltensgraphen zu reagieren. Parameter wurden z. B. von Karat (1982) und Schmalhofer und Polson (1986) eingeführt, um nichtkonsistentes Verhalten von Personen zu beschreiben. In diesen Arbeiten werden aber zur Schätzung der Parameter keine Aussagen gemacht, die einer mathematisch statistischen Theorie genügen würden.

Zur *Modellüberprüfung* lassen sich eine Reihe von Aussagen machen, die empirisch getestet werden können. So können wir u. a. Vorhersagen treffen

a) über *Operatorsequenzen*, die Wege im Problem-Verhaltensgraphen beschreiben. Zum Matchen derartiger Sequenzen geben Card, Moran und Newell (1983, S. 190f.) ein Verfahren an, das eventuell vorhandene Lücken sowohl in der vom Modell vorhergesagten wie auch in der empirisch beobachteten Operatorsequenz berücksichtigt. Allerdings darf man nicht aus einer Operatorsequenz direkt auf ein Modell schließen, solange nicht ausgeschlossen ist, daß alternative Regelmodelle dieselbe Sequenz erzeugen können.

b) über *Operator-Latenz-Zeiten*. Darunter versteht man die Zeit zwischen zwei beobachtbaren Reaktionen einer Person. Diese Zeitintervalle können gedeutet werden, wenn das Regelmodell unterschiedliche Intervalle vorhersagen kann (Karat, 1982).

c) über *Zustandsübergänge*. Das Konzept des Problem-Verhaltensgraphen birgt den Vorteil, zu jedem Zustand mit Hilfe des Modells den Nachfolgezustand vorhersagen zu können. Teilt man die Übergänge in z. B. drei Güteklassen (auf die Lösung führende, zu der Lösung nicht hinführende und falsche Züge), kann man die Modellvorhersagen mit einfachen nichtparametrischen Tests überprüfen.

d) über *Verbalisierungen*. Nach der *chunk*-Theorie von Rosenbloom und Newell ersetzen *chunks* als automatisierte Prozeduren Zielhierarchien. Es wird dann unwahrscheinlich, daß derartig ersetzte Ziele noch verbalisiert werden können. Weitere Überlegungen hierzu finden sich bei Ericsson und Simon (1984).

e) über die *Bearbeitungsdauer eines Problems*. Diese Zeit kann als Funktion der Zykluszahl von Regelanwendungen *(recognize-act-cycles)* angesehen werden (Polson & Kieras, 1985). Voraussetzung dabei ist natürlich, daß sich während der Bearbeitung die Regeln nicht durch Kompilierung (Anderson, 1983a; Neves & Anderson, 1981) oder *chunk*-Bildung wesentlich verändern.

f) über *Transfereffekte*. Die Lernzeit für die Beherrschung neuer Aufgaben und Probleme ist abhängig von der Zahl der neu für diese Aufgabe zu lernenden spezifischen Produktionen (Polson & Kieras, 1985). Produktionen, die in der alten und neuen Aufgabe gemeinsam enthalten sind, brauchen nicht neu hinzugelernt werden *(common-rules*-Hypothese nach Polson, Muncher & Engelbeck, 1986).

Damit haben wir hier eine Reihe von Möglichkeiten genannt, die empirische Tragfähigkeit von Regelmodellen zu prüfen. Sie stehen damit in keiner Weise hinter den bekannten mathematisch-statistischen Parametermodellen zurück.

Deduktive Informationsverarbeitung mit zielorientierten, rückwärtsverkettenden Regelsystemen

In den beiden vorangegangenen Abschnitten haben wir uns mit vorwärtsverkettenden Regelsystemen befaßt. Ihrem Charakter nach waren sie stärker daten- als zielorientiert. Ziele wurden im Arbeitsspeicher eingeführt, wenn die externe Situation und ihre Repräsentation mittels Daten keine eindeutigen Hinweise auf den weiteren Verlauf der Aktionen zuließ.

In diesem Abschnitt wollen wir uns den zielorientierten Regelsystemen zuwenden. Wir erwarten dabei, daß sich mit ihnen bestimmte empirisch beobachtbare Phänomene besonders gut beschreiben und eventuell modellieren lassen. Wir hoffen, damit etwa die Beantwortung von Fragen mittels deduktiver Prozesse oder das Problemlösen mittels Aufbau von Unterzielhierarchien erfassen zu können. Wir wollen unsere Überlegungen am Problem der Fragebeantwortung entwickeln und stützen uns hier auf die Beispiele, die Klix (in diesem Band) bei der Diskussion semantischer Netze benutzt hat.

Zunächst werden einige grundlegende Begriffe und Konzepte am Beispiel der Ereignisbeschreibungen „Lehrsituation" und „Behandlungssituation" (Klix, in diesem Band, Abb. 2) eingeführt.

Das die beiden Situationen beschreibende rückwärtsverkettende Regelsystem ist in Tab. 18.1 abgebildet.

Wir bezeichnen das Regelsystem als *logisches Programm*. Das wird durch eine Reihe von Eigenschaften gerechtfertigt, auf die wir teilweise noch näher eingehen wollen.

Das Programm setzt sich zusammen aus 18 Faktoren, die die zwei Situationen „Lehren" und „Behandeln" beschreiben, und 6 Regeln, die die Umsetzung von quasi-natürlichsprachlichen Fragen in ihre propositionale Repräsentation bewirken. So können dem System u. a. folgende Fragen gestellt werden (die Antworten sind rechts daneben geschrieben):

„(WEM LEHREN _Antwort)?"	„_Antwort = Schüler"
„(WOMIT BEHANDELN _Antwort)?"	„_Antwort = nicht-schulmedizinisch"
	„_Antwort = Skalpell-Narkose"

Die zweite Frage initiiert im Gegensatz zur ersten gleich zwei Antworten.

Bevor wir auf die Semantik des Regelsystems und seinen Interpreter eingehen, wollen wir kurz die Syntax der hier vorgestellten Regelsysteme beschreiben. Wir verwenden dazu die schon aus dem Beitrag von Opwis (in diesem Band) bekannten Backus-Naur-Regeln (BNF-Syntaxregeln). Die Regeln werden folgendermaßen gelesen. Das in spitze Klammern ⟨ ⟩ links vom Ersetzungszeichen ::= geschriebene Symbol wird ersetzt durch die rechte Seite der Regel. Alternativen werden durch den senkrechten Oder-Strich | gekennzeichnet. Die anderen Sym-

Tab. 18.1.: Formalisierung der Ereignisbeschreibungen „Lehrsituation" und „Behandlungssituation"

```
/* zu Klix, in diesem Band, Abb. 2.2., S. 26, "Ereignisgebundene
                                                    Begriffsbeziehungen"*/
((LEHREN   Fin   Wissen))
((LEHREN   Fin   Können))
((LEHREN   Loc   Schule))
((LEHREN   Ht1   Lehrer))
((LEHREN   Ht2   Schüler))
((LEHREN   Obj   Fach))
((BEHANDELN   Fin   Diagnose-für-Eingriff))
((BEHANDELN   Fin   Operation))
((BEHANDELN   Loc   Klinik))
((BEHANDELN   Loc   Unfallklinik))
((BEHANDELN   Ht1   Arzt))
((BEHANDELN   Ht1   Chirurg))
((BEHANDELN   Ht2   Patient))
((BEHANDELN   Ht2   Unfallpatient))
((BEHANDELN   Obj   Krankheit))
((BEHANDELN   Obj   Beinbruch))
((BEHANDELN   Instr   nicht-schulmedizinisch))
((BEHANDELN   Instr   Skalpell-Narkose))
((WOZU   _Handlung   _Ziel)
          (_Handlung   Fin   _Ziel))
((WO      _Handlung   _Ort)
          (_Handlung   Loc   _Ort))
((WER     _Handlung   _Akteur)
          (_Handlung   Ht1   _Akteur))
((WEM     _Handlung   _Adressat)
          (_Handlung   Ht2   _Adressat))
((WAS     _Handlung   _Gegenstand)
          (_Handlung   Obj   _Gegenstand))
((WOMIT   _Handlung   _Mittel)
          (_Handlung   Instr   _Mittel))
```

bole (wie z. B. die runden Klammern) sind terminal. Das bedeutet, daß sie ihrerseits nicht weiter ersetzt werden können.

Die Syntaxregeln lauten dann:

⟨Klausel⟩ ::= ⟨Faktum⟩ | ⟨Regel⟩
⟨Faktum⟩ ::= (⟨Ziel⟩)
⟨Regel⟩ ::= (⟨Kopf⟩ ⟨Körper⟩)
⟨Kopf⟩ ::= ⟨Ziel⟩
⟨Körper⟩ ::= ⟨Ziel⟩ {⟨Ziel⟩} {} bedeutet null oder mehrmalige Wiederholung
⟨Ziel⟩ ::= (⟨Relationsname⟩ {⟨Argument⟩})
⟨Argument⟩ ::= ⟨Konstante⟩ | ⟨Variable⟩
⟨Relationsname⟩ ::= LEHREN | BEHANDELN | WOZU | WO | WER | WEM | WAS | WOMIT
⟨Konstante⟩ ::= Fin | Wissen | Können | . . . | Skalpell-Narkose
⟨Variable⟩ ::= _Handlung | _Ziel | _Ort | _Akteur | _Adressat | _Gegenstand | _Mittel
⟨Frage⟩ ::= ⟨Ziel⟩?

Wir fassen Fakten und Regeln unter dem Begriff Klausel zusammen. Ein Faktum ist eine Regel ohne einen Körper. Der Kopf der Regel steht für die Konsequenz (WENN-Komponente) und der Körper für eine Folge von Zielen (DANN-Komponente). Im Gegensatz zu den vorangegangenen Abschnitten werden die Regeln hier in umgekehrter Reihenfolge niedergeschrieben. Dadurch wird die rückwärts verkettende Arbeitsweise des die Regeln abarbeitenden Interpreters erleichtert.

Ein Ziel kann in einem Fakt, einer Regel, aber auch einer Frage vorkommen. Es setzt sich zusammen aus einem Relationsnamen und Null oder mehreren Argumenten, die hier Variable oder Konstante sein können. Unter einer Variablen verstehen wir ein abstraktes Objekt, das Konstante als Werte annehmen kann. Die Namen der Variablen sollen dabei auf zu erwartende Werte hinweisen. In unserem Beispiel kommen z. B. die Ziele (LEHREN Fin Wissen), (WEM _Handlung _Adressat) oder (_Handlung Obj _Gegenstand) vor.

Die Bedeutung des Regelsystems (oder des Programms) besteht in der Menge aller Folgerungen (d. h. Antworten), die sich aus ihm ableiten lassen.

Fakten stehen für die Behauptung, daß ein Ziel ohne jede Vorbedingung wahr ist. Fragen dagegen initiieren einen Antwort- oder Beweisprozeß, der zeigen soll, ob das in die Frage gekleidete Ziel wahr ist. Der Beantwortungsprozeß gliedert sich in eine Reihe von Spezialfällen, in denen die Antwort sofort gegeben werden kann und einen allgemeinen Fall, in dem Such-, Mustervergleichs- und Deduktionsprozesse in Gang gesetzt werden.

Ist z. B. das Ziel in der Frage mit dem Ziel in einem Faktum identisch, wird die Frage mit „Ja" beantwortet. Andere Spezialfälle ergeben sich durch die Berücksichtigung von Variablen. Eine Variable in einer Frage wird als existenzquantifiziert aufgefaßt. Der Frage äquivalent ist die Formulierung: „Gibt es ein ... so daß ... gilt?" Entspricht das Muster des Ziels einer derartigen Frage dem Ziel in einem Faktum und können Frage und Ziel im Faktum durch Ersetzung der Variablen durch Konstante einander gleich gemacht werden (geschrieben ⟨Variable⟩ / ⟨Konstante⟩), wird die Frage ebenfalls mit „Ja" beantwortet. Als zusätzliche Information erhält man noch eine Aussage darüber, welchen Wert die Variable angenommen hat. Wird eine Variable durch eine Konstante ersetzt, liegt eine Variablensubstitution vor. Es wird so eine Instantiierung der Variablen vorgenommen. Ziele ohne Variable wollen wir auch *Grund*ziele nennen. Sie kann man durch Instantiierung erhalten.

Ein dritter Spezialfall, der aber hier in unseren Regelsystemen nicht vorkommt, betrifft Variable in den Fakten. Diese sind allquantifiziert („Für alle ... gilt ...!"). Enthält eine Frage keine Variable (d. h. sie ist eine Grundfrage) und kann ein Ziel in einem Faktum durch Variableninstantiierung identisch mit dem Ziel in der Frage gemacht werden, wird die Frage mit „Ja" beantwortet.

Enthält das Programm Regeln, gestaltet sich die Beantwortung einer Frage aufwendiger. Für die weiteren Überlegungen wollen wir jetzt eine Kurznotation einführen. Ein Faktum ist eine Regel ohne Vorbedingung, und wir schreiben

$$A \leftarrow$$

Eine Regel wird geschrieben als

$$A \leftarrow B_1, B_2, \ldots, B_i, \ldots, B_n \text{ mit } 1 < n$$

Das A steht für das Ziel des Kopfes (DANN-Komponente) und die B_i stehen für die Ziele im Körper (WENN-Komponente der Regel). Eine Frage schreiben wir dann als

$$\leftarrow Q$$

In dem Regelsystem „Ereignisbeschreibung" befinden sich sechs Regeln. Der Gültigkeitsbereich einer Variablen soll lokal auf die Regel beschränkt sein. Darüber hinaus sind die Variablen in den Regeln allquantifiziert. Die Regeln lassen zwei Lesarten zu. Die *prozedurale* Interpretation lautet z. B. für die erste Regel: „Um die Frage zu beantworten, welchem Ziel (wozu) eine Handlung dient, beantworte zuerst die Frage, ob es eine Relation zwischen Handlung, Ziel und dem Fin-Indikator in der Wissensbasis gibt." Die *deklarative* Interpretation lautet: „Für alle Handlungen und Ziele gilt: Wenn es eine Relation zwischen einer Handlung, einem Ziel und dem Fin-Indikator in der Wissensbasis gibt, dann dient die Handlung dem Ziel."

Diese Ausführungen mögen jetzt noch etwas gekünstelt klingen, da die Regeln in diesem Beispiel nur einem Wechsel der Sprachebenen (Primitivdeutsch vs. propositionalisiertes Wissen) dienen. Im nächsten Beispiel werden wir jedoch Regeln formulieren (Tab. 18.2), bei denen der Implikationscharakter stärker durchdringt.

Allgemein läuft die Beantwortung einer Frage unter Einschaltung von Regeln auf der Basis des verallgemeinerten modus-ponens-Schlusses ab. Eine Frage, die ein existentiell quantifiziertes Ziel Q beinhaltet und die nicht mit den o.a. Spezialfällen beantwortet werden kann, setzt einen Suchprozeß in Gang. Findet sich eine Regel

$$A \leftarrow B_1, B_2, \ldots, B_i, \ldots, B_n$$

so daß A und Q durch Variablensubstitutionen δ, θ identisch gemacht werden können, so daß

$$A' = A\,\delta = Q\,\theta = Q' \qquad \begin{array}{l} A' \text{ ist Instanz von } A \\ Q' \text{ ist Instanz von } Q \end{array}$$

gilt, wird das alte Ziel der Frage Q durch die konjunktiv verknüpften Ziele

$$B'_1, B'_2, \ldots, B'_i, \ldots, B'_n \text{ (mit } B'_i = B_i\theta)$$

ersetzt. Die Frage Q kann mit „Ja" beantwortet werden, wenn die B'_i aus dem Programm ableitbar sind. Statt *einer* Frage Q sind jetzt die n-Teilfragen B'_i zu beantworten.

Wir wollen das an der Frage Q = (WEM LEHREN _Antwort)? erläutern. Zu Q paßt die Regel

((WEM _Handlung _Adressat)(_Handlung Ht2 _Adressat)),

wenn im Kopf der Regel die Variablensubstitutionen δ = {_Handlung / LEHREN, _Adressat / _Antwort} vorgenommen werden. Das alte Ziel Q wird jetzt ersetzt durch das instantiierte Ziel im Körper der Regel (LEHREN Ht2 _Antwort).

Zu diesem Ziel gibt es ein entsprechendes Faktum in der Wissensbasis, wenn die Variablensubstitution θ = {_Antwort / Schüler} vorgenommen wird. Die Frage kann daher mit „Ja" und _Antwort = Schüler beantwortet werden.

Das zweite Beispiel bezieht sich auf eine *Abstraktionshierarchie*. Bei Klix (in diesem Band, Abb. 1). finden sich aus empirischen Daten abgeleitete Differenzen im Abstraktionsniveau zwischen merkmalsbestimmten Begriffsbeziehungen. Wir haben in Abb. 18.5. einen entsprechenden Abstraktionsbaum rekonstruiert.

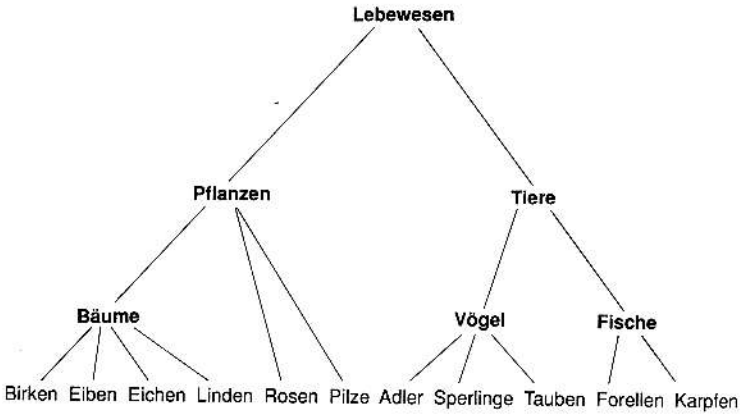


Abb. 18.5.: Aus merkmalsbestimmten Begriffsbeziehungen abgeleitete Abstraktionshierarchie (vgl. Klix, in diesem Band, Abb. 2.1., S. 25)

Das diese Abstraktionshierarchie generierende Regelsystem ist in Tab. 18.2. angegeben. Das System besteht aus 16 Fakten und 6 Regeln. Die MAXIMUM-Regel ist rein technischer Natur und dient nur zur Bestimmung eines Maximums. Wir haben bei den Fakten durch den Plural angedeutet, daß wir Klassen von Lebewesen und nicht Individuen betrachten. Auf die Frage (UNÄHNLICHKEIT Forellen Eichen _Antwort)? antwortet das System mit _Antwort = 3. Die gleiche Antwort kommt auf die Frage (UNÄHNLICHKEIT Adler Pilze _Antwort)? Wir können also die numerischen Einträge in Abb. 1 von Klix (in diesem Band) reproduzieren. Der hypothetische Mechanismus für die Generation von Unähnlich-

Tab. 18.2.: Formalisierung einer Abstraktionshierarchie und Generierung von Unähnlichkeitsurteilen

```
/* zu Klix, in diesem Band, Abb. 2.1., S. 25, "Merkmalsbestimmte
((SIND   Birken   Bäume))                                    Begriffsbeziehungen"*/
((SIND   Eiben   Bäume))
((SIND   Eichen   Bäume))
((SIND   Linden   Bäume))
((SIND   Rosen   Pflanzen))
((SIND   Pilze   Pflanzen))
((SIND   Bäume   Pflanzen))
((SIND   Pflanzen   Lebewesen))
((SIND   Adler   Vögel))
((SIND   Sperlinge   Vögel))
((SIND   Tauben   Vögel))
((SIND   Forellen   Fische))
((SIND   Karpfen   Fische))
((SIND   Vögel   Tiere))
((SIND   Fische   Tiere))
((SIND   Tiere   Lebewesen))
((UNTERGEORDNET   _Unterbegriff   _Oberbegriff     1)
        (SIND   _Unterbegriff   _Oberbegriff))
((UNTERGEORDNET   _Unterbegriff   _Oberbegriff   _Stufen)
        (SIND   _Unterbegriff   _direkter-Oberbegriff)
        (UNTERGEORDNET   _direkter-Oberbegriff   _Oberbegriff   _Stufen-1)
        (+   1   _Stufen-1   _Stufen))
((UNÄHNLICHKEIT   _Begriff1   _Begriff2   _Stufen)
        (UNTERGEORDNET   _Begriff1   _Begriff2   _Stufen)/)
((UNÄHNLICHKEIT   _Begriff2   _Begriff1   _Stufen)
        (UNTERGEORDNET   _Begriff1   _Begriff2   _Stufen)/)
((UNÄHNLICHKEIT   _Begriff1   _Begriff2   _Stufen)
        (UNTERGEORDNET   _Begriff1   _Oberbegriff   _Stufen1)
        (UNTERGEORDNET   _Begriff2   _Oberbegriff   _Stufen2)
        (MAXIMUM   _Stufen1   _Stufen2   _Stufen)/)
/*------------------------ Primitive -------------------------------*/
((MAXIMUM   _Zahl1   _Zahl2   _maxZahl)
        (IF   (≥   _Zahl1   _Zahl2)
              ((EQ   _maxZahl   _Zahl1))
             ((EQ   _maxZahl   _Zahl2))))
```

keitsurteilen als Funktion von Beziehungen in der Abstraktionshierarchie ist transparent in den Regeln niedergelegt. Für die Bestimmung der Unähnlichkeitsgrade benötigen wir die Relation UNTERGEORDNET. Die Bedeutung dieser Relation findet sich in zwei Regeln. Die erste Regel besagt, daß zwischen zwei Begriffen eine UNTERGEORDNET-Relation mit Hierarchiestufendifferenz 1 besteht, vorausgesetzt, es findet sich ein Faktum (SIND _Unterbegriff _Oberbegriff) in der Datenbasis. Anderenfalls wird die zweite Regel herangezogen. Sie prüft, ob es eine UNTERGEORDNET-Relation zwischen zwei Begriffen mit einer zunächst noch unbekannten Stufendifferenz dieser Begriffe gibt. Gibt es einen direkten Oberbegriff zu dem Unterbegriff (wird geprüft mit (SIND _Unterbegriff _direkter–Oberbegriff)) und eine UNTERGEORDNET-Relation von diesem noch zu findenden direkten Oberbegriff zum ursprünglichen Oberbegriff

mit Stufendifferenz _Stufen-1, wird die ursprünglich interessierende Stufendifferenz _Stufen durch Erhöhung von _Stufen-1 um 1 bestimmt.

Die Berechnung der Unähnlichkeit gliedert sich in drei Fälle. Dementsprechend haben wir auch drei Regeln formuliert, von denen nur jeweils eine zur vollständigen Anwendung kommt (gekennzeichnet durch das sogenannte Schnittsymbol / am Ende der drei Regeln). Die ersten beiden Regeln können wir verbal zusammenfassen zu: „Zwischen zwei Begriffen besteht eine numerisch quantifizierbare Unähnlichkeit, die sich aus der Stufendifferenz in der Abstraktionshierarchie herleiten läßt, wenn die beiden Begriffe in einer UNTERGEORDNET-Relation zueinander stehen." Damit sind Unähnlichkeitsurteile zwischen Birken–Bäume und Bäume–Pflanzen möglich. Allerdings wäre ein Unähnlichkeitsurteil zwischen Eichen–Linden oder Pilze–Adler noch nicht möglich. Dafür nehmen wir hypothetisch an, daß Unähnlichkeitsurteile noch nach einer dritten Regel durchgeführt werden. Stehen zwei Begriffe nicht in einer UNTERGEORDNET-Relation zueinander, wird geprüft, ob es in der Abstraktionshierarchie einen für beide Begriffe gemeinsamen Oberbegriff gibt. Es wird die Stufendifferenz beider Begriffe zu diesem Oberbegriff bestimmt. Das Unähnlichkeitsmaß ergibt sich dann durch das Maximum der beiden Stufendifferenzen.

Jetzt sind wir auch in der Lage, die *Funktionsweise des abstrakten Regelinterpreters* zu beschreiben. Dazu führen wir noch ein paar Begriffe ein. Eine Liste von Zielen, die zur Beantwortung anstehen, soll Resolvente heißen. Die Grundinstanz einer Regel $A <- B_1, \ldots, B_n$ ist eine Regel, in der alle Ziele durch die gleiche Variablensubstitution θ zu Grundzielen gemacht wurden: $A' <- B'_1, \ldots, B'_n$ mit $A' = A\,\theta$ und $B'_i = B_i\theta$.

Der abstrakte Regelinterpreter funktioniert nun folgendermaßen:

1. Initialisiere die Resolvente mit der Grundinstanz der Frage Q. Enthielt die Frage Q konjunktiv verknüpfte Teilfragen $A_1, \ldots, A_k$, enthält die Resolvente jetzt die Grundziele $A'_1, \ldots, A'_k$ mit $A'_i = A_i\theta$.
2. Solange die Resolvente nicht leer ist, mache folgendes:

2.1 Wähle ein Ziel A'_i aus der Resolvente $(1 \leqslant i \leqslant k)$
und

2.2 Wähle eine Grundinstanz einer Klausel
a) $A' <—$

oder

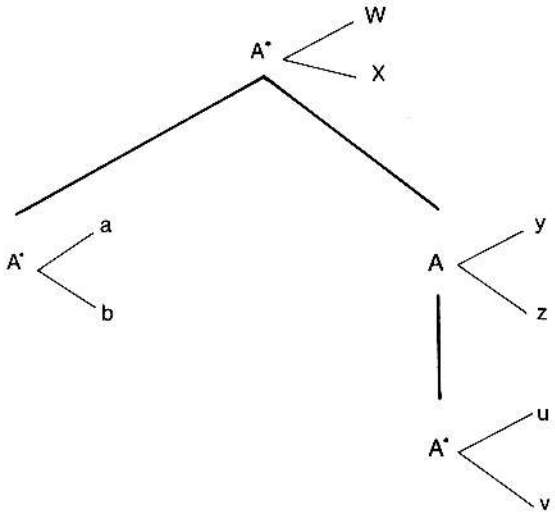
b) $A' <— B'_1, \ldots, B'_n$ mit $(1 \leqslant n)$
aus dem Regelsystem, so daß Ziel und Kopf gleich sind:
$A'_i = A'$. Gibt es keine solche Klausel, verlasse die Schleife. Sonst

2.3 Verändere die Resolvente zu
a) $A'_1, \ldots, A'_{i-1}, A'_{i+1}, \ldots, A'_n$

oder

b) $A'_1, \ldots, A'_{i-1}, B'_1, \ldots, B'_n, A'_{i+1}, \ldots A'_n$

3. Ist die Resolvente leer, antworte mit „Ja" und gebe die Variablensubstitution θ als zusätzliche Antwort. Im anderen Fall antworte mit „Nein".



Frageschema: (Vergleiche <1. Konzept A> <2. Konzept A'> <Antwort>)

Beispiele: (Vergleiche (w x y z) (w x) _Antwort)
(Vergleiche (w x y z) (w x y z u v) _Antwort
(Vergleiche (w x y z) (w x a b) _Antwort
(Vergleiche (w x y z) (w x y z) _Antwort

Abb. 18.6.: Abstraktionshierarchie mit Standardkonzept A und alternativen Vergleichskonzepten A' (vgl. a. Klix, in diesem Band, Abb. 2.5., S. 33)

Das dritte Beispiel bezieht sich auf ein Regelsystem, das in der Lage ist, Unter- und Oberbegriffsrelationen sowie Nebenordnungen von Begriffen aus den Merkmalslisten der Attribute zu bestimmen. Das Regelsystem kann also Abstraktionshierarchien mit Merkmalsvererbung aufbauen. Wir haben das Beispiel von Klix (in diesem Band, Abb. 5) teilweise in Abb. 6 graphisch aufbereitet. Wir haben ein Standardkonzept A mit den Merkmalen (w x y z). Das Konzept oder der Begriff A wird mit dem Konzept A' verglichen. Umfaßt A' nur die Merkmale (w x), ist A' der Oberbegriff und A der Unterbegriff. Dabei erbt A von A' die beiden Begriffen gemeinsamen Merkmale (w x). Am Konzeptknoten A werden nur die konzeptspezifischen Merkmale (y z) angelagert.

Für den Vergleichsbegriff A' werden dann alle Möglichkeiten der Ober-, Neben- und Unterordnung zu A durchgespielt. Die Abstraktionshierarchie geht dabei von größtmöglicher Speicherökonomie bei der Ablegung von Merkmalen aus. Das Regelsystem, das die Frage (VERGLEICHE ⟨1.Konzept⟩ ⟨2.Konzept⟩ _Antwort)? mit der Ober-, Neben- oder Unterordnung der Konzepte beantwortet, findet sich in Tab. 18.3.

Tab. 18.3.: Regelsystem zur Prüfung zweier Konzepte auf Ober-, Neben- oder Unterordnung

```
/*************** Abgleich von Attributmengen ******************/
/*1*/
((Vergleiche  _Begriff1  _Begriff2   _Antwort)
    (Durchschnitt  _Begriff1  _Begriff2  _Gemeinsamkeiten)
    (EQ  _Gemeinsamkeiten  ())
    (EQ  _Antwort  "keine Ordnung auf den Begriffen"))
/*2a*/
((Vergleiche  _Begriff1  _Begriff2   _Antwort)
    (Differenz  _Begriff1  _Begriff2   _Differenz12)
    (Differenz  _Begriff2  _Begriff1   _Differenz21)
    (EQ  _Differenz12  ())
    (EQ  _Differenz21  ())
    (EQ  _Antwort  "identische Begriffe"))
/*2b*/
((Vergleiche  _Begriff1  _Begriff2   _Antwort)
    (Differenz  _Begriff2  _Begriff1   _Differenz21)
    (Differenz  _Begriff1  _Begriff2   _Differenz12)
    (EQ  _Differenz21  ())
    (EQ  _Differenz12  ())
    (EQ  _Antwort "identische Begriffe")/)
/*3a*/
((Vergleiche  _Begriff1  _Begriff2  _Antwort)
    (Differenz  _Begriff1  _Begriff2   _Differenz12)
    (EQ  _Differenz12  ())
    (EQ  _Antwort  "Begriff1 ist Ober- u. Begriff2 ist Unterbegriff"))
/*3b*/
((Vergleiche  _Begriff1  _Begriff2  _Antwort)
    (Differenz  _Begriff2  _Begriff1  _Differenz21)
    (EQ  _Differenz21  ())
    (EQ  _Antwort  "Begriff1 ist Unter- u. Begriff2 ist Oberbegriff"))
/*4a*/
((Vergleiche  _Begriff1  _Begriff2  _Antwort)
    (Differenz  _Begriff2  _Begriff1  _Differenz21)
    (EQ  _Differenz21  ())
    (EQ  _Antwort   "Begriff1 ist Unter- u. Begriff2 ist Oberbegriff"))
/*4b*/
((Vergleiche  _Begriff1  _Begriff2  _Antwort)
    (Differenz  _Begriff1  _Begriff2  _Differenz12)
    (EQ  _Differenz12  ())
    (EQ  _Antwort  "Begriff1 ist Ober- u. Begriff2 ist Unterbegriff"))
/*5*/
((Vergleiche  _Begriff1  _Begriff2  _Antwort)
    (Durchschnitt  _Begriff1  _Begriff2  _Gemeinsamkeiten)
    (NOT  EQ  _Gemeinsamkeiten  ())
    (Differenz  _Begriff2  _Begriff1   _Differenz21)
    (Differenz  _Begriff1  _Begriff2   _Differenz12)
    (NOT  EQ  _Differenz21  ())
    (NOT  EQ  _Differenz12  ())
    (EQ  _Antwort  "Begriff1 und Begriff2 sind nebengeordnet")/)
/*************** primitive Relationen *************************/
((Durchschnitt    ()      _Liste2      ())   /)
((Durchschnitt    _Liste1     ()       ())   /)
((Durchschnitt     (_first1 | _rest1)     _Liste2      (_first1 | _rest1Liste2))
     (ON  _first1  _Liste2)
     (Durchschnitt     _rest1     _Liste2     _rest1Liste2)  /)
((Durchschnitt    (_first1 | _rest1)    _Liste2     _rest1Liste2)
     (Durchschnitt    _rest1    _Liste2    _rest1Liste2)  /)
((Differenz    ()    _Liste2     ())   /)
((Differenz  _Liste1  ()       _Liste1 ))
((Differenz  (_first1 | _rest1)  _Liste2  (_first1 | _rest1-Liste2))
     (NOT  ON  _first1  _Liste2)
     (Differenz  _rest1  _Liste2  _rest1-Liste2)  /)

((Differenz  (_first1 | _rest1)  _Liste2   _rest1-Liste2)
     (Differenz  _rest1  _Liste2  _rest1-Liste2)  /)
```

Den Ablauf der Entscheidungsfindung haben wir in Abb. 18.7 dargestellt.

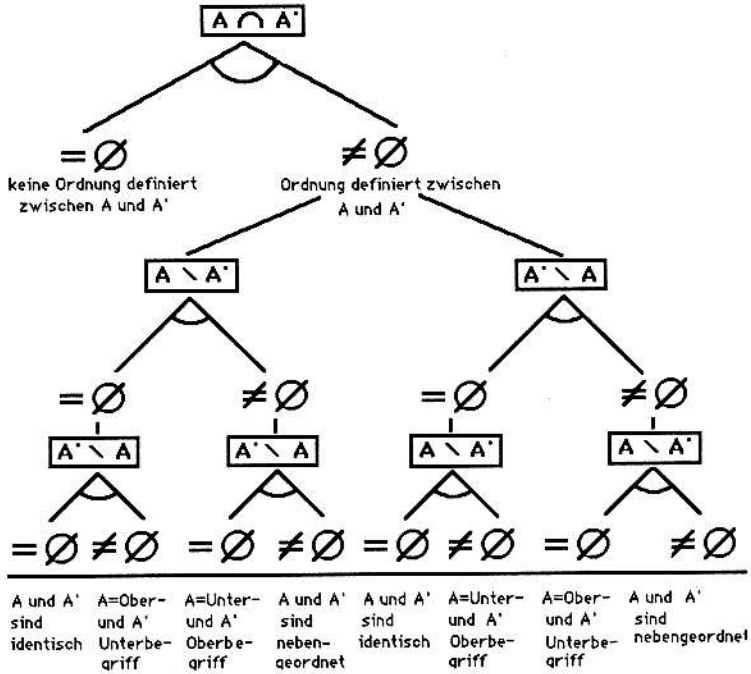


Abb. 18.7.: Entscheidungsbaum zur Prüfung auf Ober-Unter- oder Unter-Ober-Konzeptrelation oder auf Nebenordnung

Im Regelsystem von Tab. 18.3. sind einige neue Details aufgeführt, die der Verarbeitung von Merkmalslisten dienen. Wir wollen sie hier kurz erklären. Genaueres läßt sich z. B. bei Clark und McCabe (1984) nachlesen.

Die Relation „Durchschnitt" umfaßt vier Regeln, mit denen alle Sonderfälle bei der Bestimmung von gemeinsamen Merkmalen zweier Konzepte abgedeckt werden. Es kommt nur jeweils eine der vier Regeln erfolgreich zum Abschluß. Die ersten zwei Regeln sagen aus, daß der Durchschnitt einer nichtleeren Liste mit einer leeren Liste leer ist. In der dritten Regel wird von Listenmustern (_X | _Y) und dem Prädikat ON Gebrauch gemacht.

Lautet die Frage z. B. (Durchschnitt w x y z) (w x) _Antwort)?, finden die beiden ersten Regeln keine Anwendung, weil weder die erste noch die zweite Merkmalsliste leere Listen (nämlich =()) sind. Die Frage und der Kopf der 3. Regel können jedoch durch zwei Variablensubstitutionen δ und θ gleich gemacht werden:

δ = {_first1 / w, _rest1 / (x y z), _Liste2 / (w x)}
θ = {_Antwort / (w | _rest1Liste2}

Die ursprüngliche Resolvente wird dann ersetzt durch zwei Ziele: (ON w (w x)) und (Durchschnitt (x y z) _rest1Liste2). Wir haben das Ursprungsproblem auf ein einfaches (ON . . .) und ein vereinfachtes (Durchschnitt (x y z). . .) zurückgeführt. Die vierte Regel brauchen wir, wenn das erste Listenelement der ersten Merkmalsliste nicht in der zweiten Liste enthalten ist. Dann wird einfach der Durchschnitt der um das erste Element verkürzten ersten Liste mit der zweiten Liste ermittelt.

In entsprechender Weise werden die konzeptspezifischen Merkmalslisten mit der Relation „Differenz" berechnet. Diese beiden technischen Relationen werden in den Regeln 1–5 als Primitive benutzt. In diesen Regeln taucht noch die Relation EQ auf. Sie kann zur Prüfung auf Gleichheit von Konstanten verwendet werden. Enthalten die Argumente dagegen Variable, findet eine Variablensubstitution mit dem Ziel statt, beide Argumente identisch abzugleichen. Das wird beim letzten Ziel jeder Regel zur Generierung der Antwort benutzt.

Wir haben unsere Regelsysteme als logische Programme konzipiert und einen *abstrakten Interpreter* angegeben, der *die Rolle des Informationsverarbeiters* einnimmt. Dem Interpreter wurde dabei freigestellt, welche Ziele und welche Regel er auswählt. Ferner wurde über die konkrete Verfahrensweise bei den Variablensubstitutionen keine Angaben gemacht. Wir sind nur davon ausgegangen, daß der Interpreter eine „Ja"-Antwort findet, wenn sie sich prinzipiell finden läßt.

Für praktische Computersimulationen müssen diese Freiheiten jedoch eingeschränkt werden. Wir haben für konkrete Implementationen die Programmiersprache PROLOG gewählt (Clark & McCabe, 1984; Colmerauer, 1985; Cohen, 1985). Auf die Einschränkung von PROLOG hinsichtlich der Ziel- und Regelauswahl, wollen wir hier aber nur kurz eingehen und verweisen auf die Literatur.

Repräsentation von semantischen Netzen: Zu den Ansätzen von Quillian und Klix

Die Theorie semantischer Netze wurde mit einem Artikel von Quillian (1968) in dem für die Computermodellierung einflußreichen Buch „Semantic Information Processing" von Minsky (1968) begründet. Quillian formuliert ein netzartiges nichthierarchisches Wissensrepräsentationsmodell. Ausschnitte des Netzes repräsentieren inhaltliche Konzepte und werden Scheiben *(planes)* genannt. Aebli (1981, Bd. II, S. 252) spricht hier auch von Rahmen. Für jede Scheibe gibt es einen *type*-Knoten, der die Rolle einer Wortmarke für das Konzept einnimmt. Die anderen Knoten in der Scheibe werden *token* genannt. Sie charakterisieren die übergeordneten *types*. Letztere definieren sich dann wechselseitig.

Wegweisend für spätere Theorie war außer der *type-token*-Unterscheidung, die u. a. von Norman und Rumelhart (1978) und Kintsch (1974) aufgegriffen wurde, die Einführung von sich ausbreitenden Aktivationshüllen oder -wellen (Quillian,

1968, S. 249). Darunter ist folgendes zu verstehen: Versucht eine Frage, zwei Konzepte miteinander zu verbinden („Ist ein *Zebra* ein *Pferd*?"), laufen von beiden Konzepten („Pferd" und „Zebra") Aktivierungswellen aus. Schneiden sich die Wellen unterhalb einer oberen Zeitschranke, wird die Frage mit „Ja" beantwortet. Dieses hier zum erstenmal vorgeschlagene Aktivierungskonzept ist auch heute noch zentraler Bestandteil vieler Modelle zum semantischen Gedächtnis (Pollack & Waltz, 1986).

Von der psychologischen Forschung wurde aber erst die zweite Theorie von Quillian, die er zusammen mit Collins (Collins & Quillian, 1969) veröffentlicht hat, aufgegriffen. Sie ist wesentlich einfacher gehalten als die erste Theorie und erlaubt es auch ohne Computersimulation, empirische Vorhersagen zu machen. Aebli (1981, Bd. II, S. 251) stuft sie als „mittelalterlich" bzw. „aristotelisch-scholastisch-empiristisch" ein. Wir finden das Grundkonzept jedoch flexibel und prinzipiell erweiterungsfähig.

Die Theorie von Collins und Quillian (1969) bezieht sich auf eine Abstraktionshierarchie oder Begriffspyramide (s. z. B. Abb. 18.8.)

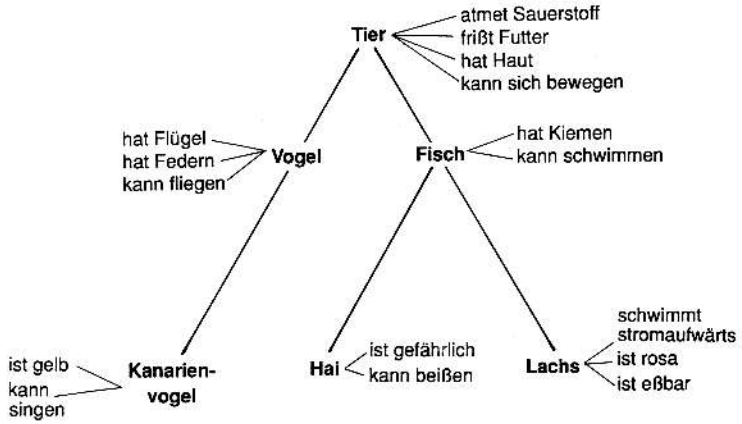


Abb. 18.8.: Abstraktionshierarchie mit Merkmalsvererbung nach Collins & Quillian (1969)

Es gilt die Annahme, daß sich Unter- bzw. Oberbegriffe als Knoten in einem Baum anordnen lassen. Eigenschaften werden an dem höchsten Knoten der Knotenmenge, für die sie Gültigkeit haben sollen, angelagert. Die Eigenschaften vererben sich auf die unteren Knoten und gelten damit für die entsprechenden Unterbegriffe. Bestimmend ist die Vorstellung *kognitiver Ökonomie* (Smith, 1978): Informationen werden nichtredundant abgespeichert und vererben sich im Bedarfsfall. Die Kanten des Baums werden durch die Relationen IST-EIN, KANN, HAT etc. gebildet. Die Zahl der Kanten, die bei einem Weg von einem Knoten zu einem anderen passiert werden müssen, bestimmen die *semantische Distanz* zweier Informationen. Die Reaktionszeit zur Beantwortung von Fragen der Art „Kanarienvö-

gel können fliegen! Wahr oder falsch?" soll wiederum von der semantischen Distanz abhängen. Demnach müßte die Reaktionszeit auf die „Kanarienvogelfrage" kürzer sein als auf die nachstehende Frage „Haie atmen! Wahr oder falsch?"

Die verschiedenen Annahmen von Collins und Quillian wurden in den folgenden Jahren von einer Reihe von Autoren kritisiert. So ist Smith (1978) der Ansicht, daß der entscheidende Faktor für die Schnelligkeit der Fragebeantwortung nicht die Zahl der Kanten zwischen den Knoten, sondern die Relationsstärke ist. Die Relationsstärke läßt sich dabei mit neuronaler Bahnung interpretieren. Im Modell sollte also die Kantenbewertung und nicht die Kantenzahl die Schnelligkeit der Suchprozesse steuern.

Der zweite Punkt, an dem sich Kritik entzündete (Smith, 1978; Wender, Colonius & Schulze, 1980, S. 13), bezog sich auf die nicht explizierte Möglichkeit der „Rückrechnung" von Unter-Oberbegriffsrelationen aus den Eigenschaften, die den Konzepten zugeordnet werden.

Widerspruch erregte auch die Annahme einer starren Merkmalsvererbung. Informationen, die an einem Knoten angelagert werden, bleiben dort fixiert und können nicht an mehrere Stellen des Baumes kopiert werden. Es gibt jedoch empirische Befunde (Rips, Shoben & Smith, 1973), daß die Häufigkeit der Abrufung bestimmter Informationen die Wahrscheinlichkeit deren direkter Abspeicherung erhöht. Seltener benötigte Fakten werden dagegen eher *deduziert*. Wir haben daher eine Dimension „Speicherung vs. Berechnung" vorliegen.

Wir wollen in Ansätzen zeigen, daß die Regelmodelle flexibel genug sind, auf die geäußerten Einwände einzugehen. Die Berechnung von Unter- und Oberbegriffsinformation aus Merkmalseigenschaften haben wir schon im vorangegangenen Abschnitt vorgestellt. Auf die Relationsstärke und die Dichotomie „Speicherung vs. Berechnung" gehen wir unten mit einem Beispiel ein. Dagegen haben wir aus Platzgründen darauf verzichtet, ein Regelmodell mit dynamischer Informationsverwaltung darzustellen.

Wir formalisieren die Begriffspyramide von Abb. 18.8. als Regelmodell mit *expliziter* Abspeicherung von Merkmalsinformationen (s. Tab. 18.4.).

Tab. 18.4.: Formalisierte Abstraktionshierarchie mit Merkmalsvererbung und *expliziter* Informationsabspeicherung

```
/********************* Abstraktionshierarchie ******************/
/*********** mit  e x p l i z i t e r  Informationsabspeicherung ********/
/**************** nach COLLINS & QUILLIAN (1969) **************/
((kann  Kanarienvogel  singen))
((kann  Hai  beißen))
((kann  Vogel  fliegen))
((kann Fisch  schwimmen))
((kann Tier  sich-bewegen))
((kann  Kanarienvogel  _etwas) (kann  Vogel  _etwas))
((kann  Hai  _etwas) (kann  Fisch  _etwas))
((kann  Lachs  _etwas) (kann  Fisch  _etwas))
((kann  Vogel  _etwas) (kann  Tier  _etwas))
((kann  Fisch  _etwas) (kann  Tier _etwas))
```

```
((ist  Kanarienvogel  gelb))
((ist  Hai  gefährlich))
((ist  Lachs  rosa))
((ist  Lachs eßbar))
((schwimmt  Lachs  stromaufwärts))
((hat  Vogel  Flügel))
((hat  Vogel  Federn))
((hat  Fisch  Flossen))
((hat  Fisch  Kiemen))
((hat  Tier  Haut))
((hat  Kanarienvogel  _etwas) (hat  Vogel  _etwas))
((hat  Hai  _etwas) (hat  Fisch  _etwas))
((hat  Lachs  _etwas) (hat  Fisch  _etwas))
((hat  Vogel  _etwas) (hat  Tier  _etwas))
((hat  Fisch  _etwas) (hat  Tier  _etwas))
((frißt  Tier  Futter))
((frißt  Kanarienvogel  _etwas) (frißt  Vogel  _etwas))
((frißt  Hai  _etwas) (frißt  Fisch  _etwas))
((frißt  Lachs  _etwas) (frißt  Fisch  _etwas))
((frißt  Vogel  _etwas) (frißt  Tier  _etwas))
((frißt  Fisch  _etwas) (frißt  Tier  _etwas))
((atmet  Tier  Sauerstoff))
((atmet  Kanarienvogel  _etwas) (atmet  Vogel  _etwas))
((atmet  Hai  _etwas) (atmet  Fisch  _etwas))
((atmet  Lachs  _etwas) (atmet  Fisch  _etwas))
((atmet  Vogel  _etwas)  (atmet  Tier  _etwas))
((atmet  Fisch  _etwas)  (atmet  Tier  _etwas))
```

Die Möglichkeit, Unter- bzw. Oberbegriffsinformation zu *deduzieren*, führt zu einem Regelmodell mit *impliziter* Abspeicherung (s. Tab. 18.5.).

Tab. 18.5.: Formalisierte Abstraktionshierarchie mit Merkmalsvererbung und *impliziter* Informationsabspeicherung

```
/********************* Abstraktionshierarchie *****************/
/********** mit  i m p l i z i t e r  Informationsabspeicherung ********/
((kann  Kanarienvogel  singen))
((kann  Hai  beißen))
((kann  Vogel  fliegen))
((kann Fisch  schwimmen))
((kann Tier  sich-bewegen))
((kann  _wer  _etwas)
          (ist-Unterbegriff  _wer  _Oberbegriff) (kann  _Oberbegriff  _etwas))
((ist  Kanarienvogel  gelb))
((ist  Hai  gefährlich))
((ist  Lachs  rosa))
((ist  Lachs eßbar))
((ist-ein  Kanarienvogel  Vogel))
((ist-ein  Vogel  Tier))
((ist-ein  Hai  Fisch))
((ist-ein  Lachs  Fisch))
```

```
((ist-ein  Fisch  Tier))
((ist-Unterbegriff  _Unterbegriff  _Oberbegriff)
        (ist-ein  _Unterbegriff  _Oberbegriff))
((ist-Unterbegriff  _Unterbegriff  _indirekter-Oberbegriff)
        (ist-ein  _Unterbegriff  _direkter-Oberbegriff)
        (ist-Unterbegriff  _direkter-Oberbegriff  _indirekter-Oberbegriff))
((schwimmt  Lachs  stromaufwärts))
((hat  Vogel  Flügel))
((hat  Vogel  Federn))
((hat  Fisch  Flossen))
((hat  Fisch  Kiemen))
((hat  Tier  Haut))
((hat  _wer  _etwas)
        (ist-Unterbegriff  _wer  _Oberbegriff) (hat  _Oberbegriff  _etwas))
((frißt  Tier  Futter))
((frißt  _wer  _etwas)
        (ist-Unterbegriff  _wer  _Oberbegriff) (frißt  _Oberbegriff  _etwas))
((atmet  Tier  Sauerstoff))
((atmet  _wer  _etwas)
        (ist-Unterbegriff  _wer  _Oberbegriff)
        (atmet  _Oberbegriff  _etwas))
```

Die sechs Regeln ermöglichen Suchprozesse über das ganze Netz hinweg, so daß das Netz mit minimaler Kantenzahl auskommt: Speicherökonomie zu Lasten der Berechnungsökonomie.

Die Relations- oder Assoziationsstärke können wir einführen, indem wir Fakten und Regeln linear ordnen. Die vertikale Ordnung der Klauseln in den Tab. 18.4. und 18.5. soll dieser Ordnung entsprechen. Entsprechend muß der abstrakte Interpreter eine geänderte Verarbeitungsstrategie befolgen. Die Regelauswahl wird nicht länger nichtdeterministisch, sondern entsprechend der hier eingeführten Ordnung vorgenommen. Zuerst werden die Fakten entsprechend ihrer vertikalen Ordnung und damit nach ihrer Assoziationsstärke durchsucht. Erst danach werden Inferenzprozesse in Gang gesetzt. Denkbar sind auch Interpreter, die die Anordnung der Klauseln im Sinne von Verstärkung, Hemmung oder dem Refraktärprinzip *dynamisch* verändern.

Wir wollen jetzt mit der Frage „Wer kann was?" einen Suchprozeß im Regelmodell mit impliziter Abspeicherung (Tab. 18.5) in Gang setzen. Die Frage wird in das Ziel (kann _wer _was)? umformuliert. Die entsprechenden Instantiierungen der Variablen _wer und _was lauten der Reihenfolge nach:

_wer = Kanarienvogel	Hai	Vogel	Fisch
_was = singen	beißen	fliegen	schwimmen
_wer = Tier	Kanarienvogel	Kanarienvogel	Vogel
_was = sich-bewegen	fliegen	sich-bewegen	sich-bewegen
_wer = Hai	Hai	Lachs	Lachs
_was = schwimmen	sich-bewegen	schwimmen	sich-bewegen
_wer = Fisch	Kanarienvogel	Hai	Lachs
_was = sich-bewegen	sich-bewegen	sich-bewegen	sich-bewegen

Die Instantiierungen können als aktive Musterabgleichung *(pattern matching)* verstanden werden. Sie tragen Informationen in die Regeln hinein und wieder hinaus. Entsprechende kognitionspsychologische Untersuchungen finden sich bei Anderson (1976, 1983a).

Zum Schluß wollen wir noch eine einfache *chronometrische Studie* zum Komplex der Unter- bzw. Oberbegriffserkennung vornehmen, die sich auf die Ausführungen von Klix zu diesem Thema (Klix, in diesem Band, Abb. 5) stützt. Das von uns konzipierte Regelsystem ist in Tab. 18.6. wiedergegeben.

Tab. 18.6.: Formalisierung der Prüfung auf Neben-, Über- oder Unterordnung eines Begriffs As

```
((IST-A-UNTERBEGRIFF-VON-As?  _A  _As  _Antwort  _Antwortzeit)
             (TICKS  _Anfangszeit)
/*0*/    (EINE-GEMEINSAMKEIT?  _A  _As)
             (IF
/*1*/         (EINE-SPEZIFISCHE-EIGENSCHAFT?   _A   _As)
                  ((IF
/*2*/               (ENTHALTEN?  _As  _A  )
                        ((EQ  _Antwort  "A ist As untergeordnet"))
                        ((EQ  _Antwort  "A ist As nebengeordnet"))))
                  ((EQ  _Antwort  "A ist As übergeordnet oder identisch")))
            (TICKS  _Endzeit)
            (-  _Endzeit  _Anfangszeit  _Antwortzeit) /)
((IST-A-OBERBEGRIFF-VON-As?  _A  _As  _Antwort  _Antwortzeit)
             (TICKS  _Anfangszeit)
/*0*/    (EINE-GEMEINSAMKEIT?  _A  _As)
             (IF
/*1*/          (KEINE-SPEZIFISCHE-EIGENSCHAFT?   _A   _As)
                  ((IF
/*2*/               (ENTHALTEN?    _A  _As)
                        ((EQ  _Antwort "A ist identisch mit As"))
                        ((EQ  _Antwort "A ist As übergeordnet"))))
                  ((EQ  _Antwort  "A ist As neben- oder untergeordnet")))
            (TICKS  _Endzeit)
            (-  _Endzeit  _Anfangszeit  _Antwortzeit) /)
((EINE-GEMEINSAMKEIT?  (_firstA | _restA )  _As )
             (ON  _firstA  _As)  /)
((EINE-GEMEINSAMKEIT?  (_firstA | _restA )  _As )
             (EINE-GEMEINSAMKEIT?  _restA  _As)  /)
((EINE-SPEZIFISCHE-EIGENSCHAFT?    (_firstA | _restA)     ()) /)
((EINE-SPEZIFISCHE-EIGENSCHAFT?    (_firstA | _restA)  _As)
             (NOT  ON  _firstA  _As)  /)
((EINE-SPEZIFISCHE-EIGENSCHAFT?    (_firstA | _restA)  _As)
             (EINE-SPEZIFISCHE-EIGENSCHAFT?   _restA   _As)  /)
((KEINE-SPEZIFISCHE-EIGENSCHAFT?  _A  _As)
            (NOT  EINE-SPEZIFISCHE-EIGENSCHAFT?  _A  _As))
((ENTHALTEN?     ()      _A))
((ENTHALTEN?    (_firstAs | _restAs)  _A)
             (ON  _firstAs  _A)
            (ENTHALTEN?  _restAs  _A))
```

Auch dieses Regelwerk ist wie alle anderen rückwärtsverkettenden Systeme in MacPROLOG geschrieben. Es taucht hier ein neues Sprachkonstrukt auf, das einer kurzen Erläuterung bedarf. Die Relation IF hat folgende Syntax:

⟨IF-Relation⟩ ::= (IF ⟨Bedingungsrelation⟩ ⟨Ja-Zweig⟩ ⟨Nein-Zweig⟩)
⟨Bedingungsrelation⟩ ::= ⟨Ziel⟩
⟨Ja-Zweig⟩ ::= ({⟨Ziel⟩}) {}bedeutet Null oder mehrmalige Wiederholung
⟨Nein-Zweig⟩ ::= ({⟨Ziel⟩})

Ist die Bedingungsrelation erfüllt, wird die Liste der Ziele des Ja-Zweigs anderenfalls die des Nein-Zweigs abgearbeitet.

Erläutern wir zunächst die *Unter*-Oberbegriffserkennung. Dazu nehmen wir an, daß der Referenzbegriff A die Merkmalsliste (w x y z) und der Vergleichsbegriff As die Liste (w x) aufweisen. Wir stellen dem Regelsystem die Frage (IST-A-UNTERBEGRIFF-VON-As? (w x y z) (w x) _Antwort _Antwortzeit). Das in der Frage enthaltene Ziel aktiviert die erste Regel in Tab. 18.6. Danach wird mit dem Ziel (TICKS _Anfangszeit) die Anfangszeit an die Variable _Anfangszeit gebunden. Als nächstes wird das Ziel (EINE-GEMEINSAMKEIT? _A _As) aktiviert. Die Markierung /*0*/ vor dem Ziel verweist auf die Numerierung des Prüfschritts im Text von Klix („O-ter Schritt"). Die Prüfung ergibt, daß eine Liste mit gemeinsamen Merkmalen vorliegt. Sie lautet (w x). Deshalb ist das Ziel erfüllt, und wir können uns dem nächsten Ziel (EINE-SPEZIFISCHE _EIGENSCHAFT? _A _As) zuwenden.

Es wird jetzt geprüft, ob A gegenüber As spezifische Merkmale besitzt. Das Ziel kann mit der für A spezifischen Eigenschaftsliste (y z) als erfüllt gelten.

Die Markierung /*1*/ weist ebenfalls wieder auf die Klixsche Prüfschrittnumerierung hin.

Hätte A gegenüber As keine spezifischen Eigenschaften aufgewiesen, wäre die Prüfung zu Ende und die Antwort wäre: „A ist As übergeordnet oder identisch."

In unserem Fall liegen jedoch spezifische Merkmale vor. Deshalb kann man jetzt zum Ziel (ENTHALTEN? _As _A) übergehen. Wir prüfen jetzt, ob der Merkmalssatz von As eine Untermenge von Merkmalssatz A ist. Wenn das den Tatsachen entspricht, lautet die Antwort: „A ist As untergeordnet". Anderenfalls stellt das Regelsystem fest: „A ist As nebengeordnet."

Anschließend werden mit den Zielen (TICKS _Endzeit) und (– _Endzeit _Anfangszeit _Antwortzeit) die Endzeit und die verbrauchte Rechenzeit bestimmt.

Die für die *Ober*-Unterbegriffserkennung entsprechende Prüfschrittfolge ist in der zweiten Regel mit dem Kopf (IST-A-OBERBEGRIFF-VON-As? _A _As _Antwort _Antwortzeit) niedergelegt. Auf die Prüfung gemeinsamer Merkmale folgt die Prüfung, ob A gegenüber As *keine* spezifische Eigenschaft besitzt. Ist das der Fall, ist A dem Begriff As übergeordnet oder mit ihm identisch. Diese beiden Fälle können noch unterschieden werden mit einer Prüfung (ENTHALTEN? _As _A). Bei Identität von As und A ist Ziel erfüllt, und die Antwort lautet: „A ist identisch mit As." Liegt dagegen eine spezifische Eigenschaft von A gegenüber

As vor, bleibt nur die Feststellung: „A ist As neben- oder untergeordnet." Wir haben nun für jeweils drei permutierte Merkmalslistenpaare die Entscheidungszeiten zusammengetragen (Tab. 18.7.).

Tab. 18.7.: Zeittakte zur Durchführung der Unter-Ober- sowie der Ober-Unterbegriffsbestimmung in Abhängigkeit von der Anordnung der Begriffe in einer Abstraktionshierarchie

	wahre Situation:			
	a)	b)	c)	d)
	As	A	A As	A = As
Prüfung:	A	As		
Ist-A-Unterbegriff-Von-As?	11	12	12	12
Ist-A-Oberbegriff-Von-As?	8	22	8	20

a) Unterordnung von A
b) Unterordnung von As
c) Nebenordnung
d) Identität
Die Zahlen geben gemittelte und dann gerundete Zeittakte an, die zur Prüfung der jeweiligen Situation notwendig sind. Die Merkmalslisten entsprechen den in der Abb. 18.6. benutzten. Jede Prüfung wurde dreimal mit permutierten Listen repliziert.

Auffällig dabei und für die empirische Hypothesenbildung nützlich ist unsere Beobachtung, daß die Prüfung IST-A-OBERBEGRIFF-VON-As? in zwei Fällen erheblich länger dauert als die inverse Prüfung IST-A-UNTERBEGRIFF-VON-As? Der erste Fall liegt vor, wenn tatsächlich As übergeordnet ist. Aber auch bei identischen A und As dauert die IST-A-OBERBEGRIFF-VON-As?-Prüfung länger als die Berechnung des inversen Prädikats.

Die Prüfung IST-A-OBERBEGRIFF-VON-As dauert bei beiden Fällen im wesentlichen länger, weil wir zusätzlich die Prüfung (ENTHALTEN? _As _A) einschalten, um die Fälle „A ist identisch mit As" und „A ist As übergeordnet" auseinanderhalten können. Bei den anderen Tabellendifferenzen ergeben sich die Zeitunterschiede auch im wesentlichen durch die Einfügung oder den Fortfall der (ENTHALTEN? _As _A) Prüfung. Die Zeitunterschiede, die durch Permutation der Merkmalslisten entstanden, waren wesentlich geringer. Sie wurden daher nicht berücksichtigt.

Schlußbemerkungen

Wir haben ansatzweise gezeigt, wie Modellierungen mit Regelmodellen aussehen können. Dazu haben wir elementare deterministische Regelsysteme, die vorwärts- und rückwärtsverkettend ausgelegt waren, diskutiert. Es wurden Hinweise auf empirische Indikatoren gegeben, die aus den Modellen ableitbar sind. Als Implementationssprachen wurden LISP und PROLOG benutzt. Es zeigte sich, daß der Nutzen von Computermodellen vielfältig sein kann. Einmal wird die Theorien- und Hypothesenbildung verfeinert, da alle Prozeßkomponenten expliziert werden können. Zum anderen sind Prüfungen der Vollständigkeit und Widerspruchsfreiheit möglich, weil die Modelle *ablauffähig* sind. Damit kann das Modell in seinem dynamischen Verhalten genau studiert werden. Zum dritten sind aus den Modellen hinreichend viele empirische Indikatoren ableitbar, so daß die empirische Tragfähigkeit überprüfbar ist und der größere Detaillierungsgrad der Regelmodelle gegenüber den stochastischen Parametermodellen klassischer Art nicht zu einem Absicherungsmanko führt.

Leider haben wir in diesem Beitrag die stochastische Seite der Modelle nur rudimentär behandelt. Gänzlich unberücksichtigt ließen wir zudem die Verarbeitung widersprüchlicher und unscharfer Informationen. Uns sind zwar diese Lükken bewußt, andererseits standen diese Themen auch bei den von uns zitierten Theorien nicht im Mittelpunkt der Erörterungen.

Literatur

Aebli, H. (1980/81). *Denken: Das Ordnen des Tuns* (Bd. I/II). Stuttgart: Klett-Cotta.

Anderson, J. A., & Hinton, G. E. (1981). Models of information processing in the brain. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory* (pp. 9–48). Hillsdale, N. J.: Erlbaum.

Anderson, J. R. (1976). *Thought and memory.* Hillsdale, N. J.: Erlbaum.

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

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

Anderson, J. R. (1983b). Acquisition of proof skills in geometry. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 191–219). Palo Alto: Tioga.

Anderson, J. R., Kline, P. J., & Beasley, Ch.. M. (1980). Complex learning processes. In R. E. Snow, P. A. Federico & W. E. Montague (Eds.), *Aptitude, learning and instruction.* Hillsdale, N. J.: Erlbaum.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124–140.

Bandler, W., & Kohout, L. J. (1985). Probalistic versus fuzzy production rules in expert systems. *International Journal of Man-Machine Studies, 22*, 347–353.

Bauer, F. L., & Goos, G. (1984, 3. Auflage). *Informatik, Bd. I, Bd. II.* Berlin: Springer.

Bauer, F. L., & Wössner, H. (1984). Algorithmische Sprache und Programmentwicklung. Berlin: Springer.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, N. J.: Erlbaum.

Clark, K. L., & McCabe, F. G. (1984). *Micro-PROLOG: Programming in logic.* Englewood Cliffs, N. J.: Prentice-Hall.

Cohen, J. (1985). Describing PROLOG by its interpretation and compilation. *Communications of the ACM, 28*, 1311–1324.

Collins, A. M., & Quillian, M. R. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior, 8*, 240–247.

Colmerauer, A. (1985). PROLOG in 10 figures. *Communications of the ACM, 28*, 1296–1310.

Crowe, D. W. (1956). The n-dimensional cube and tower of Hanoi. *American Mathematical Monthly, 63*, 29–30.

Doignon, J. P., & Falmagne, J. C. (1985). Spaces for the assessment of knowledge. *International Journal of Man-Machine Studies, 23*, 175–196.

Dörner, D. (1976). *Problemlösen als Informationsverarbeitung.* Stuttgart: Kohlhammer.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, Mass.: MIT Press.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*, 17–37.

Gardner, M. (1982, 5. Auflage). *Mathematische Rätsel und Probleme.* Braunschweig: Vieweg.

Gediga, G., & Schöttke, H. (1985). *Zur Schätzung der automatischen, lokalen und globalen Anteile der Informationsverarbeitung beim Problemlösen*, Arbeitsbericht 12. Universität Osnabrück: FB Psychologie.

Greeno, J. G. (1974). Representation of learning as discrete transition in a finite state space. In D. H. Krantz, R. C. Atkinson, R. D. Luce & P. Suppes (Eds.), *Contemporary developments in mathematical psychology, Vol. I: Learning, memory and thinking* (pp. 1–43). San Francisco: Freeman.

Hasemer, T. (1984). An introduction to LISP. In T. O'Shea & M. Eisenstadt (Eds.), *Artificial intelligence and applications* (pp. 22–62). New York: Harper & Row.

Hinton, G. E., & Anderson, J. A. (1981). *Parallel models of associative memory.* Hillsdale, N. J.: Erlbaum.

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

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22*, 365–394.

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

Klahr, D. (1981). Informationsverarbeitungsmodelle der Denkentwicklung. In R. H. Kluwe & H. Spada (Hg.), Studien zur Denkentwicklung (S. 231–289). Bern: Huber.

Klix, F. (1971). *Information und Verhalten.* Berlin: VEB Deutscher Verlag der Wissenschaften.

Krantz, D. H., Atkinson, R. C., Luce, R. D., & Suppes, P. (Eds.) (1974). *Contemporary developments in mathematical psychology, Vol. I: Learning, memory and thinking, Vol. II: Measurement, psychophysics and neural information processing.* San Francisco: Freeman.

Lüer, G. (1981). Bemerkungen zu mathematischen und psychometrischen Modellen der kognitiven Entwicklung aus der Sicht von Theorien der Informationsverarbeitung. In R. H. Kluwe & H. Spada (Hg.), *Studien zur Denkentwicklung* (S. 79–91). Bern: Huber.

McClelland, J. R., Rumelhart, D. E., & PDP Research Group (1986). *Parallel distributed processing II.* Cambridge, Mass.: MIT Press.

Millward, R. B., & Wickens, T. D. (1974). Concept-identification models. In D. H. Krantz, R. C. Atkinson, R. D. Luce & P. Suppes (Eds.), *Contemporary developments in mathema-*

tical psychology, Vol. I: Learning, memory & thinking (pp. 45–100). San Francisco: Freeman.

Minsky, M. (Ed.) (1968). *Semantic information processing.* Cambridge, Mass.: Massachusetts Institute of Technology Press.

Möbus, C., & Nagl, W. (1983). Messung, Analyse und Prognose von Veränderungen. In J. Bredenkamp & H. Feger (Hg.), *Bd. 5 der Serie: Forschungsmethoden der Psychologie in der Enzyklopädie der Psychologie* (S. 239–470). Göttingen: Hogrefe.

Möbus, C., Göricke, G., & Kröh, P. (1983). Statistical analysis of single-case experimental designs: Conditional equivalence of the general-linear model approach of Glass, Willson & Gottman with the intervention model of BOX & Tiao. *EDV in Medizin und Biologie/ EDP in Medicine and Biology,* 14, 98–108.

Möbus, C., Göricke, G., & Kröh, P. (1985). Nichtparametrische und parametrische Methoden der Einzelfallstatistik. In H. Appelt & B. Strauss (Hg.), *Ergebnisse einzelfallstatistischer Untersuchungen* (S. 170–188). Heidelberg: Springer.

Neves, O., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 57–84). Hillsdale, N. J.: Erlbaum.

Newell, A. (1980). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. S. Nickerson (Ed.), *Attention and performance VIII.* Hillsdale, N. J.: Erlbaum.

Newell, A. (1980). Physical symbol system. *Cognitive Science, 4,* 135–183.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, N. J.: Prentice Hall.

Nilsson, N. J. (1971). *Problem solving methods in artificial intelligence.* New York: McGraw Hill.

Nilsson, N. J. (1980). *Principles of artificial intelligence.* Palo Alto, CA: Tioga.

Norman, D. A., & Rumelhart, D. E. (Hg.) (1978). *Strukturen des Wissens.* Stuttgart: Klett-Cotta.

Piaget, J. (1976). *The grasp of consciousness.* Cambridge, Mass.: Harvard University Press.

Polson, P. G., & Kieras, D. E. (1984). A formal description of users' knowledge of how to operate a device and user complexity. *Behavior Research Methods, Instruments & Computers, 16,* 249–255.

Polson, P. G., & Kieras, D. E. (1985). A quantitative model of the learning and performance of text editing knowledge. *CHI '85 Proceedings,* 207–212.

Polson, P. G., Muncher, E., & Engelbeck, G. (1986). A test of a common elements theory of transfer. *CHI '86 Proceedings,* 78–83.

Pollack, J., & Waltz, D. L. (1986). Interpretation of natural language: A potential application of parallelism. *Byte,* 189–198.

Pylyshyn, Z. W. (1984). *Computation and cognition.* Cambridge, Mass.: MIT Press.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing* (pp. 227–270). Cambridge, Mass.: MIT Press.

Rips, L. J., Shoben, E. J., & Smith, E. E. (1973). Semantic distance and the verification of semantic relations. *Journal of Verbal Leraning und Verbal Behavior, 12,* 1–20.

Rosenbloom, P. S., & Newell, A. (1986). The chunking of goal hierarchies. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning* (Vol. II) (pp. 247–288). Los Altos, CA: Kaufmann.

Rumelhart, D. E., McClelland, J. L., & PDP Research Group (1986). *Parallel distributed processing I.* Cambridge, Mass.: MIT Press.

Schmalhofer, F., & Polson, P. G. (1986). A production system model for human problem solving. *Psychological Review, 48,* 113–122.

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

Simon, H. A., & Newell, A. (1974). Thinking processes. In D. H. Krantz, R. C. Atkinson,

R. D. Luce & P. Suppes (Eds.), *Contemporary developments in mathematical psychology, Vol. I: Learning memory and thinking* (pp. 101–144). San Francisco: Freeman.
Smith, E. E. (1978). Theories of semantic memory. In W. K. Estes (Ed.), *Handbook of learning and cognitive processes* (Vol. 6) (pp. 1–56). Hillsdale, N. J.: Erlbaum.
Spada, H. (1973). Die Analyse kognitiver Lerneffekte mit stichprobenunabhängigen Verfahren. In K. Frey & M. Lang (Hg.), *Kognitionspsychologie und naturwissenschaftlicher Unterricht* (S. 94–129). Bern: Huber.
Stoyan, H., & Görz, G. (1984). *LISP*. Berlin: Springer.
Sydow, H. (1970). Zur metrischen Erfassung von subjektiven Problemzuständen und zu deren Veränderung im Denkprozeß (II). *Zeitschrift für Psychologie, 178*, 1–50.
Wender, K. F., Colonius, H., & Schulze, H. H. (1980). *Modelle des menschlichen Gedächtnisses*. Stuttgart: Kohlhammer.

Anhang 1:

Der Turm von Hanoi als vorwärtsverkettendes Produktionssystem und ein Interpreter

Das von uns verwendete Produktionssystem stützt sich auf vier Produktionen, die gegenüber S. 432 weiter präzisiert werden. Es folgen noch einige Erläuterungen: das Ziel, von Stab A die Scheibe 1 nach Stab C zu schaffen, wird symbolisiert durch die Liste (1 A C). Diese bildet ein Element des Arbeitsspeichers. Für diesen intendierten Zug ist A der Ausgangsstab und C der Zielstab. Ein Turm ist eine geordnete Menge von Scheiben. So existieren in Abb. 18.3 zwei Türme. Der erste befindet sich bei Stab A und der zweite bei Stab C.

Die Produktionen lauten nun:

P1: WENN alle Scheiben sich bei Stab C befinden,
DANN ist das Problem gelöst und halte.

P2: WENN es kein gegenwärtiges Ziel gibt,
DANN nimm Dir als neues Ziel vor, die größte Scheibe, die sich nicht bei Stab C befindet, nach Stab C zu schaffen.

P3: WENN es ein Ziel gibt, eine Scheibe zu bewegen,
UND
WENN die Scheibe oben auf dem Turm liegt
UND
WENN entweder die Scheibe kleiner ist als die oberste Scheibe des Zielturms
ODER es gar keinen Zielturm gibt,
DANN setze das Ziel in eine Aktion um
UND lösche das Ziel.

P4: WENN es ein Ziel gibt, eine Scheibe zu bewegen,
UND
WENN diese Scheibe nicht bewegt werden kann,
DANN lösche das Ziel

UND nimm Dir als neues Ziel vor, die größte störende Scheibe zu dem Stab zu schaffen, der im vorherigen Ziel weder Ausgangs- noch Zielstab war.

Kommen wir nun zur Beschreibung unseres Produktionssystems. Lesern, die gewohnt sind, LISP-Code zu lesen, sei die vorherige Lektüre von Hasemer (1984) empfohlen. Wir haben einige für den Anfänger fortgeschrittene Konzepte, wie „nichtlokaler Ausgang" und Funktionen mit CATCH-THROW, Zufallsauswahl der Produktionen zur Demonstration ihrer Orthogonalität, „Produktionsgedächtnis als unendliche Datenstruktur" und speicherstrukturverändernde Funktionen (wie z. B. RPLACA und RPLACD) verwendet. Der Text des Interpreters findet sich weiter unten. Er wurde nicht auf Effizienz, sondern auf gute Lesbarkeit hin geschrieben. Die zugehörigen TLC-LISP-Funktionen, die nicht im Lehrbuch von Stoyan und Görz dokumentiert sind, finden sich im Anhang 2.

Der Interpreter weist gegenüber den von Opwis (in diesem Band) zitierten Systemen PRISM, OPS5 und GRAPES natürlich eine Reihe von Vereinfachungen auf. So haben wir keine Konfliktlösungsstrategie einprogrammiert. Die Produktionen erfordern dieses auch gar nicht. Der Mustervergleich wird auch sehr einfach gehandhabt (mit der LISP-Funktion EQUAL). Kompliziertestes *pattern matching* benötigen wir nicht, weil wir hier keinen allgemein anwendbaren Interpreter schreiben wollen. Ferner fehlt eine Kompilierung der Regeln in ein Netzwerk, wie es von Forgy (1982) vorgeschlagen wurde. Dieses ist zwar ein interessanter Aspekt, der aber an dieser Stelle für Modellierungen nicht weiter hilfreich ist.

Das Produktionssystem setzt sich aus einem Initialisierungsteil (Funktion „hanoi"), einem Interpreter (Funktion „productions") und untergeordneten Funktionen (wie „peg!" ... „tower-above?") zusammen. Der Start erfolgt z. B. bei einem 4-Scheibenproblem mit dem Aufruf (hanoi '(1 2 3 4)), wenn die Regeln der Reihe nach abgearbeitet werden sollen. Bei Zufallsanwendung der Regeln wird dagegen das zusätzliche Argument „T" in den Aufruf gesetzt: „(hanoi '(1 2 3 4) T)". Wir gehen davon aus, daß der 4er-Turm von Stab A nach Stab C geschafft werden soll. Das Argument T wird an die globale Variante ***randomp*** angebunden. Fehlt das Argument, ist ***randomp*** gleich NIL. Dann wird der Arbeitsspeicher und der Produktionssystemspeicher initialisiert. Der Arbeitsspeicher besteht aus 4 symbolischen Einheiten. Die erste enthält das Ziel, die drei weiteren die Enkodierung der Scheibenanordnungen an den Stäben A, B, C. Die Speicher werden erst als leer („NIL") angenommen. Dann erfolgt mit (Peg! 'A tower) die Positionierung des Turms (1 2 3 4) auf Stab A. Danach werden die vier Produktionen mit (rule! ⟨Text der Produktion⟩) in das Produktionsgedächtnis aufgenommen. Die Regeln korrespondieren zu den o.a. umgangssprachlich formulierten Regeln. Wir kommen noch später darauf zurück.

Nach der „Instruktionsphase" wird mit dem Funktionsaufruf „(ring-of-productions)" die Liste der Produktionen an Anfang und Ende zu einem Ring verknüpft. Dadurch wird die Bestimmung der nächsten anzuwendenden Regel mittels der Funktion „nextrule!" sehr vereinfacht.

Anschließend wird mit dem Funktionsaufruf „(productions)" der Produktions-

interpreter aufgefordert, mit der Regelanwendung zu beginnen. Sollte die Endbedingung in Regel 1 erreicht sein (alle Scheiben befinden sich bei Stab C), wird die Meldung „the problem is solved!" geäußert und die Regelanwendung nichtlokal verlassen. Bevor zur Marke im Aufruf von „(CATCH ⟨Marke⟩ ...)" gesprungen wird, werden noch die Aktionen ausgeführt, die hinter „(productions)" im Aufruf „(UNWIND-PROTECT ., . .)" stehen. Dann hält der Interpreter an. Die Regeln sind leicht zu lesen. Sie werden in das Produktionsgedächtnis mit der Funktion „(rule! ⟨text⟩)" eingesetzt. Der ⟨text⟩ setzt sich zusammen aus ⟨text⟩::= (⟨Regelname⟩ ⟨IF-THEN-Ausdruck⟩). Der ⟨IF-THEN-Ausdruck⟩ läßt sich weiter ausdifferenzieren in ⟨IF-THEN-Ausdruck⟩::= (IF ⟨Bedingungsteil⟩ THEN ⟨Aktionsteil⟩). Die *erste Regel* stellt die Umsetzung der ersten natürlichsprachlichen Regel dar. Wir stützen uns dabei auf das Prädikat (reached-goal?). Das Prädikat prüft, ob Stab A und B leer sind. Wenn diese Bedingung erfüllt ist, liefert der Aufruf (reached-goal?) den Wert „wahr" („T" in LISP). Dann springen wir mit der Meldung „the problem is solved!" nichtlokal zur Marke „problem-solved" im Aufruf „(CATCH problem-solved ...)", leeren die Speicher und halten an.

In *Regel 2* wird gefragt, ob die Funktion goal? uns ein leeres Ziel im Arbeitsspeicher liefert. Sollte das der Fall sein, wird mit „(goal! ⟨zu-bewegende-Scheibe⟩ ⟨ausgangsstab⟩ ⟨zielstab⟩)" ein neues Ziel gesetzt. Die zu bewegende Scheibe ist die größte Scheibe, die sich noch nicht beim Zielstab C befindet. Sie wird mit dem Funktionsaufruf „(max-disc? 'a 'b)" bestimmt. Dabei wird die Funktion „peg?" benutzt, die mit „(peg? ⟨stab⟩)" aufgerufen wird und uns die Scheibenanordnung dieses Stabes zurückgibt; „(peg? ⟨stab⟩)" enthält also ebenso wie zum Beispiel „(max disc? ...)" eine perzeptive Komponente der Strategie.

Der Ausgangsstab wird dann mit „(which-peg-is? ⟨scheibe⟩)" bestimmt. Der Funktionswert gibt an, wo sich die größte Scheibe befindet, die nicht beim Ziel liegt. Zielstab für den Zug ist unverändert der Stab C.

Die *dritte Regel* arbeitet einen UND/ODER-Baum (Nilson, 1971, S. 87 ff.) von Bedingungen oder Zielen ab. Dabei kann der Aktionsteil der Produktion als Oberziel und der Bedingungsteil der Produktion als Subzielhierarchie angesehen werden, die allerdings in unserem vorwärtsarbeitenden System von unten her abgearbeitet wird, ohne daß das Oberziel vorher bekannt wäre.

Dabei stellt der ODER-Teilbaum (rechts unten) die Ausdifferenzierung der umgangssprachlichen Formulierung „wenn diese Scheibe nicht bewegt werden kann" dar. Der Aktionsteil der vierten Regel besteht aus einer Redefinition des Ziels. Es soll die „größte störende" Scheibe aus dem Weg geräumt werden. Sie soll zu einem Stab geschafft werden, der weder Ausgangs- noch Zielstab im gegenwärtigen Ziel ist. Der neue Zielstab wird mit der Funktion „newdestination" bestimmt. Die Angabe der größten störenden Scheibe mit der Funktion „most-annoying-disc?" setzt einen relativ komplizierten Suchprozeß voraus. Es müssen sowohl im Ziel wie im Ausgangsturm alle Scheiben bestimmt und das neue Ziel mit „(goal! ...)" gesetzt werden.

Der Regelinterpreter ist in der Funktion „productions" enthalten und ist selbstdokumentierend. Die aktuelle Regel wird mit dem Aufruf „(rule?)" ermittelt. Mit

den Funktionen „condition-of?" und „action-of?" erhalten wir den Bedingungs- und Aktionsteil einer Produktion. Die Funktion „try!!" wendet den Bedingungsteil auf die Inhalte des Arbeitsgedächtnisses an. Wir nehmen mit der Funktion „EVAL" die Hilfe des LISPinterpreters in Anspruch. Ist die Bedingung erfüllt, wird die Aktion mit der Funktion „do!!" ausgeführt. Dabei wird der Inhalt des Arbeitsgedächtnisses ausgegeben. Falls die globale Variable ***randomp*** auf T gesetzt wurde, wird die nächste Regel per Zufall bestimmt. Dazu wird die Funktion „nextrule!" aufgerufen.

Restliche Einzelheiten sind dem Programmtext zu entnehmen. Ein Ablauf des Produktionssystems mit Zufallsauswahl der Regeln findet sich im Anhang 3.

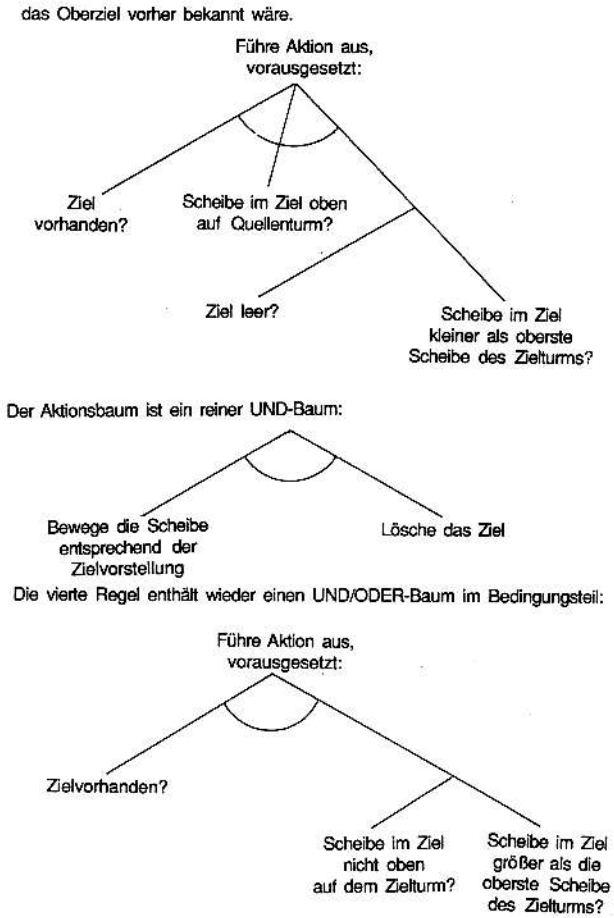


Abb.: Repräsentation der Produktionen P3 und P4 durch UND/ODER-Bäume

```
;-------------------------------------------------------------------------------
;                    the rule-interpreter is written in TLC-Lisp :
;                           The LISP Company
;                           430 Monterey Ave. #4
;                           Los Gatos, California 95030
;                           U S A
;
;-------------------------------------------------------------------------------
;
(DE hanoi (tower &opt (***randomp***    NIL) )          ; if true then random
                                                        ; selection of productions
    (LET ((***working-memory***      '( NIL             ; empty goal
                                        NIL             ; empty peg a
                                        NIL             ; empty peg b
                                        NIL             ; empty peg c
                                      )
          )
          (***production-memory***     NIL)             ; empty production memory
         )
         (peg!  'a  tower)
         ;------------------ rule 1 ------------------------------------------
         (rule!  '("rule1"                              ; insert rule 1
                   (IF   (reached-goal?)
                    THEN (THROW problem-solved "the problem is solved !" )
                   )
                  )
         )
         ;------------------ rule 2 ------------------------------------------
         (rule!  '("rule2"                              ; insert rule 2
                   (IF   (empty? (goal?))
                    THEN (goal! (max-disc? 'a 'b)
                                (which-peg-is? (max-disc? 'a 'b))
                                'c
                         )
                   )
                  )
         )
         ;------------------ rule 3 ------------------------------------------
         (rule!  '("rule3"                              ; insert rule 3
                   (IF   (AND (NOT (empty? (goal?)))
                              (EQUAL (disc-in? (goal?))
                                     (top-of-peg? (source-of? (goal?)))
                              )
                              (OR (empty? (peg? (destination-of? (goal?))))
                                  (LT  (disc-in? (goal?))
                                       (top-of-peg? (destination-of? (goal?)))
                                  )
                              )
                         )
                    THEN (AND (move! (disc-in? (goal?))
                                     (source-of?(goal?))
                                     (destination-of? (goal?))
                              )
                              (goal! NIL NIL NIL)
                         )
                   )
                  )
         )

(DE max-disc? (peg1 peg2)
    (maximum (APPEND (peg? peg1) (peg? peg2)))
)
(DE maximum (list)
    (DO ((index       1     (ADD1 index))                    ; initialization
         (max-element 1)
        )
        (((GT index (LENGTH list))  max-element))           ; exit test
        (SETQ max-element (MAX max-element (NTH list index))) ; body
    )
)
(DE which-peg-is? (disc)
    (COND ((MEMBER disc (peg? 'a))    'a)
          ((MEMBER disc (peg? 'b))    'b)
          ((MEMBER disc (peg? 'c))    'c)
          (t                          "disc not found")
    )
```

```
)
(DE rule-name? ()
    (FIRST (FIRST ***production-memory***))
)
(DE source-of?(goal)
    (IF goal (SECOND goal) NIL)
)
(DE destination-of? (goal)
    (IF goal (THIRD goal)  NIL)
)
(DE disc-in? (goal)
    (IF goal (FIRST goal) NIL)
)
(DE top-of-peg? (peg)
    (IF (peg? peg) (FIRST (peg? peg)) 10000) ; if empty then size of bottom
)                                            ; is larger than any disc
(DE move! (disc source destination)
    (peg! source (COPY (REST (peg? source))))
    (peg! destination (COPY (CONS disc (peg? destination))))
)
(DE new-destination? (source destination)
    (FIRST (REMOVE source (REMOVE destination (LIST 'a 'b 'c))))
)
(DE most-annoying-disc? (disc source-peg destination-peg)
    (maximum (APPEND (tower-above? disc source-peg)
                     (tower-above? disc destination-peg)
             )
    )
)
(DE tower-above? (reference-disc  tower)
    (DO ( (index  1  (ADD1 index))
          (above NIL)
        )
        ( ( (empty? tower)                          above)
          ( (GT index (LENGTH tower))               above)
          ( (LE reference-disc (NTH tower index))   above)
        )
        (SETQ above (CONS (NTH tower index) above))
    )
)
;-------------------------------------------------------------------------------

(DE peg? (peg)                    ; what tower is on peg?
    (COND((EQUAL peg  'a)  (SECOND ***working-memory***))
          ((EQUAL peg  'b)  (THIRD  ***working-memory***))
          ((EQUAL peg  'c)  (FOURTH ***working-memory***))
    )
)
(DE rule! (new-rule)                  ; insert new rule in ***production-memory***
    (SETQ ***production-memory*** (APPEND ***production-memory***
                                          (LIST new-rule)
                                  )
    )
)
(DE ring-of-productions ()                  ; change list of productions to ring
    (SETQ   ***number-of-rules*** (LENGTH ***production-memory***))
    (RPLACD (nonempty-tail-of-production-memory) ***production-memory***)
)
(DE nonempty-tail-of-production-memory ()
    (DO ( (tail  ***production-memory***  (REST tail))
        )
        ( ((empty? (REST tail))       tail )
        )
    )
)
(DE reached-goal? ()
    (AND (empty? (peg? 'a))(empty? (peg? 'b)))
)
(DE try!! (condition)
    (prin2 (rule-name?)) (prin2 " is tried !") (TERPRI)
    (EVAL condition)
)
(DE do!! (action)
    (prin2 (rule-name?)) (prin2 " is fired !") (TERPRI)
    (EVAL action)
    (prin2 "***working-memory*** is: ")(print ***working-memory***)
```

```
)
(DE condition-of? (rule)
    (SECOND rule)
)
(DE action-of? (rule)
    (FOURTH rule)
)
(DE goal! (disc  source  destination)          ; insert new goal in working memory
    (LET ((new-goal (LIST disc source destination)))
         (COND ((OR (EQUAL (goal?) NIL) (EQUAL new-goal NIL))
                (RPLACA  ***working-memory***  new-goal)
               )
               (T                    (RPLACB (goal?) new-goal))
         )
    )
)
(DE goal? ()                                  ; what is current goal ?
    (FIRST ***working-memory***)
)
(DE empty? (goal-or-peg)
    (OR (EMPTY  goal-or-peg)
        (AND (EQUAL (FIRST  goal-or-peg) NIL)
             (EQUAL (SECOND goal-or-peg) NIL)
             (EQUAL (THIRD  goal-or-peg) NIL)
        )
    )
)
(DE rule? ()                                  ; what is current rule ?
    (SECOND (FIRST ***production-memory***))
)
(DE next-rule! (&OPT (next 1))                ; prepare access to next rule !
    (FOR (index-of-rule  1  next)
         (SETQ ***production-memory*** (REST ***production-memory***))
    )
)
;------------------------ rule 4 ---------------------------------------
(rule!  '("rule4"                             ; insert rule 4
          (IF   (AND (NOT (empty? (goal?)))
                     (OR (NOT (EQUAL (disc-in? (goal?))
                                     (top-of-peg? (source-of? (goal?)))
                              )
                         )
                         (GT (disc-in? (goal?))
                             (top-of-peg? (destination-of? (goal?)))
                         )
                     )
                )
           THEN (goal! (most-annoying-disc?
                              (disc-in? (goal?))
                              (peg? (source-of? (goal?)))
                              (peg? (destination-of? (goal?)))
                       )
                       (which-peg-is? (most-annoying-disc?
                                        (disc-in? (goal?))
                                        (peg? (source-of? (goal?)))
                                        (peg? (destination-of?(goal?)))
                                      )
                       )
                       (new-destination? (source-of? (goal?))
                                         (destination-of? (goal?))
                       )
                )
          )
         )
)
;-------------------------------------------------------------------------
(ring-of-productions)
(CATCH problem-solved
       (UNWIND-PROTECT (productions)
                       (goal!   NIL NIL NIL)  ; before leaving hanoi
                       (peg! 'a NIL)
                       (peg! 'b NIL)
                       (peg! 'c NIL)
                       (SETQ ***production-memory*** NIL)
       )
)
;-------------------------------------------------------------------------
```

```
    )
)
;------------------------- interpreter for productions ------------------------
(DE productions ()                                    ; Interpreter for productions
    (IF (try!! (condition-of? (rule?)))     ; if condition meets
        (do!!  (action-of? (rule?)))                  ; action is fired
    )
    (IF ***randomp***                                 ; random selection of the
        (next-rule! (RN ***number-of-rules***))       ; next rule or
        (next-rule!)                                  ; deterministic selection
    )
    (productions)
)
;------------------------- primitive base-functions ---------------------------
(DE peg! (peg new-tower)                                        ; change tower on peg !
    (COND ((OR (EQUAL (peg? peg) NIL) (EQUAL new-tower NIL))
           (SELECTQ peg (a (RPLACA (REST ***working-memory***) new-tower))
                        (b (RPLACA (REST (REST ***working-memory***))new-tower))
                        (c (RPLACA (REST (REST (REST ***working-memory***)))
                                   new-tower
                           )
                        )
           )
           (T      (RPLACB (peg? peg) new-tower))
    )
)
```

Anhang 2

Dokumentation der für den Produktionssysteminterpreter benutzten LISP-Funktionen

Fast alle LISP-Funktionen lassen sich über die Register in Stoyan und Görz (1984, S. 327–345) in ihrer Bedeutung identifizieren. Zusätzlich geben wir noch Informationen bezüglich einiger weniger Funktionen. Die Reihenfolge bezieht sich auf ihr Vorkommen im LISP-Code.

. De s. a.: Stoyan und Görz, Kap. 7.3 bezüglich des Schlüsselworts „&opt".

. UNWIND-PROTECT hat die Syntax:
(UNWIND-PROTECT ⟨form 1⟩ {⟨form 2⟩}) wobei {} Null oder mehrmalige Wiederholung bedeutet;
und die Semantik:
„UNWIND-PROTECT is a „super" catcher to guard against THROW's that might otherwise throw too far, too fast. ⟨Form 1⟩ is evaluated then {⟨form 2⟩} is evaluated. The value returned is the value of the last expression in {⟨form 2⟩}. This effect is equivalent to: (progn ⟨form 1⟩ {⟨Form 2⟩}). However, if during the evaluation of ⟨form 1⟩ we attempt to throw to a catch label outside the unwind-protect then {⟨form 2⟩} is guaranteed to be evaluated before the catch is completed." (TLC-Handbuch, 1984, Part III, Rev. 1.4, S. 30)

. RPLACB hat die Syntax:
(RPLACB ⟨sexpr 1⟩ ⟨sexpr 2⟩) und die Semantik:
„replaces the CAR-part of ⟨sexpr 1⟩ with the CAR-part of ⟨sexpr 2⟩ and the CDR-part of ⟨sexpr 1⟩ is replaced with the CDR-part of ⟨sexpr 2⟩. ⟨sexpr 1⟩ and ⟨sexpr 2⟩ must both the nonempty lists or dotted pairs." (TLC-Handbuch, 1984, III, S. 54)
. SECOND = CADR, THIRD = CADDR, FOURTH = CADDDR
. NTH hat Syntax:
(NTH ⟨list⟩ ⟨num⟩) und Semantik:
„NTH returns ⟨num⟩-th element of ⟨list⟩." (TLC-Handbuch, 1984, III, S. 49)

Anhang 3

Ein Ablaufprotokoll mit randomisierter Regelauswahl

```
))) (load "a:hanoi")
tower-above?
))) (hanoi '(1 2 3) t)
rule1  is tried !
rule1  is tried !
rule1  is tried !
rule1  is tried !
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((3 a c) (1 2 3) nil nil)
rule4  is tried !
rule4  is fired !
***working-memory*** is:  ((2 a b) (1 2 3) nil nil)
rule4  is tried !
rule4  is fired !
***working-memory*** is:  ((1 a c) (1 2 3) nil nil)
rule4  is tried !
rule1  is tried !
rule4  is tried !
rule2  is tried !
rule2  is tried !
rule2  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) (2 3) nil (1))
rule1  is tried !
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((3 a c) (2 3) nil (1))
rule1  is tried !
rule4  is tried !
rule4  is fired !
***working-memory*** is:  ((2 a b) (2 3) nil (1))
rule2  is tried !
rule2  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) (3) (2) (1))
rule2  is tried !
rule2  is fired !
```

```
***working-memory*** is:  ((3 a c) (3) (2) (1))
rule4  is tried !
rule4  is fired !
***working-memory*** is:  ((1 c b) (3) (2) (1))
rule1  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) (3) (1 2) nil)
rule4  is tried !
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((3 a c) (3) (1 2) nil)
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) nil (1 2) (3))
rule1  is tried !
rule1  is tried !
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((2 b c) nil (1 2) (3))
rule1  is tried !
rule4  is tried !
rule4  is fired !
***working-memory*** is:  ((1 b a) nil (1 2) (3))
rule2  is tried !
rule2  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) (1) (2) (3))
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((2 b c) (1) (2) (3))
rule1  is tried !
rule4  is tried !
rule2  is tried !
rule2  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) (1) nil (2 3))
rule1  is tried !
rule1  is tried !
rule1  is tried !
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((1 a c) (1) nil (2 3))
rule1  is tried !
rule4  is tried !
rule3  is tried !
rule3  is fired !
***working-memory*** is:  ((nil nil nil) nil nil (1 2 3))
rule2  is tried !
rule2  is fired !
***working-memory*** is:  ((1 c c) nil nil (1 2 3))
rule4  is tried !
rule4  is tried !
rule4  is tried !
rule4  is tried !
rule1  is tried !
rule1  is fired !
"the problem is solved !"
```

Inhalt

Modelle der Wissensrepräsentation (Gedächtnis und Wissen – Semantische Netzwerke – Produktionssysteme – Analoge Repräsentation) **Erwerb von Wissen** (Theoretische Ansätze – Wissensentwicklung und -erwerb) **Wissen und Wahrnehmung** (Zur Rolle des Wissens in der Wahrnehmung) **Wissen, Denken und Handeln** (Begriffliches Denken – Wissen und Problemlösen – Wissen und Verhaltensregulation) **Wissen und Sprache** (Textproduktion – Textverstehen) **Wissen im sozialen und kulturellen Kontext** (Wissen und Kultur) **Forschungsmethoden der Wissenspsychologie** (psychologische Methoden – Augenbewegungen – Qualitative Wissensdiagnose – Daten- und Zielorientierte Regelsysteme) **Wissen aus der Sicht der Neurowissenschaften** (Neurobiologische Grundlagen – neuronale Netzwerke) **Wissen aus der Sicht der künstlichen Intelligenz** (Wissensrepräsentation in künstlichen symbolverarbeitenden Systemen) **Angewandte Wissenspsychologie** (Medien – Intelligenter computerunterstützter Unterricht)

Autorinnen und Autoren:

H. Aebli – G. Dirlich – D. Dörner – Ch. Freksa – A.D. Friederici H.F. Friedrich – U. Furbach – Th. Herrmann – S. Hoppe-Graff – A. Hron – L.J. Issing – F. Klix – R.H. Kluwe – A. Lesgold – G. Lüer – H. Mandl – C. Möbus – R. Oerter – K. Opwis – G. Palm – W. Putz-Osterloh – W. Schnotz – H. Spada – G. Steiner – S.-O. Tergan – M. Waldmann – F.E. Weinert – K.F. Wender – F. Wilkening

ISBN 3-621-27016-7

Psychologie Verlags Union